Package: PhotoGEA (via r-universe)

May 25, 2026

Version 1.4.0

Date 2025-08-25

Title Photosynthetic Gas Exchange Analysis

Description Read, process, fit, and analyze photosynthetic gas exchange measurements. Documentation is provided by several vignettes; also see Lochocki, Salesse-Smith, & McGrath (2025) <doi:10.1111/pce.15501>.

Depends R (>= 3.6.0)

Imports openxlsx, lattice, dfoptim, DEoptim

Suggests knitr, rmarkdown, plantecophys, testthat (>= 3.0.0)

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/eloch216/PhotoGEA, https://eloch216.github.io/PhotoGEA/

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

Repository https://eloch216.r-universe.dev

Date/Publication 2025-08-28 20:09:03 UTC

RemoteUrl https://github.com/eloch216/photogea

RemoteRef HEAD

RemoteSha 947709bed9fa1e6cd71cd9a91d771b5e47fd22c8

Contents

`apply_gm` *Calculate CO2 concentration in the chloroplast or mesophyll*

Description

Calculates CO2 concentration in the chloroplast or mesophyll, the CO2 drawdown across the stomata, and the CO2 drawdown across the mesophyll. This function can accomodate alternative column names for the variables taken from the Licor file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
apply_gm(
  exdf_obj,
  gmc_at_25 = '',
  photosynthesis_type = 'C3',
  calculate_drawdown = TRUE,
  a_column_name = 'A',
  ca_column_name = 'Ca',
  ci_column_name = 'Ci',
  gmc_norm_column_name = 'gmc_norm',
  total_pressure_column_name = 'total_pressure',
  perform_checks = TRUE,
  return_exdf = TRUE
)
```

Arguments

`exdf_obj`	An `exdf` object, typically representing data from a Licor gas exchange measurement system.
`gmc_at_25`	The mesophyll conductance to CO2 diffusion at 25 degrees C, expressed in `mol m^(-2) s^(-1) bar^(-1)`. In the absence of other reliable information, `gmc_at_25` is often assumed to be infinitely large. If `gmc_at_25` is not a number, then there must be a column in `exdf_obj` called `gmc_at_25` with appropriate units. A numeric value supplied here will overwrite the values in the `gmc_at_25` column of `exdf_obj` if it exists.
`photosynthesis_type`	A string indicating the type of photosynthesis being considered (either `'C3'` or `'C4'`).

calculate_drawdown
: A logical value indicating whether to calculate drawdown values.

a_column_name
: The name of the column in exdf_obj that contains the net assimilation in micromol m^(-2) s^(-1).

ca_column_name
: The name of the column in exdf_obj that contains the ambient CO2 concentration in the chamber in micromol mol^(-1).

ci_column_name
: The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).

gmc_norm_column_name
: The name of the column in exdf_obj that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).

total_pressure_column_name
: The name of the column in exdf_obj that contains the total pressure in bar.

perform_checks
: A logical value indicating whether to check units for the required columns. This should almost always be TRUE. The option to disable these checks is only intended to be used when fit_c3_aci calls this function, since performing these checks many times repeatedly slows down the fitting procedure.

return_exdf
: A logical value indicating whether to return an exdf object. This should almost always be TRUE. The option to return a vector is mainly intended to be used when fit_c3_aci calls this function, since creating an exdf object to return will slow down the fitting procedure.

Details

For a C3 plant, the mesophyll conductance to CO2 (gmc) is said to be the conductance satisfying the following one-dimensional flux-conductance equation:

(1) An = gmc * (PCi - PCc)

where An is the net CO2 assimilation rate, PCi is the partial pressure of CO2 in the intercellular spaces, and PCc is the partial pressure of CO2 in the chloroplast. A key underlying assumption for this equation is that the flow of CO2 has reached a steady state; in this case, the flow across the stomata is equal to the flow across the mesophyll.

This equation can be rearranged to calculate PCc:

(2) PCc = PCi - An / gmc

This version of the equation can be found in many places, for example, as Equation 4 in Sharkey et al. "Fitting photosynthetic carbon dioxide response curves for C3 leaves" Plant, Cell & Environment 30, 1035–1040 (2007) [doi:10.1111/j.13653040.2007.01710.x].

It is common to express the partial pressures in microbar and the assimilation rate in micromol m^(-2) s^(-1); in this case, the units of mesophyll conductance become mol m^(-2) s^(-1) bar^(-1).

Licor measurement systems provide CO2 levels as relative concentrations with units of parts per million (ppm), or equivalently, micromol mol^(-1). Concentrations and partial pressures are related by the total gas pressure according to:

(3) partial_pressure = total_pressure * relative_concentration

Thus, it is also possible to calculate the CO2 concentration in the choloroplast (Cc) using the following equation:

(4) `Cc = Ci - An / (gmc * P)`

where `Ci` is the intercellular CO2 concentration and `P` is the total pressure. In this function, Equation (4) is used to calculate `Cc`, where the total pressure is given by the sum of the atmospheric pressure and the chamber overpressure.

When a plant is photosynthesizing, it draws CO2 into its chloroplasts, and this flow is driven by a concentration gradient. In other words, as CO2 flows from the ambient air across the stomata to the intercellular spaces and then across the mesophyll into the chloroplast, there is a decrease in CO2 concentration at each step. Sometimes it is useful to calculate these changes, which are usually referred to as "CO2 drawdown" values. So, in addition to `Ci`, this function (optionally) calculates the drawdown of CO2 across the stomata (`drawndown_cs = Ca - Ci`) and the drawdown of CO2 across the mesophyll (`drawdown_cm = Ci - Cc`).

Note: mesophyll conductance is not specified in typical Licor files, so it usually must be added using `set_variable` before calling `apply_gm`.

For a C4 plant, mesophyll conductance instead refers to the conductance associated with the flow of CO2 from the intercellular spaces into the mesophyll (rather than into the chloroplast). In this case, the equations above just require a small modification where `Pcc` and `Cc` are replaced by `PCm` and `Cm`, the partial pressure and concentration of CO2 in the mesophyll.

Value

The return value depends on the value of `return_exdf`:

- If `return_exdf` is `TRUE`, the return value is an `exdf` object based on `exdf_obj` with the following columns, calculated as described above: `Pci` and `Ci` (for C3 plants) or `PCm` and `Cm` (for C4 plants), `drawndown_s`, and `drawdown_cm`. The category for each of these new columns is `apply_gm` to indicate that they were created using this function.
- If `return_exdf` is `FALSE`, the return value is a list with a single named element (`internal_c`), which contains values of `Cc` or `Cm` as a numeric vector.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent parameter values, including gmc_norm
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_sharkey)

# Calculate Cc and drawdowns assuming a mesophyll conductance of
# 1 mol / m^2 / s / bar at 25 degrees C
licor_file <- apply_gm(licor_file, 1)

licor_file$units$Cc      # View the units of the new `Cc` column
licor_file$categories$Cc # View the category of the new `Cc` column
licor_file[, 'Cc']       # View the values of the new `Cc` column
```

as.data.frame.exdf *Convert an exdf object to a data frame*

Description

Converts an exdf object to a data frame by appending the units and categories to the top of each column in the exdf object's main_data data frame. Typically this function is used for displaying the contents of an exdf object; in fact, it is used internally by View, write.csv, and other functions. The main_data of an exdf object x can be accessed directly (without including the units and categories in the first row) via x[['main_data']] as with any other list element.

Usage

```
  ## S3 method for class 'exdf'
as.data.frame(x, ...)
```

Arguments

x	An exdf object.
...	Unused.

Value

A data frame formed from x.

See Also

exdf

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))
as.data.frame(simple_exdf) # Includes units and categories in the first rows
simple_exdf[['main_data']] # Just returns the main data
```

barchart_with_errorbars *Barcharts with error bars*

Description

barchart_with_errorbars is a wrapper for lattice::barchart that includes error bars on the chart, while bwplot_wrapper is a simple wrapper for lattice::bwplot that gives it the same function signature as barchart_with_errorbars.

Usage

```
barchart_with_errorbars(
  Y,
  X,
  eb_width = 0.2,
  eb_lwd = 1,
  eb_col = 'black',
  na.rm = TRUE,
  remove_outliers = FALSE,
  ...
)

bwplot_wrapper(Y, X, ...)
```

Arguments

`Y`	A numeric vector.
`X`	A vector with the same length as `Y` that can be used as a factor to split `Y` into one or more distinct subsets.
`eb_width`	The width of the error bars.
`eb_lwd`	The line width (thickness) of the error bars.
`eb_col`	The color of the error bars.
`na.rm`	A logical value indicating whether or not to remove NA values before calculating means and standard errors.
`remove_outliers`	A logical value indicating whether or not to remove outliers using `exclude_outliers` before calculating means and standard errors.
`...`	Additional arguments to be passed to `lattice::barchart` or `lattice::bwplot`.

Details

The `barchart_with_errorbars` function uses `tapply` to calculate the mean and standard error for each subset of `Y` as determined by the values of `X`. In other words, `means <- tapply(Y, X, mean)`, and similar for the standard errors. The mean values are represented as bars in the final plot, while the standard error is used to create error bars located at `mean +/- standard_error`.

The `bwplot_wrapper` function is a simple wrapper for `lattice::bwplot` that gives it the same input arguments as `barchart_with_errorbars`. In other words, the same `X` and `Y` vectors can be used to create a barchart using `barchart_with_errorbars` or a box-whisker plot with `bwplot_wrapper`.

Value

A `trellis` object created by `lattice::barchart` or `lattice::bwplot`.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Plot the average assimilation value for each species. (Note: this is not a
# meaningful calculation since we are combining assimilation values measured
# at different PPFD.)
barchart_with_errorbars(
  licor_file[, 'A'],
  licor_file[, 'species'],
  ylim = c(0, 50),
  xlab = 'Species',
  ylab = paste0('Net assimilation (', licor_file$units$A, ')')
)

# Make a box-whisker plot using the same data. (Note: this is not a meaningful
# plot since we are combining assimilation values measured at different PPFD.)
bwplot_wrapper(
  licor_file[, 'A'],
  licor_file[, 'species'],
  ylim = c(0, 50),
  xlab = 'Species',
  ylab = paste0('Net assimilation (', licor_file$units$A, ')')
)

# Another way to create the plots. This method illustrates the utility of the
# bwplot_wrapper function.
plot_parameters <- list(
  Y = licor_file[, 'A'],
  X = licor_file[, 'species'],
  ylim = c(0, 50),
  xlab = 'Species',
  ylab = paste0('Net assimilation (', licor_file$units$A, ')')
)
do.call(barchart_with_errorbars, plot_parameters)
do.call(bwplot_wrapper, plot_parameters)
```

---

basic_stats *Calculate basic stats (mean and standard error)*

Description

Calculates basic stats (mean and standard error) for each applicable column in an `exdf` object split up according to the values of one or more identifier columns.

Usage

```
basic_stats(
  exdf_obj,
  identifier_columns,
  na.rm = TRUE
)
```

Arguments

exdf_obj	An exdf object.
identifier_columns	The name(s) of one or more columns in a vector or list that can be used to split exdf_obj into chunks.
na.rm	A logical value indicating whether or not to remove NA values before calculating means and standard errors.

Details

This function first splits up exdf_obj into chunks according to the values of the identifier_columns. For each chunk, columns that have a single unique value are identified and excluded from the statistical calculations. For the remaining numeric columns, the mean and standard error are calculated.

Value

An exdf object including the mean and standard error for each applicable column, where each row represents one value of the identifier_columns. The column names are determined by appending '_avg' and '_stderr' to the original names.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Calculate the average assimilation and stomatal conductance values for each
# species. (Note: this is not a meaningful calculation!)
basic_stats(
  licor_file[ , c('species', 'K', 'A', 'gsw'), TRUE],
  'species'
)
```

by.exdf *Apply a function to an exdf object split by one or more factors*

Description

Divides an exdf object into groups defined by one or more factors and applies a function to each group.

Usage

```
  ## S3 method for class 'exdf'
by(data, INDICES, FUN, ...)
```

Arguments

data	An exdf object.
INDICES	A factor or a list of factors.
FUN	A function whose first input argument is an exdf object.
...	Additional arguments to be passed to FUN.

Value

Splits data into chunks x by the values of the INDICES and calls FUN(x, ...) for each chunk; returns a list where each element is the output from each call to FUN.

See Also

exdf

Examples

```
# Read a Licor file, split it into chunks according to the `species` column,
# and count the number of measurements for each species
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

by(licor_file, licor_file[, 'species'], nrow)
```

`c3_temperature_param_bernacchi` *C3 temperature response parameters from Bernacchi et al.*

Description

Parameters describing the temperature response of important C3 photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
c3_temperature_param_bernacchi
```

Format

List with 12 named elements that each represent a variable whose temperature-dependent value can be calculated using an Arrhenius equation, Johnson-Eyring-Williams equation, or a polynomial equation:

- `Gamma_star_at_25`: The value of chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation (`Gamma_star`) at 25 degrees C.
- `Gamma_star_norm`: `Gamma_star` normalized to its value at 25 degrees C.
- `gmc_norm`: The mesophyll conductance to CO2 diffusion (`gmc`) normalized to its value at 25 degrees C.
- `J_norm`: The electron transport rate (`J`) normalized to its value at 25 degrees C.
- `Kc_at_25`: The Michaelis-Menten constant for rubisco carboxylation (`Kc`) at 25 degrees C.
- `Kc_norm`: `Kc` normalized to its value at 25 degrees C.
- `Ko_at_25`: The Michaelis-Menten constant for rubisco oxygenation (`Ko`) at 25 degrees C.
- `Ko_norm`: `Ko` normalized to its value at 25 degrees C.
- `RL_norm`: The rate of non-photorespiratory CO2 release in the light (`RL`) normalized to its value at 25 degrees C.
- `Tp_norm`: The maximum rate of triose phosphate utilization (`Tp`) normalized to its value at 25 degrees C.
- `Vcmax_norm`: The maximum rate of rubisco carboxylation (`Vcmax`) normalized to its value at 25 degrees C.
- `Vomax_norm`: The maximum rate of rubisco oxygenation (`Vomax`) normalized to `Vcmax` at 25 degrees C.

In turn, each of these elements is a list with at least 2 named elements:

- `type`: the type of temperature response
- `units`: the units of the corresponding variable.

Source

Many of these parameters are normalized to their values at 25 degrees C. `Vomax` is normalized to the value of `Vcmax` at 25 degrees C. These variables include `_norm` in their names to indicate this.

Arrhenius parameters for `J` were obtained from Bernacchi et al. (2003). Here, we use the values determined from chlorophyll fluorescence measured from plants grown at 25 degrees C (Table 1). Although Bernacchi et al. (2003) reports values of `Jmax`, here we assume that both `Jmax` and the light-dependent values of `J` follow the same temperature response function and refer to it as `J` for compatibility with `c3_temperature_param_sharkey`.

Johnson-Eyring-Williams parameters for `gmc` were obtained from Bernacchi et al. (2002).

The Bernacchi papers from the early 2000s do not specify a temperature response for `Tp`, so we instead use the Johnson-Eyring-Williams response from Sharkey et al. (2007). Another option would be to use a flat temperature response; in other words, to assume that `Tp` is constant with temperature. This could be done with the following code, which takes the flat response parameters from `c3_temperature_param_flat`: `within(c3_temperature_param_bernacchi, {Tp_norm = c3_temperature_param_flat$Tp_norm})`

The Arrhenius parameters for the other variables were obtained from Bernacchi et al. (2001).

References:

- Bernacchi, C. J., Singsaas, E. L., Pimentel, C., Jr, A. R. P. & Long, S. P. "Improved temperature response functions for models of Rubisco-limited photosynthesis" Plant, Cell & Environment 24, 253–259 (2001) [doi:10.1111/j.13653040.2001.00668.x].
- Bernacchi, C. J., Portis, A. R., Nakano, H., von Caemmerer, S. & Long, S. P. "Temperature Response of Mesophyll Conductance. Implications for the Determination of Rubisco Enzyme Kinetics and for Limitations to Photosynthesis in Vivo" Plant Physiology 130, 1992–1998 (2002) [doi:10.1104/pp.008250].
- Bernacchi, C. J., Pimentel, C. & Long, S. P. "In vivo temperature response functions of parameters required to model RuBP-limited photosynthesis" Plant, Cell & Environment 26, 1419–1430 (2003) [doi:10.1046/j.00168025.2003.01050.x].
- Sharkey, T. D., Bernacchi, C. J., Farquhar, G. D. & Singsaas, E. L. "Fitting photosynthetic carbon dioxide response curves for C3 leaves" Plant, Cell & Environment 30, 1035–1040 (2007) [doi:10.1111/j.13653040.2007.01710.x].

`c3_temperature_param_flat`

C3 temperature response parameters for a flat response

Description

Parameters that specify a flat temperature response (in other words, no dependence on temperature) for important C3 photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
c3_temperature_param_flat
```

Format

List with 11 named elements that each represent a variable whose temperature-dependent value can be calculated using an Arrhenius equation or a polynomial equation:

- `Gamma_star_at_25`: The value of chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation (`Gamma_star`) at 25 degrees C.
- `Gamma_star_norm`: `Gamma_star` normalized to its value at 25 degrees C.
- `gmc_norm`: The mesophyll conductance to CO2 diffusion (`gmc`) normalized to its value at 25 degrees C.
- `J_norm`: The electron transport rate (`J`) normalized to its value at 25 degrees C.
- `Kc_at_25`: The Michaelis-Menten constant for rubisco carboxylation (`Kc`) at 25 degrees C.
- `Kc_norm`: `Kc` normalized to its value at 25 degrees C.
- `Ko_at_25`: The Michaelis-Menten constant for rubisco oxygenation (`Ko`) at 25 degrees C.
- `Ko_norm`: `Ko` normalized to its value at 25 degrees C.
- `RL_norm`: The rate of non-photorespiratory CO2 release in the light (`RL`) normalized to its value at 25 degrees C.
- `Tp_norm`: The maximum rate of triose phosphate utilization (`Tp`) normalized to its value at 25 degrees C.
- `Vcmax_norm`: The maximum rate of rubisco carboxylation (`Vcmax`) normalized to its value at 25 degrees C.

In turn, each of these elements is a list with at least 2 named elements:

- `type`: the type of temperature response
- `units`: the units of the corresponding variable.

Source

Many of these parameters are normalized to their values at 25 degrees C. These variables include `_norm` in their names to indicate this.

Here, the activation energy values (`Ea`) are all set to 0, which means that the values will not depend on temperature. Some parameters are specified at 25 degrees C; these values were obtained from Sharkey et al. (2007). (See `c3_temperature_param_sharkey`.)

References:

- Sharkey, T. D., Bernacchi, C. J., Farquhar, G. D. & Singsaas, E. L. "Fitting photosynthetic carbon dioxide response curves for C3 leaves" Plant, Cell & Environment 30, 1035–1040 (2007) [doi:10.1111/j.13653040.2007.01710.x].

`c3_temperature_param_sharkey` *C3 temperature response parameters from Sharkey et al.*

Description

Parameters describing the temperature response of important C3 photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
c3_temperature_param_sharkey
```

Format

List with 11 named elements that each represent a variable whose temperature-dependent value can be calculated using an Arrhenius equation or a polynomial equation:

- `Gamma_star_at_25`: The value of chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation (`Gamma_star`) at 25 degrees C.
- `Gamma_star_norm`: `Gamma_star` normalized to its value at 25 degrees C.
- `gmc_norm`: The mesophyll conductance to CO2 diffusion (`gmc`) normalized to its value at 25 degrees C.
- `J_norm`: The electron transport rate (`J`) normalized to its value at 25 degrees C.
- `Kc_at_25`: The Michaelis-Menten constant for rubisco carboxylation (`Kc`) at 25 degrees C.
- `Kc_norm`: `Kc` normalized to its value at 25 degrees C.
- `Ko_at_25`: The Michaelis-Menten constant for rubisco oxygenation (`Ko`) at 25 degrees C.
- `Ko_norm`: `Ko` normalized to its value at 25 degrees C.
- `RL_norm`: The rate of non-photorespiratory CO2 release in the light (`RL`) normalized to its value at 25 degrees C.
- `Tp_norm`: The maximum rate of triose phosphate utilization (`Tp`) normalized to its value at 25 degrees C.
- `Vcmax_norm`: The maximum rate of rubisco carboxylation (`Vcmax`) normalized to its value at 25 degrees C.

In turn, each of these elements is a list with at least 2 named elements:

- `type`: the type of temperature response
- `units`: the units of the corresponding variable.

Source

Many of these parameters are normalized to their values at 25 degrees C. These variables include `_norm` in their names to indicate this.

Response parameters were obtained from Sharkey et al. (2007). In this publication, gas concentrations are expressed as partial pressures (in `Pa` or `kPa`) rather than mole fractions (`micromol / mol` or `mmol / mol`). However, for consistency with `c3_temperature_param_bernacchi`, here we prefer to use mole fractions.

To convert a concentration expressed as a partial pressure (`P`; in `Pa`) to a concentration expressed as a mole fraction (`C`; in `micromol / mol`), we need a value for atmospheric pressure; we will use the typical value of `101325 Pa`. Then `C = P / 101325 * 1e6` or `C = P * cf`, where `cf = 1e6 / 101325` is a conversion factor. The same correction can be used to convert `kPa` to `mmol / mol`. The value of `cf` can be accessed using `PhotoGEA:::c_pa_to_ppm`.

References:

- Sharkey, T. D., Bernacchi, C. J., Farquhar, G. D. & Singsaas, E. L. "Fitting photosynthetic carbon dioxide response curves for C3 leaves" Plant, Cell & Environment 30, 1035–1040 (2007) [doi:10.1111/j.13653040.2007.01710.x].

`c4_temperature_param_flat` *C4 temperature response parameters for a flat response*

Description

Parameters that specify a flat temperature response (in other words, no dependence on temperature) for important C4 photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
c4_temperature_param_flat
```

Format

List with 10 named elements that each represent a variable whose temperature-dependent value can be calculated using either an Arrhenius or Gaussian equation:

- `Vcmax_norm`: The maximum rate of rubisco carboxylation (`Vcmax`) normalized to its value at 25 degrees C.
- `Vpmax_norm`: The maximum rate of PEP carboxylase activity (`Vpmax`) normalized to its value at 25 degrees C.
- `RL_norm`: The respiration rate (`RL`) normalized to the value of `Vcmax` at 25 degrees C.
- `Kc`: The Michaelis-Menten constant for rubisco carboxylation.
- `Ko`: The Michaelis-Menten constant for rubisco oxygenation.

- Kp: The Michaelis-Menten constant of PEP carboxylase.
- gamma_star: Half the reciprocal of rubisco specificity.
- ao: The ratio of solubility and diffusivity of O2 to CO2.
- gmc_norm: The mesophyll conductance to CO2 diffusion normalized to its value at 25 degrees C.
- J_norm: The electron transport rate J normalized to its value at 25 degrees C.

Each of these is a list with 4 named elements:

- type: the type of temperature response (`'Arrhenius'`)
- c: the (dimensionless) Arrhenius scaling factor.
- Ea: the activation energy in kJ / mol.
- units: the units of the corresponding variable.

Source

Some of these parameters (Vcmax, Vpmax, RL, gmc, and J) are normalized to their values at 25 degrees C. These variables include _norm in their names to indicate this.

The remaining parameters (Kc, Ko, Kp, gamma_star, ao, and gmc) are not normalized because they are assumed to not vary significantly between species.

Here, the activation energy values (Ea) are all set to 0, which means that the values will not depend on temperature. The Arrhenius scaling factors c are chosen to reproduce the parameter values at 25 degrees C as specified in von Caemmerer (2021). (See c4_temperature_param_vc.)

References:

- von Caemmerer, S. "Updating the steady-state model of C4 photosynthesis" Journal of Experimental Botany 72, 6003–6017 (2021) [doi:10.1093/jxb/erab266].

---

c4_temperature_param_vc

C4 temperature response parameters from von Caemmerer

Description

Temperature response parameters describing the temperature response of important C4 photosynthetic parameters, intended to be passed to the calculate_temperature_response function.

Usage

```
c4_temperature_param_vc
```

Format

List with 10 named elements that each represent a variable whose temperature-dependent value can be calculated using either an Arrhenius or Gaussian equation:

- `Vcmax_norm`: The maximum rate of rubisco carboxylation (`Vcmax`) normalized to its value at 25 degrees C.
- `Vpmax_norm`: The maximum rate of PEP carboxylase activity (`Vpmax`) normalized to its value at 25 degrees C.
- `RL_norm`: The respiration rate (`RL`) normalized to the value of `Vcmax` at 25 degrees C.
- `Kc`: The Michaelis-Menten constant for rubisco carboxylation.
- `Ko`: The Michaelis-Menten constant for rubisco oxygenation.
- `Kp`: The Michaelis-Menten constant of PEP carboxylase.
- `gamma_star`: Half the reciprocal of rubisco specificity.
- `ao`: The ratio of solubility and diffusivity of O2 to CO2.
- `gmc_norm`: The mesophyll conductance to CO2 diffusion normalized to its value at 25 degrees C.
- `J_norm`: The electron transport rate `J` normalized to its value at 25 degrees C.

The `J_norm` parameter is calculated using a Gaussian function and hence its corresponding list element is itself a list with 4 named elements:

- `type`: the type of temperature response (`'Gaussian'`)
- `optimum_rate`: the largest value this parameter can take.
- `t_opt`: the temperature where the optimum occurs in `degrees C`.
- `sigma`: the width of the Gaussian in `degrees C`.
- `units`: the units of the corresponding variable.

Each of the remaining elements is a list with 4 named elements:

- `type`: the type of temperature response (`'Arrhenius'`)
- `c`: the (dimensionless) Arrhenius scaling factor.
- `Ea`: the activation energy in `kJ / mol`.
- `units`: the units of the corresponding variable.

Source

Some of these parameters (`Vcmax`, `Vpmax`, `RL`, `gmc`, and `J`) are normalized to their values at 25 degrees C. These variables include `_norm` in their names to indicate this.

The remaining parameters (`Kc`, `Ko`, `Kp`, `gamma_star`, and `ao`) are not normalized because they are assumed to not vary significantly between species.

Here, the Arrhenius scaling factors (`c`; dimensionless) and activation energy values (`Ea`; kJ / mol) are obtained from von Caemmerer (2021). In that publication, the overall scaling for each parameter is specified by its value at 25 degrees C; the scaling factors are determined from this information as described in the documentation for `calculate_temperature_response_arrhenius`.

The Gaussian parameters (`t_opt` and `sigma`) for `J_norm` are also obtained from von Caemmerer (2021), assuming that `J` and `Jmax` follow the same temperature response. The value of `optimum_rate` is chosen such that `J_norm` is equal to 1 at a temperature of 25 degrees C.

References:

- von Caemmerer, S. "Updating the steady-state model of C4 photosynthesis" Journal of Experimental Botany 72, 6003–6017 (2021) [doi:10.1093/jxb/erab266].

`calculate_ball_berry_index` *Calculate the Ball-Berry index*

Description

Calculates the Ball-Berry index. This function can accomodate alternative column names for the variables taken from the Licor file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_ball_berry_index(
  data_table,
  a_column_name = 'A',
  rhleaf_column_name = 'RHleaf',
  csurface_column_name = 'Csurface'
)
```

Arguments

`data_table`	A table-like R object such as a data frame or an `exdf`.
`a_column_name`	The name of the column in `data_table` that contains the net assimilation in `micromol m^(-2) s^(-1)`.
`rhleaf_column_name`	The name of the column in `data_table` that contains the relative humidity at the leaf surface in `%`.
`csurface_column_name`	The name of the column in `data_table` that contains the CO2 concentration at the leaf surface in `micromol mol^(-1)`.

Details

The Ball-Berry index is defined as `A * h_s / c_s`, where `A` is the net assimilation rate, `h_s` is the relative humidity at the leaf surface, and `c_s` is the CO2 concentration at the leaf surface. This variable is a key part of the Ball-Berry model, which assumes that stomatal conductance is linearly related to the Ball-Berry index. For more information, please see the original publication describing the model: Ball, J. T., Woodrow, I. E. and Berry, J. A. "A Model Predicting Stomatal Conductance

and its Contribution to the Control of Photosynthesis under Different Environmental Conditions." in "Progress in Photosynthesis Research: Volume 4" (1986) [doi:10.1007/9789401705196_48].

Typically, the relative humidity and CO2 concentration at the leaf surface are not included in Licor output files. Instead, the output files only include the relative humidity and CO2 concentration in the sample chamber, and conditions at the leaf surface may be slightly different. These required inputs can be calculated using the `calculate_gas_properties` function.

Value

An object based on `data_table` that includes the Ball-Berry index as a new column called `bb_index`.

If `data_table` is an `exdf` object, the category of this new column will be `calculate_ball_berry_index` to indicate that it was created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package, calculate the
# total pressure, calculate additional gas properties, and finally calculate the
# Ball-Berry index.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_gas_properties(licor_file)

licor_file <- calculate_ball_berry_index(licor_file)

licor_file$units$bb_index      # View the units of the new `bb_index` column
licor_file$categories$bb_index # View the category of the new `bb_index` column
licor_file[,'bb_index']        # View the values of the new `bb_index` column
```

`calculate_c3_assimilation`

Calculate C3 assimilation rates

Description

Calculates C3 assimilation rates based on the Farquhar-von-Caemmerer-Berry model. This function can accomodate alternative colum names for the variables taken from Licor files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c3_assimilation(
  data_table,
  alpha_g,
```

```
    alpha_old,
    alpha_s,
    alpha_t,
    Gamma_star_at_25,
    J_at_25,
    Kc_at_25,
    Ko_at_25,
    RL_at_25,
    Tp_at_25,
    Vcmax_at_25,
    Wj_coef_C = 4.0,
    Wj_coef_Gamma_star = 8.0,
    cc_column_name = 'Cc',
    gamma_star_norm_column_name = 'Gamma_star_norm',
    j_norm_column_name = 'J_norm',
    kc_norm_column_name = 'Kc_norm',
    ko_norm_column_name = 'Ko_norm',
    oxygen_column_name = 'Oxygen',
    rl_norm_column_name = 'RL_norm',
    total_pressure_column_name = 'total_pressure',
    tp_norm_column_name = 'Tp_norm',
    vcmax_norm_column_name = 'Vcmax_norm',
    hard_constraints = 0,
    perform_checks = TRUE,
    return_table = TRUE,
    ...
  )
```

Arguments

`data_table`	A table-like R object such as a data frame or an `exdf`.
`alpha_g`	A dimensionless parameter where `0 <= alpha_g <= 1`, representing the proportion of glycolate carbon taken out of the photorespiratory pathway as glycine. `alpha_g` is often assumed to be 0. If `alpha_g` is not a number, then there must be a column in `data_table` called `alpha_g` with appropriate units. A numeric value supplied here will overwrite the values in the `alpha_g` column of `data_table` if it exists.
`alpha_old`	A dimensionless parameter where `0 <= alpha_old <= 1`, representing the fraction of remaining glycolate carbon not returned to the chloroplast after accounting for carbon released as CO2. `alpha_old` is often assumed to be 0. If `alpha_old` is not a number, then there must be a column in `data_table` called `alpha_old` with appropriate units. A numeric value supplied here will overwrite the values in the `alpha_old` column of `data_table` if it exists.
`alpha_s`	A dimensionless parameter where `0 <= alpha_s <= 0.75 * (1 - alpha_g)` representing the proportion of glycolate carbon taken out of the photorespiratory pathway as serine. `alpha_s` is often assumed to be 0. If `alpha_s` is not a number, then there must be a column in `data_table` called `alpha_s` with appropriate units. A numeric value supplied here will overwrite the values in the

alpha_s column of data_table if it exists.

alpha_t
: A dimensionless parameter where 0 <= alpha_t <= 1 representing the proportion of glycolate carbon taken out of the photorespiratory pathway as CH2-THF. alpha_t is often assumed to be 0. If alpha_t is not a number, then there must be a column in data_table called alpha_t with appropriate units. A numeric value supplied here will overwrite the values in the alpha_t column of data_table if it exists.

Gamma_star_at_25
: The chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at 25 degrees C, expressed in micromol mol^(-1). If Gamma_star_at_25 is not a number, then there must be a column in data_table called Gamma_star_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Gamma_star_at_25 column of data_table if it exists.

J_at_25
: The electron transport rate at 25 degrees C, expressed in micromol m^(-2) s^(-1). Note that this is _not_ Jmax, and in general will depend on the incident photosynthetically active flux density. If J_at_25 is not a number, then there must be a column in data_table called J_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the J_at_25 column of data_table if it exists.

Kc_at_25
: The Michaelis-Menten constant for Rubisco carboxylation at 25 degrees C, expressed in micromol mol^(-1). If Kc_at_25 is not a number, then there must be a column in data_table called Kc_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Kc_at_25 column of data_table if it exists.

Ko_at_25
: The Michaelis-Menten constant for Rubisco oxygenation at 25 degrees C, expressed in mmol mol^(-1). If Ko_at_25 is not a number, then there must be a column in data_table called Ko_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Ko_at_25 column of data_table if it exists.

RL_at_25
: The respiration rate at 25 degrees C, expressed in micromol m^(-2) s^(-1). If RL_at_25 is not a number, then there must be a column in data_table called RL_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the RL_at_25 column of data_table if it exists.

Tp_at_25
: The maximum rate of triphosphate utilization at 25 degrees C, expressed in micromol m^(-2) s^(-1). If Tp_at_25 is not a number, then there must be a column in data_table called Tp_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Tp_at_25 column of data_table if it exists.

Vcmax_at_25
: The maximum rate of rubisco carboxylation at 25 degrees C, expressed in micromol m^(-2) s^(-1). If Vcmax_at_25 is not a number, then there must be a column in data_table called Vcmax_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Vcmax_at_25 column of data_table if it exists.

Wj_coef_C
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration.

`Wj_coef_Gamma_star`
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration.

`cc_column_name`
: The name of the column in `data_table` that contains the chloroplastic CO2 concentration in `micromol mol^(-1)`.

`gamma_star_norm_column_name`
: The name of the column in `data_table` that contains the normalized `Gamma_star` values (with units of `normalized to Gamma_star at 25 degrees C`).

`j_norm_column_name`
: The name of the column in `data_table` that contains the normalized `J` values (with units of `normalized to J at 25 degrees C`).

`kc_norm_column_name`
: The name of the column in `data_table` that contains the normalized `Kc` values (with units of `normalized to Kc at 25 degrees C`).

`ko_norm_column_name`
: The name of the column in `data_table` that contains the normalized `Ko` values (with units of `normalized to Ko at 25 degrees C`).

`oxygen_column_name`
: The name of the column in `data_table` that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be `percent`.

`rl_norm_column_name`
: The name of the column in `data_table` that contains the normalized `RL` values (with units of `normalized to RL at 25 degrees C`).

`total_pressure_column_name`
: The name of the column in `data_table` that contains the total pressure in `bar`.

`tp_norm_column_name`
: The name of the column in `data_table` that contains the normalized `Tp` values (with units of `normalized to Tp at 25 degrees C`).

`vcmax_norm_column_name`
: The name of the column in `data_table` that contains the normalized `Vcmax` values (with units of `normalized to Vcmax at 25 degrees C`).

`hard_constraints`
: An integer numerical value indicating which types of hard constraints to place on the values of input parameters; see below for more details.

`perform_checks`
: A logical value indicating whether to check units for the required columns. This should almost always be `TRUE`. The option to disable these checks is only intended to be used when `fit_c3_aci` calls this function, since performing these checks many times repeatedly slows down the fitting procedure.

`return_table`
: A logical value indicating whether to return an `exdf` object. This should almost always be `TRUE`. The option to return a vector is mainly intended to be used when `fit_c3_aci` calls this function, since creating an `exdf` object to return will slow down the fitting procedure.

`...`
: Optional arguments; see below.

Details

The Busch et al. (2018) and Busch (2020) model:

This function generally follows the Farquhar-von-Caemmerer-Berry model as described in Busch et al. (2018) and Busch (2020) with a few modifications described below. In this formulation, the steady-state net CO2 assimilation rate An is calculated according to

An = (1 - Gamma_star_agt / PCc) * Vc - RL,

where Gamma_star is the CO2 compensation point in the absence of non-photorespiratory CO2 release, Gamma_star_agt is the effective value of Gamma_star accounting for glycolate carbon remaining in the cytosol, PCc is the partial pressure of CO2 in the chloroplast, Vc is the RuBP carboxylation rate, and RL is the rate of non-photorespiratory CO2 release in the light. Gamma_star_agt is given by

Gamma_star_agt = (1 - alpha_g + 2 * alpha_t) * Gamma_star,

where alpha_g and alpha_t are the fractions of glycolate carbon leaving the photorespiratory pathway as glycine and CH2-THF, respectively.

The model considers three potential values of Vc that correspond to limitations set by three different processes: Rubisco activity, RuBP regeneration, and triose phopsphate utilization (TPU). The Rubisco-limited carboxylation rate Wc is given by

Wc = PCc * Vcmax / (PCc + Kc * (1.0 + POc / Ko)),

where Vcmax is the maximum rate of Rubisco carboxylation, Kc is the Michaelis-Menten constant for CO2, Ko is the Michaelis-Menten constant for O2, and POc is the partial pressure of O2 in the chloroplast.

The RuBP-regeneration-limited carboxylation rate Wj is given by

Wj = PCc * J / (4 * PCc + Gamma_star_agt * (8 + 16 * alpha_g - 8 * alpha_t + 8 * alpha_s)),

where J is the potential electron transport rate at a given light intensity and alpha_s is the fraction of glycolate carbon leaving the photorespiratory pathway as serine.

The TPU-limited carboxylation rate is given by

Wp = PCc * 3 * Tp / (PCc - Gamma_star_agt * (1 + 3 * alpha_g + 6 * alpha_t + 4 * alpha_s)),

where Tp is the maximum rate of triose phosphate utilization. Note that this equation only applies when PCc > Gamma_star_agt * (1 + 3 * alpha_g + 6 * alpha_t + 4 * alpha_s); for smaller values of PCc, TPU cannot limit the RuBP carboxylation rate and Wp = Inf. (Lochocki & McGrath, under review).

The actual carboxylation rate is typically chosen to be the smallest of the three potential rates:

Vc = min{Wc, Wj, Wp}.

In the equations above, several of the variables depend on the leaf temperature. In particular, the leaf-temperature-adjusted values of Gamma_star, J, Kc, Ko, RL, Tp, and Vcmax are determined from their base values at 25 degrees C and a temperature-dependent multiplicative factor.

Also note that PCc is calculated from the chloroplastic CO2 concentration Cc using the total pressure (ambient pressure + chamber overpressure).

In addition to the carboxylation and assimilation rates already mentioned, it is also possible to calculate the net CO2 assimilation rates determined by Rubisco activity, RuBP regeneration, and TPU as follows:

Ac = (1 - Gamma_star_agt / PCc) * Wc - RL

`Aj = (1 - Gamma_star_agt / PCc) * Wj - RL`

`Ap = (1 - Gamma_star_agt / PCc) * Wp - RL`

The Busch model with nitrogen restrictions:

Note that the implementation as described above does not currently facilitate the inclusion of nitrogen limitations (Equations 15-21 in Busch et al. (2018)).

The "old" model:

In an older version of the model, `alpha_g`, `alpha_s`, and `alpha_t` are replaced with a single parameter `alpha_old`. Most publications refer to this simply as `alpha`, but here we follow the notation of Busch et al. (2018) for clarity. In this version, there is no disctinction between `Gamma_star_agt` and `Gamma_star`. Other differences are described below.

The RuBP-regeneration-limited carboxylation rate `Wj` is given by

`Wj = PCc * J / (Wj_coef_C * PCc + Wj_coef_Gamma_star * Gamma_star)`,

Here we have allowed `Wj_coef_C` and `Wj_coef_Gamma_star` to be variables rather than taking fixed values (as they do in many sources). This is necessary because not all descriptions of the FvCB model use the same values, where the different values are due to different assumptions about the NADPH and ATP requirements of RuBP regeneration.

The TPU-limited carboxylation rate is given by

`Wp = PCc * 3 * Tp / (PCc - Gamma_star * (1 + 3 * alpha_old))`,

Note that this equation only applies when `PCc > Gamma_star * (1 + 3 * alpha_old)`; for smaller values of `PCc`, TPU cannot limit the RuBP carboxylation rate and `Wp = Inf`. (Lochocki & McGrath, under review).

Using either version of the model:

When using `calculate_c3_assimilation`, it is possible to use either version of the model. Setting `alpha_g`, `alpha_s`, and `alpha_t` to zero is equivalent to using the older version of the model, while setting `alpha_old = 0` is equivalent to using the newer version of the model. If all `alpha` parameters are zero, there is effectively no difference between the two versions of the model. Attempting to set a nonzero `alpha_old` if either `alpha_g`, `alpha_s`, or `alpha_t` is nonzero is forbidden since it would represent a mix between the two models; if such values are passed as inputs, then an error will be thrown.

Hard constraints:

Most input parameters to the FvCB model have hard constraints on their values which are set by their biochemical or physical interpretation; for example, `Vcmax` cannot be negative and `alpha_g` must lie between 0 and 1. Yet, because of measurement noise, sometimes it is necessary to use values outside these ranges when fitting an A-Ci curve with `fit_c3_aci` or `fit_c3_variable_j`. To accomodate different potential use cases, it is possible to selectively apply these hard constraints by specifying different values of the `hard_constraints` input argument:

- `hard_constraints = 0`: Constraints are only placed on inputs that are user-supplied and cannot be fit, such as `Oxygen`.
- `hard_constraints = 1`: Includes the same constraints as when `hard_constraints` is 0, with the additional constraint that all `Cc` values must be non-negative.
- `hard_constraints = 2`: Includes the same constraints as when `hard_constraints` is 1, which additional constraints on the parameters that can be fitted. For example, `Vcmax_at_25` must be non-negative and `alpha_g` must lie between 0 and 1.

If any input values violate any of the specified constraints, an error message will be thrown.

Optional arguments:

- **use_min_A**: If an input argument called `use_min_A` is supplied and its value is `TRUE`, then the "minimum assimilation" variant of the FvCB model will be used. In this case, `An` will be calculated as `An = min{Ac, Aj, Ap}`. In general, using this variant is not recommended.It should only be used to investigate errors that may occur when using the minimal assimilation rate rather than the minimal carboxylation rate.
- **TPU_threshold**: If an input argument called `TPU_threshold` is supplied and its numeric value is not `NULL`, then TPU limitations will only be allowed for values of `Cc` above this threshold. This threshold will be used in place of the values discussed in the equations above. In general, using this option is not recommended. It should only be used to investigate errors that may occur when using a fixed TPU threshold.
- **use_FRL**: If an input argument called `use_FRL` is supplied and its value is `TRUE`, then `An` will always be set to `Ac` for `Cc < Gamma_star_agt`. This "forced Rubisco limitation" can only be used along with the "minimum assimilation" variant (`use_min_A = TRUE`).
- **consider_depletion**: If an input argument called `consider_depletion` is supplied and its value is `TRUE`, then RuBP depletion will be considered to be an additional potential limiting process. In this case, `Vc` will be calculated as `Vc = min{Wc, Wj, Wp, Wd}`, where `Wd` is zero when `Cc < Gamma_star` and `Inf` otherwise. Note that the value of `Wd` (and `Ad = (1 - Gamma_star / PCc) * Wd - RL`) will always be returned, regardless of whether RuBP depletion is considered when calculating `An`.

References:

- Busch, Sage, & Farquhar, G. D. "Plants increase CO2 uptake by assimilating nitrogen via the photorespiratory pathway." Nature Plants 4, 46–54 (2018) [doi:10.1038/s414770170065x].
- Busch "Photorespiration in the context of Rubisco biochemistry, CO2 diffusion and metabolism." The Plant Journal 101, 919–939 (2020) [doi:10.1111/tpj.14674].
- von Caemmerer, S. "Biochemical Models of Leaf Photosynthesis" (CSIRO Publishing, 2000) [doi:10.1071/9780643103405].
- Lochocki & McGrath "Widely Used Variants of the Farquhar-von-Caemmerer-Berry Model Can Cause Errors in Parameter Estimates and Simulations." submitted.

Value

The return value depends on the value of `return_table`:

- If `return_table` is `TRUE`, the return value is an `exdf` object with the following columns, calculated as described above: `Tp_tl`, `Vcmax_tl`, `RL_tl`, `J_tl`, `Ac`, `Aj`, `Ap`, `An`, `Vc`, and others. The category for each of these new columns is `calculate_c3_assimilation` to indicate that they were created using this function.
- If `return_table` is `FALSE`, the return value is a list with the following named elements: `An`, `Ac`, `Aj`, `Ap`, and `J_tl`. Each element is a numeric vector.

If `data_table` is not an `exdf` object, then the return value will be a data frame, and units and categories will not be reported.

Examples

```
# Simulate a C3 A-Cc curve with specified leaf temperature and photosynthetic
# parameters and plot the net assimilation rate along with the different
# enzyme-limited rates
inputs <- exdf(data.frame(
  Cc = seq(1, 601, by = 6),
  Tleaf = 30,
  total_pressure = 1,
  Oxygen = 21
))

inputs <- document_variables(
  inputs,
  c('', 'Cc',             'micromol mol^(-1)'),
  c('', 'Tleaf',          'degrees C'),
  c('', 'total_pressure', 'bar'),
  c('', 'Oxygen',         'percent')
)

inputs <- calculate_temperature_response(inputs, c3_temperature_param_sharkey, 'Tleaf')

assim <- calculate_c3_assimilation(inputs, 0, 0, 0, 0, '', 150, '', '', 1, 12, 120)

lattice::xyplot(
  Ac + Aj + Ap + An ~ inputs[, 'Cc'],
  data = assim$main_data,
  type = 'l',
  grid = TRUE,
  auto = TRUE,
  xlab = paste0('Chloroplast CO2 concentration (', inputs$units$Cc, ')'),
  ylab = paste0('Assimilation rate (', assim$units$An, ')')
)
```

---

calculate_c3_limitations_grassi

Estimate the relative limiting factors to C3 photosynthesis

Description

Uses the method from Grassi & Magnani (2005) to estimate the relative limitations to C3 photosynthesis due to stomatal conductance, mesophyll conductance, and biochemistry. This function can accomodate alternative column names for the variables taken from the data file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c3_limitations_grassi(
  exdf_obj,
```

```
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  cc_column_name = 'Cc',
  gamma_star_column_name = 'Gamma_star_tl',
  gmc_column_name = 'gmc_tl',
  gsc_column_name = 'gsc',
  kc_column_name = 'Kc_tl',
  ko_column_name = 'Ko_tl',
  oxygen_column_name = 'Oxygen',
  total_pressure_column_name = 'total_pressure',
  vcmax_column_name = 'Vcmax_tl',
  j_column_name = NULL
)
```

Arguments

exdf_obj	An exdf object representing gas exchange data.
Wj_coef_C	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
Wj_coef_Gamma_star	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
cc_column_name	The name of the column in exdf_obj that contains the chloroplastic CO2 concentration in micromol mol^(-1). Typically these are values that are automatically calculated by fit_c3_aci.
gamma_star_column_name	The name of the column in exdf_obj that contains the Gamma_star values in micromol mol^(-1). Typically these are the leaf-temperature dependent values that are automatically calculated by fit_c3_aci.
gmc_column_name	The name of the column in exdf_obj that contains the mesophyll conductance to CO2 in mol m^(-2) s^(-1) bar^(-1). Typically these are the leaf-temperature adjusted values that are automatically calculated by fit_c3_aci.
gsc_column_name	The name of the column in exdf_obj that contains the stomatal conductance to CO2 in mol m^(-2) s^(-1). Typically this column is calculated using calculate_gas_properties.
kc_column_name	The name of the column in exdf_obj that contains the Michaelis-Menten constant for rubisco carboxylation in micromol mol^(-1). Typically these are the leaf-temperature dependent values that are automatically calculated by fit_c3_aci.
ko_column_name	The name of the column in exdf_obj that contains the Michaelis-Menten constant for rubisco oxygenation in mmol mol^(-1). Typically these are the leaf-temperature dependent values that are automatically calculated by fit_c3_aci.
oxygen_column_name	The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

total_pressure_column_name
: The name of the column in exdf_obj that contains the total pressure in bar. Typically this is calculated using calculate_total_pressure.

vcmax_column_name
: The name of the column in exdf_obj that contains values of the maximum Rubisco carboxylation rate (Vcmax) in micromol m^(-2) s^(-1). Typically these are the leaf-temperature adjusted values that are automatically calculated by fit_c3_aci.

j_column_name
: The name of the column in exdf_obj that contains values of the RuBP regeneration rate (J) in micromol m^(-2) s^(-1). Typically these are the leaf-temperature adjusted values that are automatically calculated by fit_c3_aci.

Details

When analyzing or interpreting C3 gas exchange data, it is often useful to estimate the relative limitations to assimilation that are due to stomatal conductance, mesophyll conductance, and biochemistry. This can be done using a framework first introduced by Grassi & Magnani (2005). In this framework, the relative limitation due to stomatal conductance (ls) is

ls = [(g_t / g_sc) * (dAdC)] / [g_t + dAdC],

the relative limitation due to mesophyll conductance (lm) is

lm = [(g_t / g_mc) * (dAdC)] / [g_t + dAdC],

and the relative limitation due to biochemistry (lb) is

ln = [g_t] / [g_t + dAdC],

where g_sc is the stomatal conductance to CO2, g_mc is the mesophyll conductance to CO2, gt = 1 / (1 / g_mc + 1 / g_sc) is the total conductance to CO2, and dAdC is the partial derivative of the net CO2 assimilation rate (An) with respect to the chloroplast CO2 concentration (Cc). These can be found in Equation 7 from Grassi & Magnani (2005).

These equations were derived by assuming that CO2 assimilation is limited by Rubisco activity; in other words, that the net CO2 assimilation rate is given by

Ac = Vcmax * (Cc - Gamma_star) / (Cc + Km) - RL,

where Vcmax is the maximum Rubisco carboxylation rate, Gamma_star is the CO2 compensation point in the absence of day respiration, RL is the day respiration rate, Km is the effective Michaelis-Menten constant for Rubisco carboxylation. In turn, Km is given by Km = Kc * (1 + O / Ko), where Kc is the Michaelis-Menten constant for carboxylation, Ko is the Michaelis-Menten constant for oxygenation, and O is the oxygen concentration in the chloroplast.

Under this assumption, it is possible to analytically determine the partial derivative dAdC:

dAdC_rubisco = Vcmax * (Gamma_star + Km) / (Cc + Km)^2

In this case, the limitation due to "biochemistry" actually refers to limitation due to the value of Vcmax. Note that sometimes this derivative is estimated from the initial slope of a measured A-Ci curve rather than calculated analytically. (See, for example, Pathare et al. (2023).) However, we do not take that approach here. Also note that the value of Vcmax can be estimated using different approaches. For example, Xiong (2023) uses single-point gas exchange measurements. When possible, it would be better to use an estimate from fitting an entire A-Ci curve, as shown in the example below.

To understand the meaning of these limiting factors, note that simultaneously making small fractional increases to g_sc, g_mc, and Vcmax will generally cause an associated small fractional increase in An. The limiting factors describe the fraction of the increase in An that can be attributed to each of g_sc, g_mc, and Vcmax. For example, ls = 0.2, lm = 0.3, lb = 0.5 would mean that 20 percent of the increase in An would be due to an increase in stomatal conductance, 30 percent due to an increase in mesophyll conductance, and 50 percent due to an increase in Vcmax. Note that ls, lm, and lb always add up to 1.

Thus, when one of the factors is large, changes in the related parameter produce relatively larger changes in the assimilation rate. In that case, it can be said that that parameter is setting a large limit on the assimilation rate. On the other hand, if a factor is small, small changes in the related parameter produce relatively small changes in An, and therefore that parameter is not setting a large limit on the assimilation rate.

It is also possible to calculate dAdC when assimilation is limited by RuBP regeneration. In this case, we have

Aj = J * (Cc - Gamma_star) / (4 * Cc + 8 * Gamma_star) - RL,

where J is the RuBP regeneration rate, and the limitation due to "biochemistry" actually refers to limitation due to the value of J (rather than Vcmax. The same equations as before can be used to calculate the limiting factors (ls, lm, lb), but the partial derivative is now given by

dAdC_j = J * Gamma_star * 12 / (4 * Cc + 8 * Gamma_star)^2.

Most users will want the limitations assuming Rubisco-limited assimilation. However, if j_column_name is not NULL, values of J will be used to calculate the limiting factors assuming RuBP-regeneration-limited assimilation. For an example of how these additional factors can be used, see Sakoda et al. (2021).

References:

Grassi, G. & Magnani, F. "Stomatal, mesophyll conductance and biochemical limitations to photosynthesis as affected by drought and leaf ontogeny in ash and oak trees." Plant, Cell & Environment 28, 834–849 (2005) [doi:10.1111/j.13653040.2005.01333.x].

Pathare, V. S. et al. "Altered cell wall hydroxycinnamate composition impacts leaf- and canopy-level CO2 uptake and water use in rice." Plant Physiology kiad428 (2023) [doi:10.1093/plphys/kiad428].

Xiong, D. "Leaf anatomy does not explain the large variability of mesophyll conductance across C3 crop species." The Plant Journal 113, 1035–1048 (2023) [doi:10.1111/tpj.16098].

Sakoda, K., Yamori, W., Groszmann, M. & Evans, J. R. "Stomatal, mesophyll conductance, and biochemical limitations to photosynthesis during induction." Plant Physiology 185, 146–160 (2021) [doi:10.1093/plphys/kiaa011].

Value

This function returns an exdf object based on exdf_obj but with several new columns representing the partial derivatives and limiting factors discussed above: dAdC_rubisco, ls_rubisco_grassi, lm_rubisco_grassi, and lb_rubisco_grassi. If j_column_name is not NULL, the output will also include dAdC_j, ls_j_grassi, lm_j_grassi, and lb_j_grassi.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate additional gas properties
licor_file <- calculate_gas_properties(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Fit all curves in the data set. Here we use a faster optimizer than the
# default one to ensure the example runs quickly.
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer_nmkb(1e-7),
  fit_options = list(gmc_at_25 = 0.5)
))

# Get a subset of fitting results corresponding to the first measured point
# in each curve (where CO2_r_sp = 400 ppm)
aci_fit_subset <- aci_results$fits[aci_results$fits[, 'CO2_r_sp'] == 400, , TRUE]

# Calculate limiting factors
aci_fit_subset <- calculate_c3_limitations_grassi(aci_fit_subset)

# View the limiting factors for each species / plot
col_to_keep <- c(
  'species', 'plot',                                          # identifiers
  'ls_rubisco_grassi', 'lm_rubisco_grassi', 'lb_rubisco_grassi' # limitation info
)

aci_fit_subset[ , col_to_keep, TRUE]
```

```
# One of these fits has NA for all the limiting factors, which causes problems
# when making bar charts with some versions of the `lattice` package, so we
# exclude that curve for plotting
data_for_barchart <-
  aci_fit_subset$main_data[aci_fit_subset$main_data$species_plot != 'tobacco - 2', ]

# Display as a bar chart
lattice::barchart(
  ls_rubisco_grassi + lm_rubisco_grassi + lb_rubisco_grassi ~ species_plot,
  data = data_for_barchart,
  stack = TRUE,
  auto = TRUE,
  ylab = 'Factors limiting assimilation'
)
```

`calculate_c3_limitations_warren` *Estimate the relative limiting factors to C3 photosynthesis*

Description

Uses the method from Warren et al. (2003) to estimate the relative limitations to C3 photosynthesis due to stomatal conductance and mesophyll conductance. This function can accomodate alternative column names for the variables taken from the data file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c3_limitations_warren(
  exdf_obj,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  ca_column_name = 'Ca',
  cc_column_name = 'Cc',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  hard_constraints = 0,
  ...
)
```

Arguments

exdf_obj	An exdf object representing gas exchange data. Typically this should be an exdf object returned from fit_c3_aci; it will be expected to have columns for alpha_g, Gamma_star, J_at_25, RL_at_25, Tp, and Vcmax_at_25.
Wj_coef_C	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
Wj_coef_Gamma_star	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
ca_column_name	The name of the column in exdf_obj that contains the ambient CO2 concentration in micromol mol^(-1).
cc_column_name	The name of the column in exdf_obj that contains the chloroplastic CO2 concentration in micromol mol^(-1). Typically these are values that are automatically calculated by fit_c3_aci.
ci_column_name	The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_norm_column_name	The name of the column in exdf_obj that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C). Typically these are the leaf-temperature dependent values calculated using calculate_temperature_response.
j_norm_column_name	The name of the column in exdf_obj that contains the normalized J values (with units of normalized to J at 25 degrees C). Typically these are the leaf-temperature dependent values calculated using calculate_temperature_response.
kc_norm_column_name	The name of the column in exdf_obj that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C). Typically these are the leaf-temperature dependent values calculated using calculate_temperature_response.
ko_norm_column_name	The name of the column in exdf_obj that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C). Typically these are the leaf-temperature dependent values calculated using calculate_temperature_response.
oxygen_column_name	The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.
rl_norm_column_name	The name of the column in exdf_obj that contains the normalized RL values (with units of normalized to RL at 25 degrees C).
total_pressure_column_name	The name of the column in exdf_obj that contains the total pressure in bar. Typically this is calculated using calculate_total_pressure.

tp_norm_column_name
: The name of the column in exdf_obj that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in exdf_obj that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

hard_constraints
: To be passed to calculate_c3_assimilation; see that function for more details.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

When analyzing or interpreting C3 gas exchange data, it is often useful to estimate the relative limitations to assimilation that are due to stomatal conductance or mesophyll conductance. This can be done using a framework first introduced by Warren et al. (2003). In this framework, the relative limitation due to stomatal conductance (ls) is

ls = (An_inf_gsc - A_modeled) / An_inf_gsc

and the relative limitation due to mesophyll conductance (lm) is

lm = (An_inf_gmc - A_modeled) / An_inf_gmc. These are equations 10 and 11 in Warren et al. (2003).

In these equations A_modeled is the net assimilation rate calculated using the Farquhar-von-Caemmerer-Berry (FvCB) model at the measured value of the chloroplast CO2 concentration (Cc). The other two assimilation rates (An_inf_gsc and An_inf_gmc) are also calculated using the FvCB model, but under different assumptions: An_inf_gsc assumes that stomatal conductance is infinite while mesophyll conductance is as measured, while An_inf_gmc assumes that mesophyll conductance is infinite while stomatal conductance is as measured.

In other words, ls expresses the observed assimilation rate as a fractional decrease relative to a hypothetical plant with infinite stomatal conductance, while lm expresses the observed assimilation rate as a fractional decrease relative to a hypothetical plant with infinite mesophyll conductance.

For example, if lm = 0.4, this means that the observed assimilation rate is 40 percet lower than a hypothetical plant with infinite mesophyll conductance. If mesophyll conductance were to increase (all else remaining the same), then lm would decrease. This is not the case with other estimations of limiting factors, such as the one used in calculate_c3_limitations_grassi. (See Leverett & Kromdijk for more details.)

To actually calculate An_inf_gsc and An_inf_gmc, it is first necessary to estimate the corresponding values of Cc that would occur with infinite stomatal or mesophyll conductance. This can be done with a 1D diffusion equation expressed using drawdown values:

Cc = Ca - drawdown_cs - drawdown_cm,

where drawdown_cs = Ca - Ci is the drawdown of CO2 across the stomata (assuming infinite boundary layer conductance) and drawdown_cm = Ci - Cc is the drawdown of CO2 across the mesophyll. If one conductance is infinite, the corresponding drawdown becomes zero. Thus, we have:

Cc_inf_gsc = Ca - 0 - (Ci - Cc) = Ca - Ci + Cc

and

Cc_inf_gmc = Ca - (Ca - Ci) - 0 = Ci,

where Cc_inf_gsc is the value of Cc that would occur with infinite stomatal conductance and the measured mesophyll conductance, and Cc_inf_gmc is the value of Cc that would occur with infinite mesophyll conductance and the measured stomatal conductance.

Once values of Cc, Cc_inf_gsc, and Cc_inf_gmc, the corresponding assimilation rates are calculated using calculate_c3_assimilation, and then the limitation factors are calculated as described above.

References:

Warren, C. R. et al. "Transfer conductance in second growth Douglas-fir (Pseudotsuga menziesii (Mirb.)Franco) canopies." Plant, Cell & Environment 26, 1215–1227 (2003) [doi:10.1046/j.1365-3040.2003.01044.x].

Leverett, A. & Kromdijk, J. "The long and tortuous path towards improving photosynthesis by engineering elevated mesophyll conductance." [doi:10.22541/au.170016201.13513761/v1].

Value

This function returns an exdf object based on exdf_obj but with several new columns representing the quantities discussed above: Cc_inf_gsc, Cc_inf_gmc, An_inf_gsc, An_inf_gmc, ls_warren, and lm_warren.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate additional gas properties
licor_file <- calculate_gas_properties(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Fit all curves in the data set. Here we use a faster optimizer than the
# default one to ensure the example runs quickly.
aci_results <- consolidate(by(
```

```
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer_nmkb(1e-7)
))

# Get a subset of fitting results corresponding to the first measured point
# in each curve (where CO2_r_sp = 400 ppm)
aci_fit_subset <- aci_results$fits[aci_results$fits[, 'CO2_r_sp'] == 400, , TRUE]

# Calculate limiting factors
aci_fit_subset <- calculate_c3_limitations_warren(aci_fit_subset)

# View the limiting factors for each species / plot
col_to_keep <- c(
  'species', 'plot',       # identifiers
  'ls_warren', 'lm_warren' # limitation info
)

aci_fit_subset[ , col_to_keep, TRUE]
```

`calculate_c3_variable_j` *Calculate C3 variable J*

Description

Calculates values of mesophyll conductance and chloroplast CO2 concentration using the "variable J" equation, as originally described in Harley et al. (1992) and modified in Moualeu-Ngangue, Chen, & Stutzel (2016). This function can accomodate alternative colum names for the variables taken from Licor files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c3_variable_j(
  exdf_obj,
  alpha_g,
  alpha_s,
  alpha_t,
  Gamma_star_at_25,
  RL_at_25,
  tau,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
```

```
    phips2_column_name = 'PhiPS2',
    qin_column_name = 'Qin',
    rl_norm_column_name = 'RL_norm',
    total_pressure_column_name = 'total_pressure',
    hard_constraints = 0,
    perform_checks = TRUE,
    return_exdf = TRUE
  )
```

Arguments

exdf_obj	An exdf object.
alpha_g	A dimensionless parameter where 0 <= alpha_g <= 1, representing the proportion of glycolate carbon taken out of the photorespiratory pathway as glycine. alpha_g is often assumed to be 0. If alpha_g is not a number, then there must be a column in exdf_obj called alpha_g with appropriate units. A numeric value supplied here will overwrite the values in the alpha_g column of exdf_obj if it exists.
alpha_s	A dimensionless parameter where 0 <= alpha_s <= 0.75 * (1 - alpha_g) representing the proportion of glycolate carbon taken out of the photorespiratory pathway as serine. alpha_s is often assumed to be 0. If alpha_s is not a number, then there must be a column in exdf_obj called alpha_s with appropriate units. A numeric value supplied here will overwrite the values in the alpha_s column of exdf_obj if it exists.
alpha_t	A dimensionless parameter where 0 <= alpha_t <= 1 representing the proportion of glycolate carbon taken out of the photorespiratory pathway as CH2-THF. alpha_t is often assumed to be 0. If alpha_t is not a number, then there must be a column in exdf_obj called alpha_t with appropriate units. A numeric value supplied here will overwrite the values in the alpha_t column of exdf_obj if it exists.
Gamma_star_at_25	The chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at 25 degrees C, expressed in micromol mol^(-1). If Gamma_star_at_25 is not a number, then there must be a column in exdf_obj called Gamma_star_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Gamma_star_at_25 column of exdf_obj if it exists.
RL_at_25	The respiration rate at 25 degrees C, expressed in micromol m^(-2) s^(-1). If RL_at_25 is not a number, then there must be a column in exdf_obj called RL_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the RL_at_25 column of exdf_obj if it exists.
tau	The proportionality factor used to calculate the RuBP regeneration rate from chlorophyll fluorescence measurements (dimensionless). If tau is not a number, then there must be a column in exdf_obj called tau with appropriate units. A numeric value supplied here will overwrite the values in the tau column of exdf_obj if it exists.

Wj_coef_C A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

Wj_coef_Gamma_star
A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

a_column_name The name of the column in exdf_obj that contains the net assimilation in micromol m^(-2) s^(-1).

ci_column_name The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).

gamma_star_norm_column_name
The name of the column in exdf_obj that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).

phips2_column_name
The name of the column in exdf_obj that contains values of the operating efficiency of photosystem II (dimensionless).

qin_column_name
The name of the column in exdf_obj that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).

rl_norm_column_name
The name of the column in exdf_obj that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
The name of the column in exdf_obj that contains the total pressure in bar.

hard_constraints
An integer numerical value indicating which types of hard constraints to place on the values of input parameters; see below for more details.

perform_checks A logical value indicating whether to check units for the required columns. This should almost always be TRUE. The option to disable these checks is only intended to be used when fit_c3_variable_j calls this function, since performing these checks many times repeatedly slows down the fitting procedure.

return_exdf A logical value indicating whether to return an exdf object. This should almost always be TRUE. The option to return a vector is mainly intended to be used when fit_c3_variable_j calls this function, since creating an exdf object to return will slow down the fitting procedure.

Details

The "Variable J" method is a way to estimate the chloroplast CO2 concentration Cc and the mesophyll conductance to CO2 gmc from combined gas exchange and chlorophyll fluorescence measurements, and was originally described in Harley et al. (1992). The main idea is that along with Cc, the net CO2 assimilation rate (An), day respiration rate (RL), and CO2 compensation point in the absence of day respiration (Gamma_star) determine the actual RuBP regeneration rate (J_actual) required to support the Calvin-Benson cycle:

J_actual = (A + RL) * (4 * Cc + 8 * Gamma_star) / (Cc - Gamma_star)

This is Equation 6 in Harley et al. (1992). (Note: this equation can be derived by solving the equation for Aj from the FvCB model for J. However, this relationship holds true even when CO2 assimilation is not limited by RuBP regeneration. Hence, we distinguish between the actual regeneration rate J_actual and the maximum regeneration rate for a given incident light level J.)

This equation can be rewritten by using a 1D diffusion equation to replace Cc with Cc = Ci - An / gmc and then solving for the mesophyll conductance. The result is Equation 7 in Harley et al. (1992), which we do not reproduce here. The importance of Equation 7 is that it calculates gmc from several quantities that can be measured using gas exchange (Ci, An, and RL), a quantity whose values can be known beforehand (Gamma_star), and J_actual (which can be estimated from chlorophyll fluorescence measurements). Here we update Equation 7 to include alpha_g and alpha_s following Busch et al. (2018) (also see calculate_c3_assimilation.)

The actual RuBP regeneration rate is related to the incident photosynthetically active flux density Qin and the operating efficiency of photosystem II PhiPSII according to:

J_actual = alpha_g * beta * Qin * PhiPSII,

where alpha_g is the leaf absorptance and beta is the fraction of absorbed light energy directed to photosystem II. Qin is set by the measurement conditions, while PhiPSII can be estimated from chlorophyll fluorescence. However, the values of alpha_g and beta are generally unknown; beta in particular is difficult or impossible to measure and is often assumed to be 0.5. Thus, while Equation 7 from Harley et al. (1992) can be used to estimate gmc, there is a practical uncertainty associated with determining a value of J_actual to be used in Equation 7.

Moualeu-Ngangue, Chen, & Stutzel (2016) developed a way to address this issue. The method from that paper replaces the product of alpha_g and beta by a single new parameter tau, and uses it to estimate the actual RuBP regeneration from fluoresence (J_F):

J_F = tau * Qin * PhiPSII.

This new parameter tau is assumed to be constant across an A-Ci curve, and is treated as an unknown whose value will be determined during a fitting procedure.

In this function, the supplied values of Qin, PhiPSII, and tau are used to calculate values of J_F. Then, the values of J_F are used along with Equation 7 from Harley et al. (1992) to calculate gmc. Finally, a 1D diffusion equation is used to calculate Cc.

Hard constraints:

Most input parameters to the Variable J equations have hard constraints on their values which are set by their biochemical or physical interpretation; for example, RL cannot be negative and tau must lie between 0 and 1. Yet, because of measurement noise, sometimes it is necessary to use values outside these ranges when fitting an A-Ci curve with fit_c3_variable_j. To accomodate different potential use cases, it is possible to selectively apply these hard constraints by specifying different values of the hard_constraints input argument:

- hard_constraints = 0: Constraints are only placed on inputs that are user-supplied and cannot be fit, such as Qin.
- hard_constraints = 1: Includes the same constraints as when hard_constraints is 0, with the additional constraint that all Ci values must be non-negative.
- hard_constraints = 2: Includes the same constraints as when hard_constraints is 1, which additional constraints on the parameters that can be fitted. For example, RL_at_25 must be non-negative and tau must lie between 0 and 1.

If any input values violate any of the specified constraints, an error message will be thrown.

References:

- Harley, P. C., Loreto, F., Di Marco, G. & Sharkey, T. D. "Theoretical Considerations when Estimating the Mesophyll Conductance to CO2 Flux by Analysis of the Response of Photosynthesis to CO2" Plant Physiology 98, 1429–1436 (1992) [doi:10.1104/pp.98.4.1429].
- Moualeu-Ngangue, D. P., Chen, T.-W. & Stutzel, H. "A new method to estimate photosynthetic parameters through net assimilation rate-intercellular space CO2 concentration (A-Ci) curve and chlorophyll fluorescence measurements" New Phytologist 213, 1543–1554 (2017) [doi:10.1111/nph.14260].
- Busch, Sage, & Farquhar, G. D. "Plants increase CO2 uptake by assimilating nitrogen via the photorespiratory pathway." Nature Plants 4, 46–54 (2018) [doi:10.1038/s414770170065x].

Value

The return value depends on the value of `return_exdf`:

- If `return_exdf` is `TRUE`, the return value is an `exdf` object with the following columns, calculated as described above: `J_F`, `gmc`, `Cc`, `tau`, and `RL_tl`. The category for each of these new columns is `calculate_c3_variable_j` to indicate that they were created using this function.
- If `return_exdf` is `FALSE`, the return value is a list with the following named elements: `gmc`, `Cc`, and `J_F`. Each element is a numeric vector.

Examples

```
# Read an example Licor file included in the PhotoGEA package. This file
# includes gas exchange and chlorophyll fluorescence data.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'])

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Calculate values of J_F, gmc, and Cc assuming alpha_g = alpha_s = alpha_t = 0,
# RL_at_25 = 1.5, and tau = 0.55.
vj_res <- calculate_c3_variable_j(licor_file, 0, 0, 0, '', 1.5, 0.55)

# Plot mesophyll conductance against Cc. Note: this information is not very
# meaningful since the values of Gamma_star, tau and RL used above are
# arbitrary.
lattice::xyplot(
  gmc ~ Cc | licor_file[, 'species_plot'],
  data = vj_res$main_data,
```

```
  type = 'b',
  pch = 16,
  auto = TRUE,
  xlab = paste0('Chloroplast CO2 concentration (', vj_res$units$Cc, ')'),
  ylab = paste0('Mesophyll conductance to CO2 (', vj_res$units$gmc, ')')
)
```

`calculate_c4_assimilation` *Calculate C4 assimilation rates*

Description

Calculates C4 assimilation rates based on the von Caemmerer (2000) model. This function can accomodate alternative colum names for the variables taken from Licor files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c4_assimilation(
  exdf_obj,
  alpha_psii,
  gbs,
  J_at_25,
  RL_at_25,
  Rm_frac,
  Vcmax_at_25,
  Vpmax_at_25,
  Vpr,
  x_etr = 0.4,
  ao_column_name = 'ao',
  gamma_star_column_name = 'gamma_star',
  j_norm_column_name = 'J_norm',
  kc_column_name = 'Kc',
  ko_column_name = 'Ko',
  kp_column_name = 'Kp',
  oxygen_column_name = 'Oxygen',
  pcm_column_name = 'PCm',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  vcmax_norm_column_name = 'Vcmax_norm',
  vpmax_norm_column_name = 'Vpmax_norm',
  hard_constraints = 0,
  perform_checks = TRUE,
  return_exdf = TRUE
)
```

Arguments

exdf_obj	An exdf object.
alpha_psii	The fraction of photosystem II activity in the bundle sheath (dimensionless). If alpha_psii is not a number, then there must be a column in exdf_obj called alpha_psii with appropriate units. A numeric value supplied here will overwrite the values in the alpha_psii column of exdf_obj if it exists.
gbs	The bundle sheath conductance to CO2 in mol m^(-2) s^(-1) bar^(-1). If gbs is not a number, then there must be a column in exdf_obj called gbs with appropriate units. A numeric value supplied here will overwrite the values in the gbs column of exdf_obj if it exists.
J_at_25	The electron transport rate at 25 degrees C, expressed in micromol m^(-2) s^(-1). Note that this is _not_ Jmax, and in general will depend on the incident photosynthetically active flux density. If J_at_25 is not a number, then there must be a column in exdf_obj called J_at_25 with appropriate units. A numeric value supplied here will override the values in the J_at_25 column of exdf_obj if it exists.
RL_at_25	The total rate of mitochondrial respiration across the mesophyll and bundle sheath at 25 degrees C, expressed in micromol m^(-2) s^(-1). If RL_at_25 is not a number, then there must be a column in exdf_obj called RL_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the RL_at_25 column of exdf_obj if it exists.
Rm_frac	The fraction of the total mitochondrial respiration that occurs in the mesophyll. If Rm_frac is not a number, then there must be a column in exdf_obj called Rm_frac with appropriate units. A numeric value supplied here will overwrite the values in the Rm_frac column of exdf_obj if it exists.
Vcmax_at_25	The maximum rate of rubisco carboxylation at 25 degrees C, expressed in micromol m^(-2) s^(-1). If Vcmax_at_25 is not a number, then there must be a column in exdf_obj called Vcmax_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Vcmax_at_25 column of exdf_obj if it exists.
Vpmax_at_25	The maximum rate of PEP carboxylase activity at 25 degrees C, expressed in micromol m^(-2) s^(-1). If Vpmax_at_25 is not a number, then there must be a column in exdf_obj called Vpmax_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Vpmax_at_25 column of exdf_obj if it exists.
Vpr	The rate of PEP carboxylase regeneration, expressed in micromol m^(-2) s^(-1). If Vpr is not a number, then there must be a column in exdf_obj called Vpr with appropriate units. A numeric value supplied here will overwrite the values in the Vpr column of exdf_obj if it exists.
x_etr	The fraction of whole-chain electron transport occurring in the mesophyll (dimensionless). See Equation 29 from S. von Caemmerer (2021).
ao_column_name	The name of the column in exdf_obj that contains the dimensionless ratio of solubility and diffusivity of O2 to CO2.
gamma_star_column_name	The name of the column in exdf_obj that contains the dimensionless gamma_star values.

`j_norm_column_name`
: The name of the column in `exdf_obj` that contains the normalized `J` values (with units of `normalized to J at 25 degrees C`).

`kc_column_name`
: The name of the column in `exdf_obj` that contains the Michaelis-Menten constant for rubisco carboxylation in `microbar`.

`ko_column_name`
: The name of the column in `exdf_obj` that contains the Michaelis-Menten constant for rubisco oxygenation in `mbar`.

`kp_column_name`
: The name of the column in `exdf_obj` that contains the Michaelis-Menten constant for PEP carboxylase carboxylation in `microbar`.

`oxygen_column_name`
: The name of the column in `exdf_obj` that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be `percent`.

`pcm_column_name`
: The name of the column in `exdf_obj` that contains the partial pressure of CO2 in the mesophyll, expressed in `microbar`.

`rl_norm_column_name`
: The name of the column in `exdf_obj` that contains the normalized `RL` values (with units of `normalized to RL at 25 degrees C`).

`total_pressure_column_name`
: The name of the column in `exdf_obj` that contains the total pressure in `bar`.

`vcmax_norm_column_name`
: The name of the column in `exdf_obj` that contains the normalized `Vcmax` values (with units of `normalized to Vcmax at 25 degrees C`).

`vpmax_norm_column_name`
: The name of the column in `exdf_obj` that contains the normalized `Vpmax` values (with units of `normalized to Vpmax at 25 degrees C`).

`hard_constraints`
: An integer numerical value indicating which types of hard constraints to place on the values of input parameters; see below for more details.

`perform_checks`
: A logical value indicating whether to check units for the required columns. This should almost always be `TRUE`. The option to disable these checks is only intended to be used when `fit_c4_aci` calls this function, since performing these checks many times repeatedly slows down the fitting procedure.

`return_exdf`
: A logical value indicating whether to return an `exdf` object. This should almost always be `TRUE`. The option to return a vector is mainly intended to be used when `fit_c4_aci` calls this function, since creating an `exdf` object to return will slow down the fitting procedure.

Details

General Description of the Model

This function generally follows Sections 4.2.1 and 4.2.2 from S. von Caemmerer (2000), which provides equations for calculating the enzyme-limited net assimilation rate `Ac`, the light- and electron-transport limited rate `Aj`, and the overall net assimilation rate `An` in a C4 leaf. (These equations are also reproduced in S. von Caemmerer (2021), although we use the equation numbers from the 2000

textbook here. Also note there is a typo in Equation 22 from the 2021 paper.) The enzyme-limited assimilation rate in this model is calculated according to Equation 4.21:

```
Ac = (-b - sqrt(b^2 - 4 * a * c)) / (2 * a)
```

where the parameters a, b, and c are determined by Equations 4.22, 4.23, and 4.24, respectively. These equations are fairly long, so we do not reproduce them here. Similarly, the light-limited rate Aj is also calculated according to a quadratic equation. Finally, the overall rate is calculated as the smaller of Ac and Aj:

```
An = min(Ac, Aj)
```

An Approximation to the Full Equations

The complicated equations above can be approximiated by simpler ones. For Ac, we can use Equation 4.25:

```
Ac = min(Vp + gbs * PCm - RLm, Vcmax - RL)
```

where Vp is the rate of PEP carboxylation, gbs is the bundle sheath conductance to CO2, PCm is the partial pressure of CO2 in the mesophyll, RLm is the rate of mitochondrial respiration occuring in the mesophyll, Vcmax is the maximum rate of Rubisco carboxylation, and RL is the rate of mitochondrial respiration occurring in the bundle sheath and mesophyll. Essentially, the first term in the equation above (Vp + gbs * PCm - RLm) can be thought of as a PEP-carboxylase-limited assimilation rate Ap, while the second term (Vcmax - RL) is a Rubisco-limited rate Ar.

The PEP carboxylation rate Vp is calculated according to Equation 4.19:

```
Vp = min(Pcm * Vpmax / (PCm + Kp), Vpr)
```

where Vpmax is the maximum rate of PEP carboxylation, Kp is a Michaelis-Menten constant for PEP carboxylation, and Vpr is the carboxylation rate when PEP carboxylase activity is limited by regeneration rather than carbon availability. Thus, we can see that the approximation above actually calculates the enzyme-limited rate as the smaller of three separate assimilation rates:

```
Ac = min(Apc, Apr, Ar)
```

where Apc = Pcm * Vpmax / (PCm + Kp) + gbs * PCm - RLm is the rate due to carbon-limited PEP carboxylation, Apr = Vpr + gbs * PCm - RLm is the rate due to regeneration-limited PEP carboxylation, and Ar = Vcmax - RL is the rate due to Rubisco-limited assimilation.

In the example at the end of this documentation page, we compare Apc, Apr, and Ar to Ac as calculated by Equation 4.21. From this example, it is clear that the approximation Ac = min(Apc, Apr, Ar) is quite accurate for low values of PCm, but introduces significant errors as PCm increases. Thus, while the approximation can be helpful for gaining an intuitive understanding of C4 photosynthesis, it should not be used for realistic calculations.

To be more precise, the approximation is only reliable when Vcmax is much larger than gbs * Kc * (1 + POm / Ko), which is rarely the case; otherwise, the limiting value of An at high PCm will be smaller than Ar = Vcmax - RL. Conversely, if gbs and alpha_psii are both set to zero, then the approximation is exact.

For Aj, the simplified version is Equation 4.45:

```
Aj = min(x_etr * J / 2 - RLm + gbs * PCm, (1 - x_etr) * J / 3 - RL)
```

where x_etr is the fraction of whole-chain electron transport occurring in the mesophyll and J is the electron transport rate. We can therefore think of this equation as

```
Aj = min(Ajm, Ajbs)
```

where Ajm is the mesophyll light-limited rate and Ajbs is the bundle sheath light-limited rate. These are given by Ajm = x_etr * J / 2 - RLm + gbs * PCm and (1 - x_etr) * J / 3 - RL As in the case with Ac, this approximation is not exact.

Combining these two simplifications, we can see that the overall net assimilation rate can be approximated as the smallest of five potential rates:

An = min(Apc, Apr, Ar, Ajm, Ajbs).

Here it is very important to note that some of these potential rates have identical or similar dependence on PCm. More specifically, Apr and Ajm have identical dependence, as do Ar and Ajbs. If gbs is zero, all four of these rates have no dependence on PCm. Thus, from a fitting point of view, it is not usually possible to distinguish between these potential limiting states. For this reason, it is not advisable to fit more than one of Vcmax, Vpr, and J when estimating parameters from an experimentally measured curve.

Limiting Cases of the Approximate Equation

The bundle sheath conductance gbs is generally very small and can be ignored in a simple analysis of the above equations. In that case, when Pcm is very high, the approximate equation for Ac simplifies further to:

Ac = min(Vpmax - RLm, Vpr - RLm, Vcmax - RL)

Since respiration costs are also generally much smaller than the maximum enzyme activity and regeneration rates, the enzyme-limited assimilation rate at high levels of CO2 is therefore determined by the smaller of Vpmax, Vpr, and Vcmax. As shown in Table 4.1 of the textbook, Vpmax is typically much larger than the other two rates, so light- and CO2-saturated assimilation in C4 leaves is usually limited by either Vpr or Vcmax. The exact limiting factor can depend on many possible variables, such as the temperature. For example, see Wang (2008).

At lower values of PCm, enzyme-limited net assimilation is determined by CO2-limited PEP carboxylation according to:

An = PCm * Vpmax / Kp - RLm

where we have approximated gbs * PCm = 0 and PCm + Kp = Kp, as appropriate for small values of Pcm. Thus, we can see that for low CO2 levels, assimilation is linearly related to PCm with a slope of Vpmax / Kp and intercept of -RLm.

Respiration

Table 4.1 from von Caemmerer (2000) suggests that RL = 0.01 * Vcmax and RLm = 0.5 * RL. To allow more flexibility, we allow RL to be specified independently of Vcmax, and we also consider the ratio of RLm / RL = Rm_frac to be a variable (so that RLm is calculated from RL according to RLm = Rm_frac * RL). If Rm_frac is set to 1, then there is no distinction between RL and RLm.

Hard constraints:

Most input parameters to the C4 assimilation model have hard constraints on their values which are set by their biochemical or physical interpretation; for example, Vcmax cannot be negative and alpha_psii must lie between 0 and 1. Yet, because of measurement noise, sometimes it is necessary to use values outside these ranges when fitting an A-Ci curve with fit_c4_aci. To accomodate different potential use cases, it is possible to selectively apply these hard constraints by specifying different values of the hard_constraints input argument:

- hard_constraints = 0: Constraints are only placed on inputs that are user-supplied and cannot be fit, such as Kc.

- `hard_constraints = 1`: Includes the same constraints as when `hard_constraints` is 0, with the additional constraint that all `PCm` values must be non-negative.
- `hard_constraints = 2`: Includes the same constraints as when `hard_constraints` is 1, which additional constraints on the parameters that can be fitted. For example, `Vcmax_at_25` must be non-negative and `alpha_psii` must lie between 0 and 1.

If any input values violate any of the specified constraints, an error message will be thrown.

References

- von Caemmerer, S. "Biochemical Models of Leaf Photosynthesis" (CSIRO Publishing, 2000) [doi:10.1071/9780643103405].
- von Caemmerer, S. "Updating the steady-state model of C4 photosynthesis." Journal of Experimental Botany 72, 6003–6017 (2021) [doi:10.1093/jxb/erab266].
- Wang, D., Portis, A. R., Jr., Moose, S. P. & Long, S. P. "Cool C4 Photosynthesis: Pyruvate Pi Dikinase Expression and Activity Corresponds to the Exceptional Cold Tolerance of Carbon Assimilation in Miscanthus × giganteus." Plant Physiology 148, 557–567 (2008) [doi:10.1104/pp.108.120709].

Value

The return value depends on the value of `return_exdf`:

- If `return_exdf` is `TRUE`, the return value is an `exdf` object with the following columns: `alpha_psii`, `gbs`, `J_at_25`, `J_tl`, `Rm_frac`, `Vcmax_tl`, `Vpmax_tl`, `RL_tl`, `RLm_tl`, `Vpc`, `Vpr`, `Vp`, `Apc`, `Apr`, `Ap`, `Ar`, `Ajm`, `Ajbs`, `Ac`, `Aj`, `An`, and `c4_assimilation_msg`. Most of these are calculated as described above, while several are copies of the input arguments with the same name. The `c4_assimilation_msg` is usually blank but may contain information about any issues with the inputs. The category for each of these new columns is `calculate_c4_assimilation` to indicate that they were created using this function.
- If `return_exdf` is `FALSE`, the return value is a numeric vector containing the calculated values of `An`.

Examples

```
# Simulate a C4 A-Cm curve with specified leaf temperature and photosynthetic
# parameters and plot the net assimilation rate.
npts <- 101

inputs <- exdf(data.frame(
  PCm = seq(0, 500, length.out = npts),
  Tleaf = 25,
  Qin = 1800,
  total_pressure = 1,
  Oxygen = 21
))

inputs <- document_variables(
  inputs,
  c('', 'PCm',            'microbar'),
```

```
  c('', 'Tleaf',          'degrees C'),
  c('', 'Qin',            'micromol m^(-2) s^(-1)'),
  c('', 'total_pressure', 'bar'),
  c('', 'Oxygen',         'percent')
)

inputs <- calculate_temperature_response(inputs, c4_temperature_param_vc, 'Tleaf')

assim <- calculate_c4_assimilation(inputs, 0, 0.003, 250, 1, 0.5, 40, 200, 80)

# Now we can plot Ac, Apr, Apc, and Ar. From this plot, we can see that
# replacing the complicated quadratic equation with a simple minimum yields
# very different results. Although this approximation is helpful for
# understanding C4 photosythesis, it should not be used for calculations.
lattice::xyplot(
  Apr + Apc + Ar + Ac ~ inputs[, 'PCm'],
  data = assim$main_data,
  type = 'l',
  grid = TRUE,
  auto = TRUE,
  ylim = c(-5, 100),
  xlab = paste0('Partial pressure of CO2 in the mesophyll (', inputs$units$PCm, ')'),
  ylab = paste0('Net CO2 assimilation rate (', assim$units$An, ')')
)

# Likewise, we can look at Ajm, Ajbs, and Aj
lattice::xyplot(
  Ajm + Ajbs + Aj ~ inputs[, 'PCm'],
  data = assim$main_data,
  type = 'l',
  grid = TRUE,
  auto = TRUE,
  ylim = c(-5, 45),
  xlab = paste0('Partial pressure of CO2 in the mesophyll (', inputs$units$PCm, ')'),
  ylab = paste0('Net CO2 assimilation rate (', assim$units$An, ')')
)

# Finally, we can see whether enzyme activity or light limits overall
# assimilation. In this case, assimilation is always enzyme-limited.
lattice::xyplot(
  Ac + Aj + An ~ inputs[, 'PCm'],
  data = assim$main_data,
  type = 'l',
  grid = TRUE,
  auto = TRUE,
  ylim = c(-5, 40),
  xlab = paste0('Partial pressure of CO2 in the mesophyll (', inputs$units$PCm, ')'),
  ylab = paste0('Net CO2 assimilation rate (', assim$units$An, ')')
)
```

`calculate_c4_assimilation_hyperbola` *Calculate C4 assimilation rates using a hyperbola*

Description

Calculates C4 assimilation rates based on an empirical hyperbolic model. This function can accomodate alternative colum names for the variables taken from Licor files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_c4_assimilation_hyperbola(
  exdf_obj,
  c4_curvature,
  c4_slope,
  rL,
  Vmax,
  ci_column_name = 'Ci',
  hard_constraints = 0,
  perform_checks = TRUE,
  return_exdf = TRUE
)
```

Arguments

`exdf_obj`	An `exdf` object.
`c4_curvature`	The empirical curvature parameter of the hyperbola (`dimensionless`). If `c4_curvature` is not a number, then there must be a column in `exdf_obj` called `c4_curvature` with appropriate units. A numeric value supplied here will overwrite the values in the `c4_curvature` column of `exdf_obj` if it exists.
`c4_slope`	The empirical slope parameter of the hyperbola (`mol m^(-2) s^(-1)`). If `c4_slope` is not a number, then there must be a column in `exdf_obj` called `c4_slope` with appropriate units. A numeric value supplied here will overwrite the values in the `c4_slope` column of `exdf_obj` if it exists.
`rL`	The respiration rate, expressed in `micromol m^(-2) s^(-1)`. If `rL` is not a number, then there must be a column in `exdf_obj` called `rL` with appropriate units. A numeric value supplied here will overwrite the values in the `rL` column of `exdf_obj` if it exists.
`Vmax`	The maximum gross assimilation rate, expressed in `micromol m^(-2) s^(-1)`. If `Vmax` is not a number, then there must be a column in `exdf_obj` called `Vmax` with appropriate units. A numeric value supplied here will overwrite the values in the `Vmax` column of `exdf_obj` if it exists.
`ci_column_name`	The name of the column in `exdf_obj` that contains the intercellular CO2 concentration, expressed in `micromol mol^(-1)`.

hard_constraints
: An integer numerical value indicating which types of hard constraints to place on the values of input parameters; see below for more details.

perform_checks
: A logical value indicating whether to check units for the required columns. This should almost always be TRUE. The option to disable these checks is only intended to be used when fit_c4_aci_hyperbola calls this function, since performing these checks many times repeatedly slows down the fitting procedure.

return_exdf
: A logical value indicating whether to return an exdf object. This should almost always be TRUE. The option to return a vector is mainly intended to be used when fit_c4_aci_hyperbola calls this function, since creating an exdf object to return will slow down the fitting procedure.

Details

General Description of the Model

In contrast to the mechanistic model implemented in calculate_c4_assimilation, this is a simple empirical model for C4 assimilation based on a four-parameter hyperbola. In this model, the net CO2 assimilation rate (An) is given by

An = Ag - rL,

where Ag is the gross assimilation rate and rL is the respiration rate. In turn, Ag is given by the smaller root of the following quadratic equation:

curvature * Ag^2 - (Vinitial + Vmax) * Ag + Vinitial * Vmax = 0,

where 0 <= curvature <= 1 is an empirical curvature factor, Vmax is the maximum gross assimilation rate, and Vinitial represents the initial response of Ag to increases in the intercellular CO2 concentration (Ci):

Vinitial = slope * Ci.

Here the slope is another empirical factor.

By including the respiration offset, it is also possible to define two other quantities: the maximum net CO2 assimilation rate (Amax) and the initial net CO2 assimilation rate (Ainitial). These are given by

Amax = Vmax - rL

and

Ainitial = Vinitial - rL.

Overall, this model exhibits a linear response of An to Ci at low Ci, a flat plateau of An at high Ci, and a smooth transition between these regions. The sharpess of the transition is set by the curvature. When curvature = 1, the model simplifies to

An = min{Vinitial, Vmax} - rL = min{Ainitial, Amax}.

As the curvature increases to 1, the transition becomes smoother. When the curvature is not zero, An approaches Amax asymptotically, and may not reach Amax at a reasonable value of Ci.

Code implementation

In this function, curvature and slope above are referred to as c4_curvature and c4_slope to avoid any potential ambiguity with other models that may also have curvature and slope parameters.

Temperature response

Because this model does not represent any photosynthetic mechanisms, temperature response functions are not applied.

Hard constraints

Most input parameters to the this model have hard constraints on their values which are set by their interpretation; for example, `Vmax` cannot be negative and `c4_curvature` must lie between 0 and 1. Yet, because of measurement noise, sometimes it is necessary to use values outside these ranges when fitting an A-Ci curve with `fit_c4_aci_hyperbola`. To accomodate different potential use cases, it is possible to selectively apply these hard constraints by specifying different values of the `hard_constraints` input argument:

- `hard_constraints = 0`: No constraints are applied.
- `hard_constraints = 1`: Checks whether all `Ci` values are non-negative.
- `hard_constraints = 2`: Includes the same constraints as when `hard_constraints` is 1, which additional constraints on the parameters that can be fitted. For example, `Vmax` must be non-negative and `c4_curvature` must lie between 0 and 1.

If any input values violate any of the specified constraints, an error message will be thrown.

Value

The return value depends on the value of `return_exdf`:

- If `return_exdf` is `TRUE`, the return value is an `exdf` object with the following columns: `Ag`, `Ainitial`, `Amax`, `An`, `c4_curvature`, `c4_slope`, `rL`, `Vinitial`, `Vmax`, and `c4_assimilation_hyperbola_msg`. Most of these are calculated as described above, while several are copies of the input arguments with the same name. The `c4_assimilation_hyperbola_msg` is usually blank but may contain information about any issues with the inputs. The category for each of these new columns is `calculate_c4_assimilation_hyperbola` to indicate that they were created using this function.
- If `return_exdf` is `FALSE`, the return value is a numeric vector containing the calculated values of `An`.

Examples

```
# Simulate a C4 A-Ci curve and plot the net assimilation rate.
npts <- 101

inputs <- exdf(data.frame(
  Ci = seq(0, 1000, length.out = npts),
  total_pressure = 1
))

inputs <- document_variables(
  inputs,
  c('', 'Ci',             'micromol mol^(-1)'),
  c('', 'total_pressure', 'bar')
)

assim <- calculate_c4_assimilation_hyperbola(inputs, 0.8, 0.5, 1.0, 55)
```

```
lattice::xyplot(
  Ainitial + Amax + An ~ inputs[, 'Ci'],
  data = assim$main_data,
  type = 'l',
  grid = TRUE,
  auto = TRUE,
  ylim = c(-5, 65),
  xlab = paste0('Intercellular CO2 concentration (', inputs$units$Ci, ')'),
  ylab = paste0('Net CO2 assimilation rate (', assim$units$An, ')')
)
```

`calculate_gamma_star` *Calculate Gamma_star from Rubisco specificity*

Description

Calculates the CO2 compensation point in the absence of non-photorespiratory CO2 release (`Gamma_star`) from the Rubisco specificity (on a molarity basis), the oxygen concentration (as a percentage), and the temperature-dependent solubilities of CO2 and O2 in H2O.

Usage

```
calculate_gamma_star(
  exdf_obj,
  alpha_pr = 0.5,
  oxygen_column_name = 'Oxygen',
  rubisco_specificity_column_name = 'rubisco_specificity_tl',
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

`exdf_obj`	An `exdf` object.
`alpha_pr`	The number of CO2 molecules released by the photorespiratory cycle following each RuBP oxygenation.
`oxygen_column_name`	The name of the column in `exdf_obj` that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be `percent`.
`rubisco_specificity_column_name`	The name of the column in `exdf_obj` that contains the Rubisco specificity `S_aq` at the leaf temperature; the units must be `M / M`, where the molarity `M` is moles of solute per mole of solvent.
`tleaf_column_name`	The name of the column in `exdf_obj` that contains the leaf temperature in `degrees C`.

Details

The CO2 compensation point in the absence of non-photorespiratory CO2 release (Gamma_star) is the partial pressure of CO2 in the chloroplast at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation; this quantity plays a key role in many photosynthesis calculations. One way to calculate its value is to use its definition, which can be found in many places, such as Equation 2.17 from von Caemmerer (2000):

Gamma_star = alpha_pr * O / S,

where O is the partial pressure (or mole fraction) of oxygen in the chloroplast, S is the Rubisco specificity on a gas basis, and alpha_pr is the number of CO2 molecules released by the photorespiratory cycle following each RuBP oxygenation (usually assumed to be 0.5).

The Rubisco specificity is often measured from an aqueous solution where the concentrations of O2 and CO2 are specified as molarities (moles of dissolved CO2 or O2 per mole of H2O). In this context, the equation above becomes

Gamma_star_aq = alpha_pr * O_aq / S_aq,

where Gamma_star_aq and O_aq are the molarities of CO2 and O2 corresponding to Gamma_star and O under the measurement conditions and S_aq is the specificity on a molarity basis.

Henry's law can be used to relate these two versions of the equation; Henry's law states that the concentration of dissolved gas is proportional to the partial pressure of that gas outside the solution. The proportionality factor H is called Henry's constant (or sometimes the solubility), and its value depends on the temperature, gas species, and other factors. Using Henry's law, we can write Gamma_star_aq = Gamma_star_aq * H_CO2 and O = O_aq * H_O2, where H_CO2 is Henry's constant for CO2 dissolved in H2O and H_O2 is Henry's constant for O2 dissolved in H2O. With these replacements, we can re-express the equation above as:

Gamma_star / H_CO2 = alpha_pr * (O / H_O2) / S_aq

Solving for Gamma_star, we see that:

Gamma_star = (alpha_pr * O / S_aq) * (H_CO2 / H_O2).

In other words, both the Rubisco specificity (as measured on a molarity basis) and the ratio of the two Henry's constants (H_CO2 / H_O2) play a role in determining Gamma_star. This equation also shows that it is possible to relate S (the specificity on a gas concentration basis) and S_aq as S = S_aq * H_O2 / H_CO2.

The values of H_O2 and H_CO2 can be calculated from the temperature using Equation 18 from Tromans (1998) and Equation 4 from Carroll et al. (1991), respectively.

In calculate_gamma_star, it is assumed that the value of specificity S_aq was was measured or otherwise determined at the leaf temperature; the leaf temperature is only used to determine the values of the two Henry's constants. Sometimes it is necessary to calculate the temperature-dependent value of the specificity using an Arrhenius equation; this can be accomplished via the calculate_temperature_response_arrhenius function from PhotoGEA.

Finally, it is important to note that Gamma_star can also be directly calculated using an Arrhenius equation, rather than using the oxygen concentration and the specificity. The best approach for determining a value of Gamma_star in any particular situation will generally depend on the available information and the measurement conditions.

References:

von Caemmerer, S. "Biochemical Models of Leaf Photosynthesis." (CSIRO Publishing, 2000) [doi:10.1071/9780643103405].

Carroll, J. J., Slupsky, J. D. and Mather, A. E. "The Solubility of Carbon Dioxide in Water at Low Pressure." Journal of Physical and Chemical Reference Data 20, 1201–1209 (1991) [doi:10.1063/1.555900].

Tromans, D. "Temperature and pressure dependent solubility of oxygen in water: a thermodynamic analysis." Hydrometallurgy 48, 327–342 (1998) [doi:10.1016/S0304386X(98)000073].

Value

An `exdf` object based on `exdf_obj` that includes the following additional columns, calculated as described above: `Gamma_star_tl` (the value of `Gamma_star` at the leaf temperature), `H_CO2`, `H_O2`, and `specificity_gas_basis`. There are many choices for expressing Henry's constant values; here we express them as molalities per unit of pressure: `(mol solute / kg H2O) / Pa`. The category for each of these new columns is `calculate_gamma_star` to indicate that they were created using this function.

Examples

```
# Example 1: Calculate Gamma_star for each point in a gas exchange log file
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
)

licor_data <- get_oxygen_from_preamble(licor_data)

licor_data <- set_variable(
    licor_data,
    'rubisco_specificity_tl',
    'M / M',
    value = 90
)

licor_data <- calculate_gamma_star(licor_data)

licor_data[, c('specificity_gas_basis', 'Oxygen', 'Gamma_star_tl'), TRUE]

# Example 2: Calculate Gamma_star at 21% and 2% oxygen for a Rubisco whose
# specificity was measured to be 100 M / M at 25 degrees C.

exdf_obj <- calculate_gamma_star(
  exdf(
    data.frame(
      Oxygen = c(2, 21),
      rubisco_specificity_tl = c(100, 100),
      TleafCnd = c(25, 25)
    ),
    data.frame(
      Oxygen = 'percent',
      rubisco_specificity_tl = 'M / M',
      TleafCnd = 'degrees C',
      stringsAsFactors = FALSE
    )
```

```
  )
)

exdf_obj[, c('specificity_gas_basis', 'Oxygen', 'Gamma_star_tl'), TRUE]

# Example 3: Here we recreate Figure 1 from Long, S. P. "Modification of the
# response of photosynthetic productivity to rising temperature by atmospheric
# CO2 concentrations: Has its importance been underestimated?" Plant, Cell and
# Environment 14, 729-739 (1991). This is a fairly complicated example where
# Arrhenius constants for Rubisco parameters are determined by fitting
# published data and then used to determine the Rubisco specificity across a
# range of temperatures.

# Specify leaf temperature and oxygen concentration
leaf_temp <- seq(0, 50, by = 0.1)

exdf_obj <- exdf(
  data.frame(
    Oxygen = rep_len(21, length(leaf_temp)),
    TleafCnd = leaf_temp
  ),
  data.frame(
    Oxygen = 'percent',
    TleafCnd = 'degrees C',
    stringsAsFactors = FALSE
  )
)

# Get Arrhenius constants for Rubisco parameters using data from Table 2 of
# Jordan, D. B. and Ogren, W. L. "The CO2/O2 specificity of ribulose
# 1,5-bisphosphate carboxylase/oxygenase" Planta 161, 308-313 (1984).
rubisco_info <- data.frame(
  temperature = c(7,    12,   15,   25,   30,   35),
  Vc          = c(0.13, 0.36, 0.63, 1.50, 1.90, 2.90),
  Kc          = c(2,    3,    4,    11,   14,   19),
  Ko          = c(550,  510,  510,  500,  600,  540),
  Vo          = c(0.24, 0.48, 0.69, 0.77, 1.1,  1.3)
)

rubisco_info$x <- 1 / (8.314e-3 * (rubisco_info$temperature + 273.15))

lm_Vc <- stats::lm(log(Vc) ~ x, data = rubisco_info)
lm_Kc <- stats::lm(log(Kc) ~ x, data = rubisco_info)
lm_Ko <- stats::lm(log(Ko) ~ x, data = rubisco_info)
lm_Vo <- stats::lm(log(Vo) ~ x, data = rubisco_info)

arrhenius_info <- list(
  Vc = list(
    c = as.numeric(lm_Vc$coefficients[1]),
    Ea = -as.numeric(lm_Vc$coefficients[2]),
    units = 'micromol / mg / min'
  ),
  Kc = list(
```

```
    c = as.numeric(lm_Kc$coefficients[1]),
    Ea = -as.numeric(lm_Kc$coefficients[2]),
    units = 'microM'
  ),
  Ko = list(
    c = as.numeric(lm_Ko$coefficients[1]),
    Ea = -as.numeric(lm_Ko$coefficients[2]),
    units = 'microM'
  ),
  Vo = list(
    c = as.numeric(lm_Vo$coefficients[1]),
    Ea = -as.numeric(lm_Vo$coefficients[2]),
    units = 'micromol / mg / min'
  )
)

# Get temperature-dependent values of Rubisco parameters using Arrhenius
# equations
exdf_obj <- calculate_temperature_response_arrhenius(
  exdf_obj,
  arrhenius_info
)

# Calculate temperature-dependent specificity values
exdf_obj <- set_variable(
  exdf_obj,
  'rubisco_specificity_tl',
  units = 'M / M',
  value = exdf_obj[, 'Vc'] * exdf_obj[, 'Ko'] /
    (exdf_obj[, 'Vo'] * exdf_obj[, 'Kc'])
)

# Calculate Gamma_star and Henry constants
exdf_obj <- calculate_gamma_star(exdf_obj)

# Make a plot similar to Figure 1 from Long (1991)
lattice::xyplot(
  rubisco_specificity_tl + H_CO2 / H_O2 ~ TleafCnd,
  data = exdf_obj$main_data,
  auto = TRUE,
  grid = TRUE,
  type = 'l',
  xlim = c(0, 50),
  ylim = c(0, 250),
  xlab = "Temperature [ degrees C ]",
 ylab = "Rubisco specificity or ratio of Henry's constants (H_CO2 / H_O2)\n[ dimensionless ]"
)

# We can also make a plot of Gamma_star across this range
lattice::xyplot(
  Gamma_star_tl ~ TleafCnd,
  data = exdf_obj$main_data,
  auto = TRUE,
```

```
  grid = TRUE,
  type = 'l',
  xlim = c(0, 50),
  ylim = c(0, 120),
  xlab = "Temperature [ degrees C ]",
  ylab = paste('Gamma_star at leaf temperature [', exdf_obj$units$Gamma_star_tl, ']')
)
```

`calculate_gas_properties` *Calculate gas properties that are typically not included in Licor files*

Description

Calculates gas properties that are typically not included in Licor files. This function can accomodate alternative column names for the variables taken from the Licor file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_gas_properties(
  licor_exdf,
  a_column_name = 'A',
  ca_column_name = 'Ca',
  total_pressure_column_name = 'total_pressure',
  e_column_name = 'E',
  gbw_column_name = 'gbw',
  gsw_column_name = 'gsw',
  h2o_s_column_name = 'H2O_s',
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

`licor_exdf`	An `exdf` object representing data from a Licor gas exchange measurement system.
`a_column_name`	The name of the column in `licor_exdf` that contains the net assimilation in `micromol m^(-2) s^(-1)`.
`ca_column_name`	The name of the column in `licor_exdf` that contains the ambient CO2 concentration in the chamber in `micromol mol^(-1)`.
`total_pressure_column_name`	The name of the column in `licor_exdf` that contains the total pressure in `bar`.
`e_column_name`	The name of the column in `licor_exdf` that contains the transpiration rate in `mol m^(-2) s^(-1)`.

`gbw_column_name`
: The name of the column in `licor_exdf` that contains the boundary layer conductance to water vapor in `mol m^(-2) s^(-1)`.

`gsw_column_name`
: The name of the column in `licor_exdf` that contains the stomatal conductance to water vapor in `mol m^(-2) s^(-1)`.

`h2o_s_column_name`
: The name of the column in `licor_exdf` that contains the sample cell H2O concentration in `mmol mol^(-1)`.

`tleaf_column_name`
: The name of the column in `licor_exdf` that contains the leaf temperature in `degrees C`.

Details

By default, a Licor file provides the following gas concentrations and conductances:

- Water vapor conductance to diffusion through the stomata (`gsw`).
- Water vapor conductance to diffusion through the boundary layer (`gbw`).
- Water vapor conductance to diffusion from the leaf's intercellular spaces to the ambient air; in other words, the total conductance to water vapor (`gtw`).
- Water vapor concentration in the sample cell (`H2O_s`).
- CO2 conductance to diffusion from the leaf's intercellular spaces to the ambient air; in other words, the total conductance to CO2 (`gtc`).
- CO2 concentration in the sample cell, corrected for any chamber leaks (`Ca`).
- CO2 concentration in the leaf's intercellular spaces (`Ci`).

However, it is sometimes helpful to know the "missing" conductances and concentrations, for example, when calculating mesophyll conductances or Ball-Berry parameters. This function adds these missing values, along with a few related water vapor properties:

- Water vapor concentration at the sample surface (`H2O_surf`).
- Water vapor concentration in the leaf's intercellular spaces (`H2O_i`).
- Saturation water vapor pressure at the leaf temperature (`SVPleaf`).
- Relative humidity at the leaf surface (`RHleaf`).
- CO2 conductance to diffusion through the stomata (`gsc`).
- CO2 conductance to diffusion through the boundary layer (`gbc`).
- CO2 concentration at the leaf surface (`Cs`).

Equations used for these calculations

The equations used to calculate these quantities can be found in the Licor Li-6800 manual (Appendix C), which relies heavily on Appendix 2 of the following paper: von Caemmerer, S. & Farquhar, G. D. "Some relationships between the biochemistry of photosynthesis and the gas exchange of leaves" Planta **153**, 376–387 (1981) [doi:10.1007/BF00384257]

Equation C-79 in the Licor manual describes the total flow of water vapor from the leaf interior to the ambient air using `gtw`, `H2O_i`, `H2O_s`, and the transpiration rate `E`:

(1) `gtw = E * (1000 - (H2O_i + H2O_s) / 2) / (H2O_i - H2O_s)`

In steady-state conditions, the flux of H2O molecules across any portion of the gas flow is identical to `E`, so we can also apply this equation to the flow of water vapor from the leaf surface to the ambient air:

(2) `gbw = E * (1000 - (H2O_surf + H2O_s) / 2) / (H2O_surf - H2O_s)`

Equation (2) can be solved for `H2O_surf`:

(3) `H2O_surf = (E * (1000 - H2O_s / 2) + gbw * H2O_s) / (gbw + E / 2)`

Equation C-70 in the Licor manual describes how to calculate saturation water vapor pressure from air temperature. At the leaf surface, the air temperature should be the same as the leaf temperature (`Tleaf`; in degrees C), so we can determine `SVPleaf` using Equation C-70 as follows:

(4) `SVPleaf = 0.6135 * e^((17.502 * Tleaf) / (240.97 + Tleaf))`

For gas exchange measurements, we assume that water vapor is saturated in the leaf's intecellular spaces, so we can determine `H2O_i` from `SVPleaf` and the relationship between partial pressure and molar gas concentration:

(5) `H2O_i = SVPleaf / Pcham = SVPleaf / (Pa + deltaPcham)`

where `Pcham` is th total pressure in the sample chamber, `Pa` is the atmospheric pressure, and `deltaPcham` is the chamber overpressure. These are related by `Pcham = Pa + deltaPcham`.

The relative humidity at the leaf surface `RHleaf` can be determined from `H2O_surf` and `SVPleaf` using the definitions of relative humidity and partial pressure:

(6) `RHleaf = Pwl / SVPleaf = H2O_surf * (Pa + deltaPcham) / SVPleaf`

where `Pwl`, the partial pressure of H2O at the leaf surface, is given by `H2O_surf * Pcham`.

The CO2 conductances through the stomata and boundary layer can be determined from the corresponding H2O conductances using the ratios of molecular diffusivities for the two molecules, as explained in the vicinty of Equation C-106 in the Licor manual:

(7) `gsc = gsw / 1.6`

(8) `gbc = gbw / 1.37`

Equation C-105 in the Licor manual describes the flow of CO2 from the ambient air to the intercellular spaces:

(9) `C_i = ((gtc - E / 2) * Ca - A) / (gtc + E / 2)`

where we have replaced `C_s` (the CO2 concentration in the sample chamber) with `Ca` for clarity. In steady state conditions, the flows of H2O and CO2 are identical to `E` and `A`, respectively, so we can also apply this equation to the flow of CO2 from the ambient air to the leaf surface:

(10) `Csurface = ((gbc - E / 2) * Ca - A) / (gbc + E / 2)`

This function uses Equations (3)-(8) and (10) to calculate the desired values.

Value

An `exdf` object based on `licor_exdf` that includes the following additional columns, calculated as described above: `H2O_surf`, `SVPleaf`, `H2O_i`, `RHleaf`, `gsc`, `gbc`, and `Csurface`. The category for each of these new columns is `calculate_gas_properties` to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package, calculate the
# total pressure, and calculate additional gas properties.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_gas_properties(licor_file)

licor_file$units$RHleaf      # View the units of the new `RHleaf` column
licor_file$categories$RHleaf # View the category of the new `RHleaf` column
licor_file[,'RHleaf']        # View the values of the new `RHleaf` column
```

`calculate_gm_busch` *Calculate mesophyll conductance to CO2 diffusion*

Description

Calculates mesophyll conductance to CO2 diffusion (`gmc`) from combined gas exchange and isotope discrimination measurements as described in Busch et al. (2020). This function can accomodate alternative colum names for the variables taken from `exdf_obj`; it also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_gm_busch(
  exdf_obj,
  e = -3,
  f = 11,
  e_star_equation = 20,
  gm_type = 'dis',
  a_bar_column_name = 'a_bar',
  a_column_name = 'A',
  ci_column_name = 'Ci',
  co2_s_column_name = 'CO2_s',
  csurface_column_name = 'Csurface',
  delta_c13_r_column_name = 'delta_C13_r',
  delta_obs_growth_column_name = 'Delta_obs_growth',
  delta_obs_tdl_column_name = 'Delta_obs_tdl',
  gamma_star_column_name = 'Gamma_star_tl',
  rl_column_name = 'RL',
  total_pressure_column_name = 'total_pressure',
  t_column_name = 't'
)
```

Arguments

exdf_obj	An exdf object.
e	The isotopic fractionation during day respiration in ppt.
f	The isotopic fractionation during photorespiration in ppt.
e_star_equation	The equation from Busch et al. (2020) to use for calculating e_star; must be 19 or 20.
gm_type	Determines whether day respiration is assumed to be isotopically connected to the CBB cycle (gm_type = 'con') or isotopically disconnected from the CBB cycle (gm_type = 'dis'). This choice will determine which equations are used to calculate mesophyll conductance; when gm_type is 'con', Equations 2 and 21 will be used; otherwise, Equations 13 and 22 will be used.
a_bar_column_name	The name of the column in exdf_obj that contains the weighted isotopic fractionation across the boundary layer and stomata in ppt. Values of a_bar are typically calculated using calculate_ternary_correction.
a_column_name	The name of the column in exdf_obj that contains the net CO2 assimilation rate in micromol m^(-2) s^(-1).
ci_column_name	The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).
co2_s_column_name	The name of the column in exdf_obj that contains the CO2 concentration in the sample line (outgoing air) in micromol mol^(-1).
csurface_column_name	The name of the column in exdf_obj that contains the CO2 concentration at the leaf surface in micromol mol^(-1). Values of Csurface are typically calculated using calculate_gas_properties.
delta_c13_r_column_name	The name of the column in exdf_obj that contains the CO2 isotope ratio in the reference line (incoming air) in ppt.
delta_obs_growth_column_name	The name of the column in exdf_obj that contains the observed discrimination under the typical CO2 concentration in the plant's environment during its growth (in ppt). This is only required when using Equation 20 for e_star (see e_star_equation).
delta_obs_tdl_column_name	The name of the column in exdf_obj that contains the observed isotope discrimination values in ppt.
gamma_star_column_name	The name of the column in exdf_obj that contains the chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at leaf temperature, expressed in micromol mol^(-1). Values of Gamma_star at leaf temperature are typically calculated using calculate_gamma_star or calculate_temperature_response.

`rl_column_name` The name of the column in `exdf_obj` that contains the rate of non-photorespiratory CO2 release in the light, in `micromol m^(-2) s^(-1)`.

`total_pressure_column_name`
The name of the column in `exdf_obj` that contains the total pressure in `bar`.

`t_column_name` The name of the column in `exdf_obj` that contains the ternary correction factor (`dimensionless`). Values of `t` are typically calculated using `calculate_ternary_correction`

Details

This function uses a model for photosynthetic discrimination against 13C in C3 plants to determine mesophyll conductance values as described in Busch et al. (2020). That paper provides two alternate ways to calculate `e_star`, and two alternate ways to calculate mesophyll conductance `gmc`; this function allows the user to choose between them. In more detail:

- Isotopic fractionation due to day respiration (`e_prime = e + e_star`) is calculated with `e_star` given by either Equation 19 or 20 depending on the value of `e_star_equation`.
- Isotopic discrimination assuming infinite mesophyll conductance (`Delta_i`) is calculated by setting `Cc = Ci` in either Equation 2 or 13, depending on the value of `gm_type`.
- Mesophyll conductance to CO2 (`gmc`) is calculated using either Equation 21 or 22, depending on the value of `gm_type`.

Note 1: Setting `e_star_equation = 19` and `gm_type = 'con'` should produce identical or similar results to `calculate_gm_ubierna`.

Note 2: Using `e_star_equation = 20` and `gm_type = 'dis'` is expected to be more accurate, as discussed in Busch et al. (2020); however, be aware that this method requires a value for `Delta_obs_growth`, which may not always be available unless it is intentionally measured.

References:

Busch, F. A., Holloway-Phillips, M., Stuart-Williams, H. and Farquhar, G. D. "Revisiting carbon isotope discrimination in C3 plants shows respiration rules when photosynthesis is low." Nat. Plants 6, 245–258 (2020) [doi:10.1038/s4147702006066].

Value

An `exdf` object based on `exdf_obj` that includes the following additional columns, calculated as described above: `e_prime`, `e_star`, `Delta_i`, and `gmc`, as well as the values of a few intermediate calculations such as `Delta_i_term_1` and `Delta_i_term_2`. The category for each of these new columns is `calculate_gm_busch` to indicate that they were created using this function.

Examples

```
## In this example we load gas exchange and TDL data files, calibrate the TDL
## data, pair the data tables together, and then calculate mesophyll conductance

# Read the TDL data file, making sure to interpret the time zone as US Central
# time
tdl_data <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_for_gm.dat'),
  'TIMESTAMP',
```

```
  list(tz = 'America/Chicago')
)

# Identify cycles within the TDL data
tdl_data <- identify_tdl_cycles(
  tdl_data,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Use reference tanks to calibrate the TDL data
processed_tdl <- consolidate(by(
  tdl_data,
  tdl_data[, 'cycle_num'],
  process_tdl_cycle_erml,
  noaa_valve = 2,
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
  calibration_3_valve = 26,
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
))

# Read the gas exchange data, making sure to interpret the time stamp in the US
# Central time zone
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time',
  list(tz = 'America/Chicago')
)

# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# Pair the Licor and TDL data by locating the TDL cycle corresponding to each
# Licor measurement
licor_data <- pair_gasex_and_tdl(licor_data, processed_tdl$tdl_data)

# Calculate total pressure (needed for calculate_gas_properties)
licor_data <- calculate_total_pressure(licor_data)

# Calculate Csurface (needed for calculate_ternary_correction)
licor_data <- calculate_gas_properties(licor_data)

# Calculate ternary correction
licor_data <- calculate_ternary_correction(licor_data)
```

```
# Set Rubisco specificity (needed for calculate_gamma_star)
licor_data <- set_variable(
    licor_data,
    'rubisco_specificity_tl',
    'M / M',
    value = 90
)

# Calculate Gamma_star (needed for calculate_gm_busch)
licor_data <- calculate_gamma_star(licor_data)

# Calculate isotope discrimination (needed for calculate_gm_busch)
licor_data <- calculate_isotope_discrimination(licor_data)

# Set Delta_obs_growth to the average of Delta_obs_tdl over the first 6 points,
# where the ambient CO2 concentration was set to the atmospheric value (420 ppm)
# (needed for calculate_gm_busch).
licor_data <- set_variable(
  licor_data,
  'Delta_obs_growth',
  'ppt',
  value = mean(licor_data[1:6, 'Delta_obs_tdl'])
)

# Set respiration (needed for calculate_gm_busch)
licor_data <- set_variable(
  licor_data,
  'RL',
  'micromol m^(-2) s^(-1)',
  value = 1.2
)

# Calculate mesophyll conductance
licor_data <- calculate_gm_busch(licor_data)

# Calculate Cc using the new values of mesophyll conductance
licor_data <- calculate_temperature_response(
  licor_data,
  c3_temperature_param_flat['gmc_norm']
)

licor_data <- set_variable(
  licor_data,
  'gmc_at_25',
  units = licor_data$units$gmc,
  value = licor_data[, 'gmc']
)

licor_data <- apply_gm(licor_data)

# View some of the results
licor_data[, c('replicate', 'CO2_s', 'Delta_obs_tdl', 'e_prime', 'gmc', 'Ci', 'Cc')]
```

`calculate_gm_ubierna` *Calculate mesophyll conductance to CO2 diffusion*

Description

Calculates mesophyll conductance to CO2 diffusion (gmc) from combined gas exchange and isotope discrimination measurements as described in Ubierna et al. (2018). This function can accomodate alternative colum names for the variables taken from `exdf_obj`; it also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_gm_ubierna(
  exdf_obj,
  e = -3,
  f = 11,
  a_bar_column_name = 'a_bar',
  a_column_name = 'A',
  ci_column_name = 'Ci',
  co2_s_column_name = 'CO2_s',
  csurface_column_name = 'Csurface',
  delta_c13_r_column_name = 'delta_C13_r',
  delta_obs_tdl_column_name = 'Delta_obs_tdl',
  gamma_star_column_name = 'Gamma_star_tl',
  rl_column_name = 'RL',
  total_pressure_column_name = 'total_pressure',
  t_column_name = 't'
)
```

Arguments

`exdf_obj`	An `exdf` object.
`e`	The isotopic fractionation during day respiration in `ppt`.
`f`	The isotopic fractionation during photorespiration in `ppt`.
`a_bar_column_name`	The name of the column in `exdf_obj` that contains the weighted isotopic fractionation across the boundary layer and stomata in `ppt`. Values of `a_bar` are typically calculated using `calculate_ternary_correction`.
`a_column_name`	The name of the column in `exdf_obj` that contains the net CO2 assimilation rate in `micromol m^(-2) s^(-1)`.
`ci_column_name`	The name of the column in `exdf_obj` that contains the intercellular CO2 concentration in `micromol mol^(-1)`.
`co2_s_column_name`	The name of the column in `exdf_obj` that contains the CO2 concentration in the sample line (outgoing air) in `micromol mol^(-1)`.

csurface_column_name
: The name of the column in exdf_obj that contains the CO2 concentration at the leaf surface in micromol mol^(-1). Values of Csurface are typically calculated using calculate_gas_properties.

delta_c13_r_column_name
: The name of the column in exdf_obj that contains the CO2 isotope ratio in the reference line (incoming air) in ppt.

delta_obs_tdl_column_name
: The name of the column in exdf_obj that contains the observed isotope discrimination values in ppt.

gamma_star_column_name
: The name of the column in exdf_obj that contains the chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at leaf temperature, expressed in micromol mol^(-1). Values of Gamma_star at leaf temperature are typically calculated using calculate_gamma_star or calculate_temperature_response.

rl_column_name
: The name of the column in exdf_obj that contains the rate of non-photorespiratory CO2 release in the light, in micromol m^(-2) s^(-1).

total_pressure_column_name
: The name of the column in exdf_obj that contains the total pressure in bar.

t_column_name
: The name of the column in exdf_obj that contains the ternary correction factor (dimensionless). Values of t are typically calculated using calculate_ternary_correction

Details

This function uses the comprehensive model for photosynthetic discrimination against 13C in C3 plants to calculate mesophyll conductance, as described in Ubierna et al. (2018). In particular, the following equations from that source are implemented in the code:

- Isotopic fractionation due to day respiration (e_prime) is calculated using Equations 28 and 30.
- Isotopic discrimination due to photorespiration (Delta_f), due to day respiration (Delta_e), and that would occur if Ci = Cc in the absence of any respiratory fractionation (Delta_i) are calculated using Equations 34, 33, and 31, respectively.
- Mesophyll conductance to CO2 diffusion (gmc) is calculated using Equation 44. This equation is broken up into two factors called Delta_difference and equation_top which are separately returned in the output from calculate_gm_ubierna.

For an alternative method for calculating gmc, see calculate_gm_busch.

References:

Ubierna, N., Holloway-Phillips, M.-M. and Farquhar, G. D. "Using Stable Carbon Isotopes to Study C3 and C4 Photosynthesis: Models and Calculations." in Photosynthesis: Methods and Protocols (ed. Covshoff, S.) 155–196 (Springer, 2018) [doi:10.1007/9781493977864_10].

Value

An exdf object based on exdf_obj that includes the following additional columns, calculated as described above: e_prime, Delta_i, Delta_e, Delta_f, Delta_difference, equation_top, and gmc. The category for each of these new columns is calculate_gm_ubierna to indicate that they were created using this function.

Examples

```
## In this example we load gas exchange and TDL data files, calibrate the TDL
## data, pair the data tables together, and then calculate mesophyll conductance

# Read the TDL data file, making sure to interpret the time zone as US Central
# time
tdl_data <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_for_gm.dat'),
  'TIMESTAMP',
  list(tz = 'America/Chicago')
)

# Identify cycles within the TDL data
tdl_data <- identify_tdl_cycles(
  tdl_data,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Use reference tanks to calibrate the TDL data
processed_tdl <- consolidate(by(
  tdl_data,
  tdl_data[, 'cycle_num'],
  process_tdl_cycle_erml,
  noaa_valve = 2,
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
  calibration_3_valve = 26,
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
))

# Read the gas exchange data, making sure to interpret the time stamp in the US
# Central time zone
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time',
  list(tz = 'America/Chicago')
)
```

```
# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# Pair the Licor and TDL data by locating the TDL cycle corresponding to each
# Licor measurement
licor_data <- pair_gasex_and_tdl(licor_data, processed_tdl$tdl_data)

# Calculate total pressure (needed for calculate_gas_properties)
licor_data <- calculate_total_pressure(licor_data)

# Calculate Csurface (needed for calculate_ternary_correction)
licor_data <- calculate_gas_properties(licor_data)

# Calculate ternary correction
licor_data <- calculate_ternary_correction(licor_data)

# Set Rubisco specificity (needed for calculate_gamma_star)
licor_data <- set_variable(
    licor_data,
    'rubisco_specificity_tl',
    'M / M',
    value = 90
)

# Calculate Gamma_star (needed for calculate_gm_ubierna)
licor_data <- calculate_gamma_star(licor_data)

# Calculate isotope discrimination (needed for calculate_gm_ubierna)
licor_data <- calculate_isotope_discrimination(licor_data)

# Set respiration (needed for calculate_gm_ubierna)
licor_data <- set_variable(
  licor_data,
  'RL',
  'micromol m^(-2) s^(-1)',
  value = 1.2
)

# Calculate mesophyll conductance
licor_data <- calculate_gm_ubierna(licor_data)

# Calculate Cc using the new values of mesophyll conductance
licor_data <- calculate_temperature_response(
  licor_data,
  c3_temperature_param_flat['gmc_norm']
)

licor_data <- set_variable(
  licor_data,
  'gmc_at_25',
  units = licor_data$units$gmc,
  value = licor_data[, 'gmc']
```

```
)

licor_data <- apply_gm(licor_data)

# View some of the results
licor_data[, c('replicate', 'CO2_s', 'Delta_obs_tdl', 'gmc', 'Ci', 'Cc')]
```

`calculate_isotope_discrimination` *Calculate photosynthetic isotope discrimination*

Description

Calculates photosynthetic carbon isotope discrimination from combined gas exchange and tunable diode laser absorption spectroscopy measurements.

Usage

```
calculate_isotope_discrimination(
  exdf_obj,
  co2_r_column_name = 'CO2_r',
  co2_s_column_name = 'CO2_s',
  delta_C13_r_column_name = 'delta_C13_r',
  delta_C13_s_column_name = 'delta_C13_s',
  h2o_r_column_name = 'H2O_r',
  h2o_s_column_name = 'H2O_s',
  tdl_12C_r_column_name = 'calibrated_12c_r',
  tdl_12C_s_column_name = 'calibrated_12c_s'
)
```

Arguments

`exdf_obj`	An `exdf` object representing combined data from a gas exchange + isotope discrimination measurement system. Typically `exdf_obj` is produced by calling `pair_gasex_and_tdl`.
`co2_r_column_name`	The name of the column in `exdf_obj` that contains the CO2 concentration in the gas exchange reference line (incoming air) as measured by the gas exchange system in `micromol mol^(-1)`.
`co2_s_column_name`	The name of the column in `exdf_obj` that contains the CO2 concentration in the gas exchange sample line (outgoing air) in `micromol mol^(-1)`.
`delta_C13_r_column_name`	The name of the column in `exdf_obj` that contains the CO2 isotope ratio in the gas exchange reference line (incoming air) in `ppt`.

delta_C13_s_column_name
: The name of the column in exdf_obj that contains the CO2 isotope ratio in the gas exchange sample line (outgoing air) in ppt.

h2o_r_column_name
: The name of the column in exdf_obj that contains the H2O concentration in the gas exchange reference line (incoming air) as measured by the gas exchange system in mmol mol^(-1).

h2o_s_column_name
: The name of the column in exdf_obj that contains the H2O concentration in the gas exchange sample line (outgoing air) as measured by the gas exchange system in mmol mol^(-1).

tdl_12C_r_column_name
: The name of the column in exdf_obj that contains the 12CO2 concentration in the gas exchange reference line (incoming air) as measured by the TDL in ppm.

tdl_12C_s_column_name
: The name of the column in exdf_obj that contains the 12CO2 concentration in the gas exchange sample line (outgoing air) as measured by the TDL in ppm.

Details

As described in Ubierna et al. (2018), photosynthetic 13C discrimination can be determined from combined gas exchange and tunable diode laser (TDL) absorption spectroscopy measurements according to:

Delta_obs = xsi * (delta_out - delta_in) / (1 + delta_out - xsi * (delta_out - delta_in)),

where Delta_obs is the observed discrimination, delta_in and delta_out are the carbon isotope ratios in dry air flowing in and out of the leaf chamber. xsi is given by

xsi = C_in / (C_in - C_out),

where C_in and C_out are the mole fractions of 12CO2 in dry air flowing in and out of the leaf chamber. (See equations 5 and 6 in Ubierna et al. (2018)).

In practice, there are multiple options for calculating Delta_obs and xsi because CO2 concentrations are measured by both the gas exchange system and the TDL. For example, we can alternately calculate xsi as xsi_tdl = C_in_tdl / (C_in_tdl - C_out_tdl) or xsi_gasex = C_in_gasex / (C_in_gasex - C_out_gasex). Likewise, we can also calculate Delta_obs_tdl using xsi_tdl or Delta_obs_gasex using xsi_gasex. The TDL values are typically preferred in subsequent calculations, but it can be useful to compare the two different versions as a consistency check; the TDL and gas exchange values should be similar to each other.

There are two subtelties associated with xsi_gasex. One is that the gas exchange system generally measures the total CO2 concentration, not just the 12CO2 concentration. Typically there is much less 13CO2 than 12CO2 so this is usually not a large source of error.

The other issue is that the gas exchange system generally measures CO2 concentrations in wet air. Thus, it is important to use "corrected" values of CO2 concentrations that account for the "dilution effect" due to water vapor in the air. This effect is described in the Licor LI-6400 manual: "This is a correction we don't do, at least when computing CO2 concentration in the LI-6400. The dilution effect is simply this: as you add molecules of a gas (water vapor, for example) to a mixture, the fraction of that mixture that is made up of something else (mole fraction of CO2, for instance) has to decrease, since the total number of molecules in the mixture has increased. Now for an airsteam

flowing though a chamber containing a transpiring leaf (or in a chamber sitting on moist soil), there very definitely is dilution. However, we ignore that effect when computing CO2 concentration, but account for it when computing photosynthetic rate (or soil CO2 efflux). Thus, the LI-6400 IRGA is always indicating the actual CO2 concentration, not what the CO2 concentration would be if there were no water vapor in it."

To account for the dilution effect, we define a "corrected" CO2 concentration as `CO2_corrected = CO2 / (1 - H2O)`, where `H2O` is the water vapor concentration in the air. Note: the TDL always measures concentrations in dry air, so no correction is required.

References:

Ubierna, N., Holloway-Phillips, M.-M. and Farquhar, G. D. "Using Stable Carbon Isotopes to Study C3 and C4 Photosynthesis: Models and Calculations." in Photosynthesis: Methods and Protocols (ed. Covshoff, S.) 155–196 (Springer, 2018) [doi:10.1007/9781493977864_10].

Value

An `exdf` object based on `exdf_obj` that includes several new columns: `CO2_r_corrected`, `CO2_s_corrected`, `Delta_obs_gasex`, `Delta_obs_tdl`, `xsi_gasex`, and `xsi_tdl`.

Examples

```
## In this example we load gas exchange and TDL data files, calibrate the TDL
## data, pair the data tables together, and then calculate isotope
## discrimination

# Read the TDL data file, making sure to interpret the time zone as US Central
# time
tdl_data <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_for_gm.dat'),
  'TIMESTAMP',
  list(tz = 'America/Chicago')
)

# Identify cycles within the TDL data
tdl_data <- identify_tdl_cycles(
  tdl_data,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Use reference tanks to calibrate the TDL data
processed_tdl <- consolidate(by(
  tdl_data,
  tdl_data[, 'cycle_num'],
  process_tdl_cycle_erml,
  noaa_valve = 2,
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
```

```
  calibration_3_valve = 26,
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
))

# Read the gas exchange data, making sure to interpret the time stamp in the US
# Central time zone
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time',
  list(tz = 'America/Chicago')
)

# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# Pair the Licor and TDL data by locating the TDL cycle corresponding to each
# Licor measurement
licor_data <- pair_gasex_and_tdl(licor_data, processed_tdl$tdl_data)

# Calculate isotope discrimination
licor_data <- calculate_isotope_discrimination(licor_data)

# View some of the results
licor_data[, c('A', 'xsi_gasex', 'xsi_tdl', 'Delta_obs_gasex', 'Delta_obs_tdl')]
```

`calculate_jmax` *Calculate maximum electron transport rate*

Description

Calculates maximum electron transport rates (`Jmax`) from estimates of the electron transport rate (`J`) at particular values of incident light (`Qin`).

This function is typically used after `fit_c3_aci`, `fit_c3_variable_j`, or `fit_c4_aci` is used to estimate values of `J`.

Usage

```
calculate_jmax(
  data_table,
  alpha_j_at_25 = 'column',
  theta_j_at_25 = 'column',
  alpha_j_norm_column_name = 'alpha_j_norm',
  qin_column_name = 'Qin_avg',
  theta_j_norm_column_name = 'theta_j_norm',
  tleaf_column_name = 'TleafCnd_avg',
  ...
)
```

Arguments

data_table	A table-like R object such as a data frame or an exdf.
alpha_j_at_25	The apparent quantum efficiency of electron transport alpha_j at 25 degrees C (dimensionless). If alpha_j_at_25 is not a number, then there must be a column in data_table called alpha_j_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the alpha_j_at_25 column of data_table if it exists.
theta_j_at_25	The empirical curvature parameter theta_j_at_25 at 25 degrees C (dimensionless). If theta_j_at_25 is not a number, then there must be a column in data_table called theta_j_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the theta_j_at_25 column of data_table if it exists.
alpha_j_norm_column_name	The name of the column in data_table that contains the normalized alpha_j values (with units of normalized to alpha_j at 25 degrees C).
qin_column_name	The name of the column in data_table that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).
theta_j_norm_column_name	The name of the column in data_table that contains the normalized theta_j values (with units of normalized to theta_j at 25 degrees C).
tleaf_column_name	The name of the column in data_table that contains the leaf temperature in units of degrees C.
...	Optional arguments; see below.

Details

Basic Requirements:

This function requires that data_table contains columns called J_at_25 and J_tl_avg, as would be included in the output from one of the PhotoGEA fitting functions (fit_c3_aci, fit_c3_variable_j, and fit_c4_aci). These will be used to calculate values of Jmax at 25 degrees C and at leaf temperature.

If any columns for the J confidence intervals are included in data_table (J_at_25_upper, J_at_25_lower, J_tl_avg_upper, or J_tl_avg_lower), the corresponding confidence intervals for Jmax will also be calculated.

By default, this function will take values of alpha_j and theta_j from columns of data_table with the same names.

If data_table is an exdf object, units will be checked for any columns used in the calculations.

Overview of Jmax Calculations:

The potential electron transport rate going to support RuBP regeneration (J) depends on the available light energy. J quickly increases with the incident photosynthetically active photon flux density (Qin) at low light levels, gradually reaching a plateau at high values of Qin. Although other mathematical representations have been used (Walker et al. 2021), this dependence is typically represented as a non-rectangular hyperbola:

J = (I2 + Jmax - sqrt[(I2 + Jmax)^2 - 4 * theta_j * I2 * Jmax]) / (2 * theta_j), (Eq. 1)

where Jmax is the maximum value of J that would be achieved at infinitely large Qin, 0 < theta_j <= 1 is an empirical curvature parameter, and I2 is the useful energy absorbed by photosystem II. In turn, I2 is calculated by

I2 = alpha_j * Qin,

where alpha_j is the apparent quantum efficiency of electron transport. alpha_j is often defined as

alpha_j = absorptance * phi_psii,max * beta_psii,

where absorptance is the leaf absorptance, phi_psii,max is the maximum quantum yield of photosytem II, and beta_psii is the fraction of light energy partitioned to photosystem II.

Equation 1 can be understood as a "smooth minimum" of two potential rates of electron transport: I2 (which increases linearly with Qin) and Jmax (which is independent of Qin). For lower light levels, I2 is the smaller rate, and J is approximately equal to I2; for very high light levels, Jmax is the smaller rate, and J is approximately equal to Jmax. For intermediate values of Qin, J smoothly transitions from I2 to Jmax.

This equation is often solved for Jmax, and thus it is necessary to consider the conditions for which the solution is appropriate. One key property of Equation 1 is that the largest possible value of J at a given Qin is I2, which only occurs when Jmax is much larger than I2. In other words, when considered as a function of Jmax, the range of the function in Equation 1 is 0 <= J <= I2.

Equation 1 can be solved for Jmax, enabling calculations of Jmax from estimates of J:

Jmax = J * (I2 - theta_j * J) / (I2 - J) (Eq. 2)

Because the range of the function in Equation 1 is 0 <= J <= I2, the domain of its inverse function (defined in Equation 2) is also 0 <= J <= I2. In other words, Jmax can only be calculated using Equation 2 when J < I2. Otherwise, there is no value of Jmax that can reproduce the value of J for the given value of alpha_j. This restriction can also be derived more rigorously; see the **Detailed algebra** section below for more information.

If J >= I2, the calculate_jmax function will return NA for the value of Jmax. This behavior can be bypassed by setting the optional input argument ignore_restriction to TRUE, but this is not recommended outside of pedagogical purposes. See Example 2 below for a demonstration of what goes wrong when Equation 2 is used for J >= I2.

Note that this issue is more significant at lower light levels. For example, assuming a typical value of alpha_j (0.293), I2 for Qin = 1800 micromol / m^2 / s would be 527.4 micromol / m^2 / s. Values of J are typically smaller than this, so an estimate of Jmax can almost always be made. But if a curve were measured at Qin = 300, I2 would only be 87.9 micromol / m^2 / s, placing a stronger restriction on the values of J where Jmax can be estimated. Say the best-fit value of J was 88.9 micromol / m^2 / s for a curve measured with Qin = 300 micromol / m^2 / s; in this case, it would not be possible to estimate Jmax, potentially indicating that the assumed value of alpha_j was not correct.

Typical values:

According to von Caemmerer (2000), typical values of absorptance, phi_psii,max, and beta_psii are 0.85, 1 - 0.15, and 0.5, respectively, leading to alpha_j = 0.36125, and the curvature parameter theta_j is typically 0.7.

Bernacchi et al. (2003) reports that phi_psii,max is 0.6895 for light-adapted leaves at 25 degrees C, while theta_j at 25 degrees C is 0.97875. Using this value of of phi_psii,max with typical values of absorptance and beta_psii results in an alpha_j estimate of 0.2930375.

It is not clear whether the temperture response defined in Bernacchi et al. (2003) is applicable to C4 leaves. For C4 leaves, it may be better to use the temperature-independent estimates from von Caemmerer (2000).

PhotoGEA provides two Jmax parameter lists that can be passed to `calculate_temperature_response`: `jmax_temperature_param_bernacchi` (implements the Bernacchi et al. 2003 values) and `jmax_temperature_param_flat` (implements the von Caemmerer 2000 values). Each of these parameter lists will calculate values of `alpha_j_at_25`, `alpha_j_norm`, `theta_j_at_25`, and `theta_j_norm`.

Absorbed light basis:

Values of `Jmax` can also be estimated from the absorbed photosynthetically active photon flux density (`Qabs`). In that case, we can regroup the terms in the definition of `I2` as follows:

`I2 = (Qin * absorptance) * (phi_psii,max * beta_psii) = Qabs * alpha_j_abs`,

where `alpha_j_abs` is given by `phi_psii,max * beta_psii`. When working in this basis, the default value of `alpha_j` at 25 degrees C should be divided by the assumed absorptance (0.85). For example, the default value of `alpha_j_at_25` used with the Bernacchi et al. (2003) parameters is 0.2930375, so dividing this by 0.95 would yielding an `alpha_j_abs` value of about 0.345. This value could be passed directly to `calculate_jmax` via the `alpha_j_at_25` input argument, overriding the default value. Along with this change, it would also be necessary to change the name of the light column, likely to `Qabs_avg`.

Why PhotoGEA Uses a Separate Function for Jmax:

In principle, values of `Jmax` could be estimated by the fitting functions that estimate `J`: `fit_c3_aci`, `fit_c3_variable_j`, and `fit_c4_aci`. Instead, PhotoGEA requires users to use a separate function (`calculate_jmax`) to estimate `Jmax`. This serves several purposes:

- It highlights that estimates of `Jmax` are made using the same equations for C3 and C4 leaves.
- It leaves open the possibility of other estimates of `Jmax`, such as those based on a rectangular hyperbola instead of the non-rectangular hyperbola used here.
- It emphasizes that sometimes it is not possible to provide an estimate for `Jmax`, depending on the values of `Qin`, `alpha_j`, and `J`, because of the requirement that `J < I2 = alpha_j * Qin`.

The last point is especially important. If `Jmax` were varied during the fitting process, and `J` was estimated from `Jmax` using Equation 1, there would be a restriction on the possible values of `J` that could be obtained: `J < alpha_j * Qin`. This could potentially bias the fitting results, since it may be the case that the best fit would be found for `J` outside this range.

In other words, keeping estimates of `Jmax` separate from the fitting process ensures that the values of `alpha_j` and `theta_j` have no influence on the fits or best-fit values of `J`. This is important since the true values of these parameters for a particular leaf are difficult or impossible to determine.

Detailed algebra:

Here we will solve Equation 1 for `Jmax`, arriving at Equation 2. This algebra is reproduced here to highlight the important restriction that `J < I2`.

First, multiply both sides of Equation 1 by `2 * theta_j`:

`2 * theta_j * J = I2 + Jmax - sqrt[(I2 + Jmax)^2 - 4 * theta_j * I2 * Jmax]`. (Eq. 3)

Next, isolate the square root term on one side:

`I2 + Jmax - 2 * theta_j * J = sqrt[(I2 + Jmax)^2 - 4 * theta_j * I2 * Jmax]`. (Eq. 4)

A key point here is that the right hand side cannot be negative, since the square root of a real number is never negative. Thus, the left hand side also cannot be negative. In other words,

`I2 + Jmax - 2 * theta_j * J >= 0`. (Eq. 5)

We will return to this restriction later. For now, we square both sides of Equation 4:

`(I2 + Jmax)^2 - 4 * theta_j * J * (I2 + Jmax) + 4 * theta_j^2 * J^2 = (I2 + Jmax)^2 - 4 * theta_j * I2 * Jmax`. (Eq. 6)

The term `(I2 + Jmax)^2` appears on both sides of Equation 6 and can therefore be cancelled out. Grouping the remaining terms that contain `Jmax` on one side, we have:

`4 * theta_j * Jmax * (I2 - J) = 4 * theta_j * J * (I2 - theta_j * J)` (Eq. 7)

Finally, provided that `I2 - J` is not zero (in other words, that `I2` is not equal to `J`), we can divide both sides of Equation 7 by `4 * theta_j * (I2 - J)` to obtain Equation 2 above.

Now, we can use this expression (Equation 2) to replace `Jmax` in Equation 5:

`I2 + J * (I2 - theta_j * J) / (I2 - J) - 2 * theta_j * J >= 0`. (Eq. 8)

This can be converted to a single ratio as follows:

`[(I2 - 2 * theta_j * J) * (I2 - J) + J * (I2 - theta_j * J)] / (I2 - J) >= 0`. (Eq. 9)

Multiplying out the factors in the numerator and collecting like terms, Equation 9 becomes

`[I2^2 - 2 * theta_j * I2 * J + theta_j * J^2] / (I2 - J) >= 0`. (Eq. 10)

Because `theta_j` must lie between 0 and 1, `theta_j^2` is always less than or equal to `theta_j`. This allows us to place a lower bound on the value of the numerator of the left hand side of Equation 10:

`I2^2 - 2 * theta_j * I2 * J + theta_j * J^2 >= I2^2 - 2 * theta_j * I2 * J + theta_j^2 * J^2`. (Eq. 11)

The right hand side of Equation 11 can be refactored:

`I2^2 - 2 * theta_j * I2 * J + theta_j * J^2 >= (I2 - theta_j J)^2`. (Eq. 12)

The right hand side of Equation 12 can never be negative, so from this we can also conclude that the numerator of the left hand side of Equation 10 can also never be negative. Thus, the inequality in Equation 10 is satisfied whenever its denominator is positive. In other words, whenever `I2 - J > 0`, or, equivalently, `J < I2`.

Thus, we have shown that Equation 2 holds whenever `J < I2`, since, when this inequality is satisfied, Equation 5 is also satisfied.

Although we do not do so here, it can be shown that when `I2 < J`, the value of `Jmax` that would be calculated by Equation 2 is the inverse of

`J = (I2 + Jmax + sqrt[(I2 + Jmax)^2 - 4 * theta_j * I2 * Jmax]) / (2 * theta_j)` (Eq. 13)

rather than the inverse of Equation 1. Note the difference: in Equation 13, the square root term is added to `I2 + Jmax` rather than subtracted. This is a "smooth maximum" function, rather than a smooth minimum. In fact, whenever `I2 > Jmax`, Equation 13 would predict `J > Jmax`, clearly a nonsensical result. Likewise, the inverse of the function in Equation 13 would predict some values of `Jmax` that are smaller than `J`. Example 2 below shows that it can even return negative values of `Jmax`, which is clearly not reasonable from a biological perspective.

References:

- von Caemmerer, S. "Biochemical Models of Leaf Photosynthesis" (CSIRO Publishing, 2000) [doi:10.1071/9780643103405].
- Walker, A. P. et al. "Multi-hypothesis comparison of Farquhar and Collatz photosynthesis models reveals the unexpected influence of empirical assumptions at leaf and global scales." Global Change Biology 27, 804–822 (2021) [doi:10.1111/gcb.15366].
- Bernacchi, C. J., Pimentel, C. & Long, S. P. "In vivo temperature response functions of parameters required to model RuBP-limited photosynthesis" Plant, Cell & Environment 26, 1419–1430 (2003) [doi:10.1046/j.00168025.2003.01050.x].

Value

The return value is a table based on `data_table` that includes several new columns: `I2_at_25`, `Jmax_at_25`, `Jmax_at_25_msg`, `I2_tl`, `Jmax_tl`, and `Jmax_tl_msg`. The `_msg` columns indicate when the error condition `J >= I2` has occurred.

If `J` confidence intervals were provided in the inputs, then there will be correspoding columns for the related `Jmax`, and `msg` values; for example, `Jmax_at_25_lower` and `Jmax_at_25_lower_msg`.

Examples

```
## Example 1: Estimating Jmax after fitting several C3 A-Ci curves

# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data; we will need average values of leaf temperature and
# incident PPFD in order to calculate Jmax later
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp',
    columns_to_average = c('TleafCnd', 'Qin')
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)

# Fit all curves in the data set (it is more common to do this)
```

```
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer
))

# Calculate temperature-dependent values of Jmax-related parameters
aci_results$parameters <- calculate_temperature_response(
    aci_results$parameters,
    jmax_temperature_param_bernacchi,
    'TleafCnd_avg'
)

# Calculate Jmax
aci_results$parameters <- calculate_jmax(aci_results$parameters)

# Print a few columns
col_to_view <- c('species_plot', 'J_at_25', 'J_tl_avg', 'Jmax_at_25', 'Jmax_tl')

print(aci_results$parameters[, col_to_view, TRUE])

## Example 2: Illustrating the importance of requiring I2 > J

# Define a data frame with input values
npts <- 200
J_seq <- seq_len(npts)

jmax_df <- data.frame(
  J_at_25 = J_seq,
  J_tl_avg = J_seq,
  alpha_j_norm = 1,
  Qin_avg = 300,
  theta_j_norm = 1,
  TleafCnd_avg = 25
)

# Calculate Jmax values, overriding the default behavior so that values of Jmax
# are returned even when I2 < J.
jmax_df <- calculate_jmax(
  jmax_df, alpha_j_at_25 = 0.293, theta_j_at_25 = 0.979,
  ignore_restriction = TRUE
)

# Plot the Jmax values, distinguishing between cases where J < I2 and where
# J > I2. Here we can see that when J > I2, values of Jmax are smaller than J,
# and can even be negative, which is clearly unreasonable from a biological
# perspective. To highlight these considerations, J = I2 is plotted as a dashed
# black line, Jmax = J is plotted as a black long-dashed line, and Jmax = 0 is
# plotted as a solid black line.
ymin <- -50
ymax <- 250
```

```
xmin <- min(J_seq)
xmax <- max(J_seq)

I2 <- jmax_df$I2_at_25[1]

jmax_df$Jmax_at_25_msg[jmax_df$Jmax_at_25_msg == ''] <- 'J < I2'

lattice::xyplot(
  Jmax_at_25 ~ J_at_25,
  group = Jmax_at_25_msg,
  data = jmax_df,
  auto = TRUE,
  type = 'l',
  xlim = c(xmin, xmax),
  ylim = c(ymin, ymax),
  xlab = 'J (micromol / m^2 / s)',
  ylab = 'Jmax (micromol / m^2 / s)',
  panel = function(x, y, ...) {
    lattice::panel.lines(c(0, 0) ~ c(xmin, xmax), lty = 1, col = 'black')
    lattice::panel.lines(c(ymin, ymax) ~ c(I2, I2), lty = 2, col = 'black')
    lattice::panel.lines(J_seq ~ J_seq, lty = 5, col = 'black')
    lattice::panel.xyplot(x, y, ...)
  }
)
```

`calculate_leakiness_ubierna` *Calculate leakiness*

Description

Calculates leakiness (`phi`) from combined gas exchange and isotope discrimination measurements as described in Ubierna et al. (2013). This function can accomodate alternative colum names for the variables taken from `exdf_obj`; it also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_leakiness_ubierna(
  exdf_obj,
  e = -3,
  a_bar_column_name = 'a_bar',
  a_column_name = 'A',
  ci_column_name = 'Ci',
  co2_s_column_name = 'CO2_s',
  csurface_column_name = 'Csurface',
  delta_c13_r_column_name = 'delta_C13_r',
  delta_obs_tdl_column_name = 'Delta_obs_tdl',
```

```
    rl_column_name = 'RL',
    t_column_name = 't'
  )
```

Arguments

exdf_obj	An exdf object.
e	The isotopic fractionation during day respiration in ppt.
a_bar_column_name	The name of the column in exdf_obj that contains the weighted isotopic fractionation across the boundary layer and stomata in ppt. Values of a_bar are typically calculated using calculate_ternary_correction.
a_column_name	The name of the column in exdf_obj that contains the net CO2 assimilation rate in micromol m^(-2) s^(-1).
ci_column_name	The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).
co2_s_column_name	The name of the column in exdf_obj that contains the CO2 concentration in the sample line (outgoing air) in micromol mol^(-1).
csurface_column_name	The name of the column in exdf_obj that contains the CO2 concentration at the leaf surface in micromol mol^(-1). Values of Csurface are typically calculated using calculate_gas_properties.
delta_c13_r_column_name	The name of the column in exdf_obj that contains the CO2 isotope ratio in the reference line (incoming air) in ppt.
delta_obs_tdl_column_name	The name of the column in exdf_obj that contains the observed isotope discrimination values in ppt.
rl_column_name	The name of the column in exdf_obj that contains the rate of day respiration in micromol m^(-2) s^(-1).
t_column_name	The name of the column in exdf_obj that contains the ternary correction factor (dimensionless). Values of t are typically calculated using calculate_ternary_correction

Details

This function uses the model for photosynthetic discrimination against 13C in C4 plants to determine leakiness values, as described in Ubierna et al. (2013). In particular, the following equations from that source are implemented in the code:

- Isotopic fractionation due to day respiration (e_prime) is calculated using Equation 21.
- Leakiness including respiratory and photorespiratory fractionations under high light (phi_i) is calculated using Equation 16.
- Leakiness including respiratory and photorespiratory fractionations and Cs under high light (phi_is) is calculated using Equation 15.

- Leakiness ignoring respiratory and photorespiratory fractionations and Cs (`phi_sim`) is calculated using Equation 17.

References:

Ubierna, N., Sun, W., Kramer, D. M. and Cousins, A. B. "The efficiency of C4 photosynthesis under low light conditions in Zea mays, Miscanthus x giganteus and Flaveria bidentis." Plant, Cell & Environment 36, 365–381 (2013) [doi:10.1111/j.13653040.2012.02579.x].

Value

An `exdf` object based on `exdf_obj` that includes the following additional columns, calculated as described above: `e_prime`, `phi_i`, `phi_is`, and `phi_sim`. The category for each of these new columns is `calculate_leakiness_ubierna` to indicate that they were created using this function.

Examples

```
## In this example we load gas exchange and TDL data files, calibrate the TDL
## data, pair the data tables together, and then calculate leakiness. The
## results from this example are not meaningful because these measurements
## were not collected from C4 plants.

# Read the TDL data file, making sure to interpret the time zone as US Central
# time
tdl_data <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_for_gm.dat'),
  'TIMESTAMP',
  list(tz = 'America/Chicago')
)

# Identify cycles within the TDL data
tdl_data <- identify_tdl_cycles(
  tdl_data,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Use reference tanks to calibrate the TDL data
processed_tdl <- consolidate(by(
  tdl_data,
  tdl_data[, 'cycle_num'],
  process_tdl_cycle_erml,
  noaa_valve = 2,
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
  calibration_3_valve = 26,
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
```

```
))

# Read the gas exchange data, making sure to interpret the time stamp in the US
# Central time zone
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time',
  list(tz = 'America/Chicago')
)

# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# Pair the Licor and TDL data by locating the TDL cycle corresponding to each
# Licor measurement
licor_data <- pair_gasex_and_tdl(licor_data, processed_tdl$tdl_data)

# Calculate total pressure (needed for calculate_gas_properties)
licor_data <- calculate_total_pressure(licor_data)

# Calculate Csurface (needed for calculate_ternary_correction)
licor_data <- calculate_gas_properties(licor_data)

# Calculate ternary correction
licor_data <- calculate_ternary_correction(licor_data)

# Calculate isotope discrimination (needed for calculate_leakiness_ubierna)
licor_data <- calculate_isotope_discrimination(licor_data)

# Set respiration (needed for calculate_leakiness_ubierna)
licor_data <- set_variable(
  licor_data,
  'RL',
  'micromol m^(-2) s^(-1)',
  value = 1.2
)

# Calculate leakiness
licor_data <- calculate_leakiness_ubierna(licor_data)

# View some of the results
licor_data[, c('replicate', 'CO2_s', 'Delta_obs_tdl', 'phi_i', 'phi_sim')]
```

---

calculate_temperature_response

Calculate temperature-dependent parameter values

Description

Calculate leaf-temperature-dependent values of various parameters using various temperature response functions.

Usage

```
calculate_temperature_response(
  exdf_obj,
  temperature_response_parameters,
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

`exdf_obj`	An `exdf` object representing data from a Licor gas exchange measurement system.
`temperature_response_parameters`	A list, where each element describes the temperature response of a parameter value. The name of each element must be the name of the parameter. Each element must be a list itself, whose named elements must include the type of temperature response function to use (`type`), thee units of the parameter (`units`), and the values of necessary temperature response parameters. See below for more details.
`tleaf_column_name`	The name of the column in `exdf_obj` that contains the leaf temperature in units of `degrees C`.

Details

Some key photosynthetic parameters are known to vary with temperature according to well-established temperature response functions such as the Arrhenius equation. The `calculate_temperature_response` function can be used to calculate such temperature-dependent parameter values at leaf temperature.

Depending on the `type` value supplied in each element of `temperature_response_parameters`, one of several possible functions will be used to calculate the temperature response:

- When `type` is `'Arrhenius'`, the `calculate_temperature_response_arrhenius` function will be used.
- When `type` is `'Gaussian'`, the `calculate_temperature_response_gaussian` function will be used.
- When `type` is `'Johnson'`, the `calculate_temperature_response_johnson` function will be used.
- When `type` is `'Polynomial'`, the `calculate_temperature_response_polynomial` function will be used.

Values of `type` are not case-sensitive.

It is rare to directly specify these parameters; instead, it is more typical to use one of the pre-set values such as those included in `c3_temperature_param_sharkey`.

Value

An exdf object based on exdf_obj that includes one new column for each element of temperature_response_parameters, where the temperature-dependent values of these new columns are determined using the temperature values specified by the tleaf_column_name column. The category of each of these new columns is calculate_temperature_response to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# In this example we will calculate temperature-dependent values of two
# parameters:
#
# - The `Kc` parameter (in units of `micromol mol^(-1)`) will be calculated
#   using an Arrhenius function with scaling constant `c` = 38.05 and activation
#   energy `Ea` = 79.43 kJ / mol.
#
# - The `Jmax` parameter (in units of `micromol m^(-2) s^(-1)) will be
#   using a Gaussian function with optimal temperature `t_opt` = 43 degrees C
#   and width `sigma` = 16 degrees C.
#
# So the `temperature_response_parameters` list will contain two elements,
# defined as follows:

trp <- list(
  Kc = list(
    type = 'Arrhenius',
    c = 38.05,
    Ea = 79.43,
    units = 'micromol mol^(-1)'
  ),
  Jmax = list(
    type = 'Gaussian',
    optimum_rate = 4,
    t_opt = 43,
    sigma = 16,
    units = 'micromol m^(-2) s^(-1)'
  )
)

# Now we can calculate the values of Kc and Jmax at the measured leaf
# temperatures recorded in the log file
licor_file <- calculate_temperature_response(licor_file, trp)

licor_file$units$Kc      # View the units of the new `Kc` column
licor_file$categories$Kc # View the category of the new `Kc` column
licor_file[,'Kc']        # View the values of the new `Kc` column

licor_file$units$Jmax      # View the units of the new `Jmax` column
```

```
licor_file$categories$Jmax # View the category of the new `Jmax` column
licor_file[,'Jmax']        # View the values of the new `Jmax` column
```

`calculate_temperature_response_arrhenius` *Calculate temperature-dependent values using Arrhenius equations*

Description

Calculate leaf-temperature-dependent values of various parameters using Arrhenius equations. It is rare for users to call this function directly; instead, it is used internally by `calculate_temperature_response`.

Usage

```
calculate_temperature_response_arrhenius(
  exdf_obj,
  arrhenius_parameters,
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

`exdf_obj`	An `exdf` object representing data from a Licor gas exchange measurement system.
`arrhenius_parameters`	A list of named lists. Each list element should describe the Arrhenius scaling factor (`c`), activation energy in `kJ / mol` (`Ea`), and units (`units`) for a variable that follows an Arrhenius temperature dependence. The name of each list element should be the corresponding name of the variable.
`tleaf_column_name`	The name of the column in `exdf_obj` that contains the leaf temperature in units of `degrees C`.

Details

The Arrhenius equation is often used to calculate the temperature dependence of the rate of a chemical reaction. It is often stated as follows:

(1) `rate = A * exp(-Ea / (R * T))`

where `A` is the "pre-exponential factor" that sets the overall scaling, `Ea` is the activation energy, `R` is the ideal gas constant, and `T` is the temperature in Kelvin. See, for example, the Wikipedia page for the equation.

In photosynthesis research, it is common to use an alternative form of the equation, where the pre-exponential factor `A` is rewritten as an exponent `A = exp(c)`, where `c` is a "scaling factor" whose value can be calculated from `A` according to `c = ln(A)`. In this formulation, the equation becomes:

(2) `rate = exp(c) * exp(-Ea / (R * T)) = exp(c - Ea / (R * T))`

The advantage of this version is that the natural logarithm of the rate is equal to `c - Ea / (R * T)`. This means that the Arrhenius paramerer values can be easily determined from a linear fit of `log(rate)` against `1 / (R * T)`; `c` is the y-intercept and `-Ea` is the slope.

In `calculate_temperature_response_arrhenius`, the scaling factor (`c`), activation energy (`Ea`), and units (`units`) for a variable must be specified as elements of a list, which itself is a named element of `arrhenius_parameters`. For example, if a variable called `Kc` has `c = 38.05`, `Ea = 79.43`, and units of `micromol mol^(-1)`, the `arrhenius_parameters` argument could be specified as follows: `list(Kc = list(c = 38.05, Ea = 79.43, units = 'micromol mol^(-1)'))`.

It is rare to directly specify the Arrhenius parameters; instead, it is more typical to use one of the pre-set values such as those included in `c3_temperature_param_sharkey`.

Sometimes a publication will specify the value of a variable at 25 degrees C instead of the Arrhenius scaling factor `c`. In this case, there is a "trick" for determining the value of `c`. For example, if the Arrhenius exponent should be `X` at 25 degrees C, then we have the following: `X = exp(c - Ea / (R * (25 + 273.15)))`, which we can solve algebraically for `c` as follows: `c = ln(X) + Ea / f`, where `f = R * (25 + 273.15)`. As a special case, for parameters normalized to 1 at 25 degrees C, we have `c = Ea / f`. The value of `f` can be accessed as `PhotoGEA:::f`.

Another common scenario is that we may wish to convert the units of a variable defined by Arrhenius exponents. For example, let's say `Y` is determined by an Arrhenius exponent, i.e., that `Y = exp(c - Ea / (R * T))`, and we want to convert `Y` to different units via a multiplicative conversion factor `cf`. Then, in the new units, `Y` becomes `Y_new = cf * Y = cf * exp(c - (R * T))`. Through algebra, it is possible to combine `cf` with the original value of `c` as `c_new = c + ln(cf)`. Then we can continue calculating `Y_new` using an Arrhenius factor as `Y_new = exp(c_new - Ea / (R * T))`.

Value

An `exdf` object based on `exdf_obj` that includes one new column for each element of `arrhenius_parameters`, where the temperature-dependent values of these new columns are determined using the temperature values specified by the `tleaf_column_name` column. The category of each of these new columns is `calculate_temperature_response_arrhenius` to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_temperature_response_arrhenius(
  licor_file,
  list(Kc_norm = c3_temperature_param_sharkey$Kc_norm)
)

licor_file$units$Kc_norm      # View the units of the new `Kc_norm` column
licor_file$categories$Kc_norm # View the category of the new `Kc_norm` column
licor_file[,'Kc_norm']        # View the values of the new `Kc_norm` column
```

`calculate_temperature_response_gaussian` *Calculate temperature-dependent values using Gaussian equations*

Description

Calculate leaf-temperature-dependent values of various parameters using Gaussian equations. It is rare for users to call this function directly; instead, it is used internally by `calculate_temperature_response`.

Usage

```
calculate_temperature_response_gaussian(
  exdf_obj,
  gaussian_parameters,
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

`exdf_obj`	An `exdf` object representing data from a Licor gas exchange measurement system.
`gaussian_parameters`	A list of named lists. Each list element should describe the optimal temperature in degrees `C` (`t_opt`), the "width" in degrees `C` (`sigma`), and the units (`units`) for a variable that follows a peaked Gaussian temperature dependence. The name of each list element should be the corresponding name of the variable.
`tleaf_column_name`	The name of the column in `exdf_obj` that contains the leaf temperature in units of degrees `C`.

Details

A Gaussian equation is sometimes used to model the temperature dependence of a biochemical rate parameter. Typically this is expressed by

`rate = optimal_rate * exp(-(T - T_opt)^2 / sigma^2)`

where `optimal_rate` is the highest rate which occurs at the optimal temperature `T_opt`, `T` is the current temperature, and `sigma` represents the "width" of the peak. More technically, it can be described as the difference in temperature away from the optimal value at which the rate falls to 37 percent (`1/e`) of its maximum.

In `calculate_temperature_response_gaussian`, the optimal rate (`optimal_rate`), optimal temperature (`t_opt`), width (`sigma`), and units (`units`) for a variable must be specified as elements of a list, which itself is a named element of `gaussian_parameters`. For example, if a variable called `Jmax` has `optimal_rate = 1`, `t_opt = 43`, `sigma = 26`, and units of `micromol mol^(-1)`, the `gaussian_parameters` argument could be specified as follows: `list(Jmax = list(optimal_rate = 1, t_opt = 43, sigma = 26, units = 'micromol mol^(-1)'))`.

It is rare to specify these parameters directly; instead, it is more typical to use one of the pre-set values such as those included in `c4_temperature_param_vc`.

Value

An exdf object based on exdf_obj that includes one new column for each element of gaussian_parameters, where the temperature-dependent values of these new columns are determined using the temperature values specified by the tleaf_column_name column. The category of each of these new columns is calculate_temperature_response_gaussian to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_temperature_response_gaussian(
  licor_file,
  list(J_norm = c4_temperature_param_vc$J_norm)
)

licor_file$units$J_norm      # View the units of the new `J_norm` column
licor_file$categories$J_norm # View the category of the new `J_norm` column
licor_file[,'J_norm']        # View the values of the new `J_norm` column
```

---

calculate_temperature_response_johnson

Calculate temperature-dependent values using Johnson-Eyring-Williams equations

---

Description

Calculate leaf-temperature-dependent values of various parameters using Johnson-Eyring-Williams equations. It is rare for users to call this function directly; instead, it is used internally by calculate_temperature_response

Usage

```
calculate_temperature_response_johnson(
  exdf_obj,
  johnson_parameters,
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

exdf_obj	An exdf object representing data from a Licor gas exchange measurement system.

johnson_parameters
: A list of named lists. Each list element should describe the scaling factor (c), enthalpy of activation in kJ / mol (Ha), enthalpy of deactivation in kJ / mol (Hd), entropy in kJ / K / mol (S), and units (units) for a variable that follows a Johnson-Eyring-Williams temperature dependence. The name of each list element should be the corresponding name of the variable.

tleaf_column_name
: The name of the column in exdf_obj that contains the leaf temperature in units of degrees C.

Details

The Johnson-Eyring-Williams equation is often used to calculate the temperature dependence of the rate of a chemical reaction. It can be stated as follows:

```
rate = exp(c - Ha / (R * T)) / (1 + exp(S / R - Hd / (R * T)))
```

where c is the scaling factor that sets the overall magnitude of the rate, Ha is the enthalpy of activation, Hd is the enthalpy of deactivation, S is the entropy, R is the ideal gas constant, and T is the temperature in Kelvin.

This equation exhibits a peak; in other words, there is a particular temperature (the optimal temperature) where the rate is maximized. Thus, it is often used in place of an Arrhenius equation (see calculate_temperature_response_arrhenius) for photosynthetic parameters that exhibit a decrease at high temperatures.

This equation was originally published in Johnson, Eyring, & Williams (1942) and has been used to model the temperature dependence of key photosynthetic parameters, as in Harley et al. (1992), Bernacchi et al. (2003), Sharkey et al. (2007), and others.

In calculate_temperature_response_johnson, the scaling factor (c), enthalpy of activation (Ha), enthalpy of deactivation (Hd), entopy (S), and units (units) for a variable must be specified as elements of a list, which itself is a named element of johnson_parameters. For example, if a variable called Tp has c = 21.46, Ha = 53.1, Hd = 201.8, S = 0.65, and units of micromol mol^(-1), the johnson_parameters argument could be specified as follows: list(Tp = list(c = 21.46, Ha = 53.1, Hd = 201.8, S = 0.65, units = 'micromol mol^(-1)')).

It is rare to directly specify these parameters; instead, it is more typical to use one of the pre-set values such as those included in c3_temperature_param_sharkey.

References:

- Johnson, F. H., Eyring, H. & Williams, R. W. "The nature of enzyme inhibitions in bacterial luminescence: Sulfanilamide, urethane, temperature and pressure." Journal of Cellular and Comparative Physiology 20, 247–268 (1942) [doi:10.1002/jcp.1030200302].
- Harley, P. C., Thomas, R. B., Reynolds, J. F. & Strain, B. R. "Modelling photosynthesis of cotton grown in elevated CO2." Plant, Cell & Environment 15, 271–282 (1992) [doi:10.1111/j.13653040.1992.tb00974.x].
- Bernacchi, C. J., Pimentel, C. & Long, S. P. "In vivo temperature response functions of parameters required to model RuBP-limited photosynthesis." Plant, Cell & Environment 26, 1419–1430 (2003) [doi:10.1046/j.00168025.2003.01050.x].
- Sharkey, T. D., Bernacchi, C. J., Farquhar, G. D. & Singsaas, E. L. "Fitting photosynthetic carbon dioxide response curves for C3 leaves." Plant, Cell & Environment 30, 1035–1040 (2007) [doi:10.1111/j.13653040.2007.01710.x].

Value

An exdf object based on exdf_obj that includes one new column for each element of johnson_parameters, where the temperature-dependent values of these new columns are determined using the temperature values specified by the tleaf_column_name column. The category of each of these new columns is calculate_temperature_response_johnson to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_temperature_response_johnson(
  licor_file,
  list(Tp_norm = c3_temperature_param_sharkey$Tp_norm)
)

licor_file$units$Tp_norm      # View the units of the new `Tp_norm` column
licor_file$categories$Tp_norm # View the category of the new `Tp_norm` column
licor_file[,'Tp_norm']        # View the values of the new `Tp_norm` column
```

---

calculate_temperature_response_polynomial

Calculate temperature-dependent values using polynomial equations

---

Description

Calculate leaf-temperature-dependent values of various parameters using polynomial equations. It is rare for users to call this function directly; instead, it is used internally by calculate_temperature_response.

Usage

```
calculate_temperature_response_polynomial(
  exdf_obj,
  polynomial_parameters,
  tleaf_column_name = 'TleafCnd'
)
```

Arguments

exdf_obj	An exdf object representing data from a Licor gas exchange measurement system.
polynomial_parameters	A list of named lists. Each list element should describe the polynomial coefficients (coef) and units (units) for a variable that follows a polynomial temperature dependence. The name of each list element should be the corresponding name of the variable.

`tleaf_column_name`
: The name of the column in `exdf_obj` that contains the leaf temperature in units of degrees `C`.

Details

Polynomial equations are often used to calculate the temperature dependence of the rates of chemical reactions. For example, a second-order polynomial could be given as follows:

(1) `rate = R_0 + R_1 * T + R_2 * T^2`

where `R_0`, `R_1`, and `R_2` are the zeroth, first, and second order coefficients and `T` is the temperature. Higher order polynomials can also be defined, where an order-`N` polynomial is given by

(2) `rate = R_0 + R_1 * T + R_2 * T^2 + ... + R_N * T^N`

In general, an order-`N` polynomial has `N` coefficients, although some of them may be zero.

In `calculate_temperature_response_polynomial`, the coefficients (`coef`) and units (`units`) for a variable must be specified as elements of a list, which itself is a named element of `polynomial_parameters`. The coefficients must be specified as a numeric vector, where the `i`th element represents the `i`th coefficient. For example, if a dimensionless variable called `theta` is calculated according to `theta = 0.352 + 0.022 * T - 3.4e-4 * T^2`, the `polynomial_parameters` argument could be supplied as follows: `list(theta = list(coef = c(0.352, 0.022, -3.4e-4), units = 'dimensionless'))`.

It is rare to directly specify the polynomial parameters; instead, it is more typical to use one of the pre-set values such as those included in `jmax_temperature_param_bernacchi`.

Value

An `exdf` object based on `exdf_obj` that includes one new column for each element of `polynomial_parameters`, where the temperature-dependent values of these new columns are determined using the temperature values specified by the `tleaf_column_name` column. The category of each of these new columns is `calculate_temperature_response_polynomial` to indicate that they were created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_temperature_response_polynomial(
  licor_file,
  list(theta = list(coef = c(0.352, 0.022, -3.4e-4), units = 'dimensionless'))
)

licor_file$units$theta      # View the units of the new `theta` column
licor_file$categories$theta # View the category of the new `theta` column
licor_file[,'theta']        # View the values of the new `theta` column
```

calculate_ternary_correction
Calculate ternary correction factor

Description

Calculates the ternary correction factor `t` that is used in many carbon isotope discrimination calculations.

Usage

```
calculate_ternary_correction(
  exdf_obj,
  ci_column_name = 'Ci',
  co2_s_column_name = 'CO2_s',
  csurface_column_name = 'Csurface',
  e_column_name = 'E',
  gtc_column_name = 'gtc'
)
```

Arguments

`exdf_obj`	An `exdf` object containing photosynthetic gas exchange data.
`ci_column_name`	The name of the column in `exdf_obj` that contains the intercellular CO2 concentration in `micromol mol^(-1)`.
`co2_s_column_name`	The name of the column in `exdf_obj` that contains the sample line (incoming air) CO2 concentration in `micromol mol^(-1)`.
`csurface_column_name`	The name of the column in `exdf_obj` that contains the CO2 concentration at the leaf surface in `micromol mol^(-1)`. This is typically calculated using `calculate_gas_properties`.
`e_column_name`	The name of the column in `exdf_obj` that contains the leaf transpiration rate in `mol m^(-2) s^(-1)`.
`gtc_column_name`	The name of the column in `exdf_obj` that contains the total conductance to CO2 diffusion across the boundary layer and stomata in series in `mol m^(-2) s^(-1)`.

Details

During photosynthetic gas exchange, there are separate fluxes of CO2 and H2O flowing in and out of the leaf. These gases interact with each other and with air, forming a ternary mixture. These interactions must be taken into account when modeling carbon isotope discrimination. Typically this is done via `t`, a ternary correction factor first introduced by Farquhar and Cernusak (2012). Here we calculate `t` as described in Equations 9 and 10 from Ubierna et al. (2018):

`t = alpha_ac * E / (2 * g_ac)`

and

a_bar = (a_b * (C_a - C_s) + a_s * (C_s - C_i)) / (C_a - C_i),

where E is the transpiration rate, g_ac is the total conductance to CO2 diffusion across the boundary layer and stomata in series, a_bar is the weighted fractionation across the boundary layer and stomata in series, a_b is the fractionation during diffusion through the boundary layer, a_s is the fractionation during diffusion through the stomata, C_a is the ambient CO2 concentration (in wet air), C_s is the CO2 concentration (in wet air) at the leaf surface, and C_i is the CO2 concentration (in wet air) in the intercellular spaces.

alpha_ac is the overall fractionation during diffusion through air; alpha_ac and a_bar are related according to an un-numbered equation in Ubierna et al. (2018) that appears just after Equation 9:

alpha_ac = 1 + a_bar

References:

Farquhar, G. D. and Cernusak, L. A. "Ternary effects on the gas exchange of isotopologues of carbon dioxide." Plant, Cell & Environment 35, 1221–1231 (2012) [doi:10.1111/j.13653040.2012.02484.x].

Ubierna, N., Holloway-Phillips, M.-M. and Farquhar, G. D. "Using Stable Carbon Isotopes to Study C3 and C4 Photosynthesis: Models and Calculations." in Photosynthesis: Methods and Protocols (ed. Covshoff, S.) 155–196 (Springer, 2018) [doi:10.1007/9781493977864_10].

Value

An exdf object based on exdf_obj that includes values of t, a_bar, and alpha_ac calculated as described above. The category of each new column is calculate_ternary_correction to indicate that it was created using this function.

Examples

```
## In this example we load a gas exchange data file and then calculate the
## ternary correction factor

# Read the gas exchange data
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time'
)

# Calculate total pressure (needed for calculate_gas_properties)
licor_data <- calculate_total_pressure(licor_data)

# Calculate Csurface (needed for calculate_ternary_correction)
licor_data <- calculate_gas_properties(licor_data)

# Calculate ternary correction
licor_data <- calculate_ternary_correction(licor_data)

# View some of the results
licor_data[, c('replicate', 'A', 'E', 'Csurface', 't', 'a_bar', 'alpha_ac')]
```

calculate_total_pressure *Calculate the total pressure in bar*

Description

Calculates the total pressure in bar. Licor gas exchange measurement systems report both the abient air pressure (Pa) and the chamber overpressure (DeltaPcham) in kPa; the total pressure in the chamber is therefore given by the sum of these two columns. This function can accomodate alternative column names for the variables taken from Licor log files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_total_pressure(
  exdf_obj,
  pa_column_name = 'Pa',
  deltapcham_column_name = 'DeltaPcham'
)
```

Arguments

exdf_obj	An exdf object that contains pressure measurements.
pa_column_name	The name of the column in exdf_obj that contains the ambient air pressure in kPa.
deltapcham_column_name	The name of the column in exdf_obj that contains the chamber overpressure in kPa.

Details

If deltapcham_column_name is NA, this function will simply convert the values in the pa_column_name to units of bar. Otherwise, the values from the pa_column_name and deltapcham_column_name columns will be added together and converted to bar.

Value

An exdf object based on exdf_obj that includes the total pressure values in a new column called total_pressure. The category of this new column is calculate_total_pressure to indicate that it was created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package and calculate the
# total pressure.
licor_file <- read_gasex_file(
```

```
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file$units$total_pressure      # View the units of the new `total_pressure` column
licor_file$categories$total_pressure # View the category of the new `total_pressure` column
licor_file[, 'total_pressure']       # View the values of the new `total_pressure` column
```

`calculate_wue` *Calculate intrinsic water use efficiency*

Description

Calculates the intrinsic water use efficiency (`iWUE`). This function can accomodate alternative column names for the variables taken from the data file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
calculate_wue(
  exdf_obj,
  calculate_c3 = FALSE,
  a_column_name = 'A',
  ca_column_name = 'Ca',
  cc_column_name = 'Cc',
  ci_column_name = 'Ci',
  e_column_name = 'E',
  gmc_column_name = 'gmc_tl',
  gsw_column_name = 'gsw',
  h2o_a_column_name = 'H2O_s',
  h2o_i_column_name = 'H2O_i',
  total_pressure_column_name = 'total_pressure'
)
```

Arguments

`exdf_obj`	An `exdf` object.
`calculate_c3`	A logical variable indicating whether to calculate additional variables that can be useful for C3 plants (`g_ratio` and `drawdown_ct`). Note that these quantities require values of mesophyll conductance and `Cc`, so it is not always possible to calculate them.
`a_column_name`	The name of the column in `exdf_obj` that contains the net CO2 assimilation rate in `micromol m^(-2) s^(-1)`.
`ca_column_name`	The name of the column in `exdf_obj` that contains the ambient CO2 concentration in `micromol mol^(-1)`.

cc_column_name The name of the column in exdf_obj that contains the chloroplastic CO2 concentration in micromol mol^(-1). Typically these are calculated using apply_gm.

ci_column_name The name of the column in exdf_obj that contains the intercellular CO2 concentration in micromol mol^(-1).

e_column_name The name of the column in licor_exdf that contains the transpiration rate in mol m^(-2) s^(-1).

gmc_column_name
The name of the column in licor_exdf that contains the mesophyll conductance to CO2 at leaf temperature in mol m^(-2) s^(-1) bar^(-1).

gsw_column_name
The name of the column in licor_exdf that contains the stomatal conductance to water vapor in mol m^(-2) s^(-1).

h2o_a_column_name
The name of the column in exdf_obj that contains the water vapor concentration in the air surrounding the leaf (i.e., the ambient water vapor concentration) in mmol mol^(-1).

h2o_i_column_name
The name of the column in exdf_obj that contains the water vapor concentration in the leaf's intercellular air spaces in mmol mol^(-1). Typically this value is calculated using calculate_gas_properties.

total_pressure_column_name
The name of the column in exdf_obj that contains the total pressure in bar. Typically this value is calculated using calculate_total_pressure.

Details

Leaf-level water use efficiency (lWUE) is defined as the ratio of net CO2 assimilation (An) to transpiration (E):

lWUE = An / E.

This quantity can also be expressed in terms of water and CO2 concentrations:

lWUE = 0.6 * Ca * (1 - Ci / Ca) / (H2Oi - H2Oa).

Here, Ca and Ci are the atmospheric and intercellular CO2 concentrations, and H2Oa and H2Oi are the atmospheric and intercellular water vapor concentrations. If differences in lWUE are measured between different groups of plants, it can be helpful to separately investigate Ci / Ca and H2Oi - H2Oa to see which factor is driving the differences.

The intrinsic water use efficiency iWUE is a measure of leaf-level water use efficiency, and it is defined to be the ratio An and the stomatal conductance to H2O diffusion (gsw):

iWUE = An / gsw.

For C3 plants, iWUE can be reexpressed as

iWUE = (gmc / gsw) / (1 + (gmc / gsw)) * (Ca - Cc),

where gmc is the mesophyll conductance to CO2 diffusion and Cc is the chloroplast CO2 concentration. If differences in iWUE are measured between different groups of plants, it can be helpful to separately investigate gmc / gsw and Ca - Cc to see which factor is driving the differences.

Note: both measures of water use efficiency depend directly or indirectly on stomatal conductance. Stomata are notoriously slow to reach steady-state, but water use efficiency is only reliable at steady-state. For this reason, it is recommended to only analyze water use efficiency for gas exchange measurements where stomatal conductance has stabilized. For an A-Ci or A-Q curve, only the first measured point has typically reached steady-state stomatal conductance. On the other hand, for a Ball-Berry curve, all measured points should have reached steady-state stomatal conductance.

For more details about these quantities, see Leakey et al. "Water Use Efficiency as a Constraint and Target for Improving the Resilience and Productivity of C3 and C4 Crops." Annual Review of Plant Biology 70 (1): 781–808 (2019) [doi:10.1146/annurevarplant042817040305].

In this function, the following variables are calculated:

- `lWUE`, given by `iWUE = An / E`
- `Cia_ratio`, given by `Cia_ratio = Ci / Ca`
- `drawdown_sw`, given by `drawdown_sw = H2Oi - H2Oa` (this is the drawdown of water vapor across the stomata)
- `iWUE`, given by `iWUE = An / gsw`
- `g_ratio`, given by `g_ratio = gmc / gsw`
- `drawdown_ct`, given by `drawdown_ct = Ca - Cc` (this is the total drawdown of CO2 from the ambient air to the chloroplast)

Note: `g_ratio` and `drawdown_ct` are only calculated if `calculate_c3` is `TRUE`.

Value

An `exdf` object based on `exdf_obj` that includes the quantities listed above, along with their units. The category of each of these new columns is `calculate_wue` to indicate that it was created using this function.

Examples

```
# Read an example Licor file included in the PhotoGEA package and calculate the
# water use efficiency.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_sharkey)

licor_file <- calculate_gas_properties(licor_file)

licor_file <- apply_gm(licor_file, gmc_at_25 = 0.5)

licor_file <- calculate_wue(licor_file, calculate_c3 = TRUE)

licor_file$units$iWUE      # View the units of the new `iWUE` column
licor_file$categories$iWUE # View the category of the new `iWUE` column
licor_file[, 'iWUE']       # View the values of the new `iWUE` column
```

cbind.exdf *Combine exdf objects by columns or rows*

Description

Combines one or more exdf objects by the columns or rows of their main_data. For rbind, errors will occur if column names are not the same in all of the exdf objects, and if all units and categories are not identical.

Usage

```
  ## S3 method for class 'exdf'
cbind(..., deparse.level = 1)

  ## S3 method for class 'exdf'
rbind(
    ...,
    deparse.level = 1,
    make.row.names = TRUE,
    stringsAsFactors = FALSE
  )
```

Arguments

...	Two or more exdf objects.
deparse.level	See associated documentation for the generic versions of cbind and rbind.
make.row.names	See associated documentation for the generic version of rbind.
stringsAsFactors	See associated documentation for the generic version of rbind.

Value

Returns a new exdf object.

See Also

exdf

Examples

```
# Make some simple exdf objects. 1 and 2 have the same number of rows but
# different columns, while 1 and 3 have the same columns but different rows.
simple_exdf_1 <- exdf(data.frame(A = 1), data.frame(A = 'au'), data.frame(A = 'ac'))
simple_exdf_2 <- exdf(data.frame(B = 2), data.frame(B = 'bu'), data.frame(B = 'bc'))
simple_exdf_3 <- exdf(data.frame(A = 2), data.frame(A = 'au'), data.frame(A = 'ac'))

cbind(simple_exdf_1) # will just return simple_exdf_1
```

```
cbind(simple_exdf_1, simple_exdf_2)

rbind(simple_exdf_1) # will just return simple_exdf_1
rbind(simple_exdf_1, simple_exdf_3)
```

`check_required_variables` *Make sure required variables exist*

Description

Checks whether the input table has the required variables.

Usage

```
  check_required_variables(x, required_variables, check_NA = TRUE)

  ## S3 method for class 'data.frame'
check_required_variables(x, required_variables, check_NA = TRUE)

  ## S3 method for class 'exdf'
check_required_variables(x, required_variables, check_NA = TRUE)
```

Arguments

`x`	A table-like R object such as a data frame or an `exdf`.
`required_variables`	A set of variables that must each be included in `x` as columns.
`check_NA`	A logical value indicating whether to check for columns that are all `NA`; see below.

Details

`check_required_variables` is generic, with methods defined for data frames and `exdf` objects.

When `x` is an `exdf`, the `required_variables` input argument must be a list of named strings, where the name of each element specifies the name of a column that must be included in `x`, while the value of each column specifies the corresponding units for that column. If the value is `NA`, no unit checking will be performed.

When `x` is a `data.frame`, the `required_variables` input argument can be specified as a list (as if `x` were an `exdf` object) or as a character vector specifying the names of columns that should be included in `x`.

The required variables will be checked as follows:

- If any required variable columns are missing from the table, an informative error message will be thrown.

- If check_NA is TRUE and any required variable columns are entirely NA, an informative error message will be thrown.
- If any required variable colums have incorrect units, an informative error message will be thrown. (Only applies to exdf objects.)

Otherwise, check_required_variables will have no output and produce no messages.

This function is used internally by many other functions from the PhotoGEA package to check for important columns and make sure they have the correct units. For example, see the code for apply_gm by typing PhotoGEA::apply_gm in the R terminal.

Value

The check_required_variables function does not return anything.

See Also

exdf

Examples

```
# Create a simple exdf object
simple_exdf <- exdf(
  data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)),
  data.frame(A = 'm', B = 's', stringsAsFactors = FALSE),
  data.frame(A = 'Cat1', B = 'Cat2', stringsAsFactors = FALSE)
)

# Confirm that columns named `A` and `B` are in the object, and that they have
# units of `m` and `s`, respectively.
check_required_variables(simple_exdf, list(A = 'm', B = 's'))

# Confirm that columns named `A` and `B` are in the object, but only check units
# for the `A` column.
check_required_variables(simple_exdf, list(A = 'm', B = NA))

# Use the data frame method on `simple_exdf$main_data` to confirm that columns
# named `A` and `B` are present
check_required_variables(simple_exdf$main_data, c('A', 'B'))
```

---

check_response_curve_data

Check response curve data for common issues

Description

Checks to make sure an exdf object representing multiple response curves meets basic expectations.

Usage

```
check_response_curve_data(
  exdf_obj,
  identifier_columns,
  expected_npts = 0,
  driving_column = NULL,
  driving_column_tolerance = 1.0,
  col_to_ignore_for_inf = 'gmc',
  constant_col = list(),
  error_on_failure = TRUE,
  print_information = TRUE
)
```

Arguments

exdf_obj	An exdf object representing multiple response curves.
identifier_columns	A vector or list of strings representing the names of columns in exdf_obj that, taken together, uniquely identify each curve. This often includes names like plot, event, replicate, etc.
expected_npts	A numeric vector of length 1 or 2 specifying conditions for the number of points in each curve. If expected_npts is set to a negative number, then this check will be skipped. See below for more details.
driving_column	The name of a column that is systematically varied to produce each curve; for example, in a light response curve, this would typically by Qin. If driving_column is NULL, then this check will be skipped.
driving_column_tolerance	An absolute tolerance for the deviation of each value of driving_column away from its mean across all the curves; the driving_column_tolerance can be set to Inf to disable this check.
col_to_ignore_for_inf	Any columns to ignore while checking for infinite values. Mesophyll conductance (gmc) is often set to infinity intentionally so should be ignored when performing this check. To completely disable this check, set col_to_ignore_for_inf to NULL.
constant_col	A list of named numeric elements, where the name indicates a column of exdf_obj that should be constant, and the value indicates whether the column's values must be identical or whether they must lie within a specified numeric range. If constant_col is an empty list, then this check will be skipped. See below for more details.
error_on_failure	A logical value indicating whether to send an error message when an issue is detected. See details below.
print_information	A logical value indicating whether to print additional information to the R terminal when an issue is detected. See details below.

Details

Basic Behavior:

This function makes a few basic checks to ensure that the response curve data includes the expected information and does not include any mistakes. If no problems are detected, this function will be silent with no return value. If a problem is detected, then the user will be notified in one or more ways:

- If `error_on_failure` is `TRUE`, then this function will throw an error with a short message. If `print_information` is also `TRUE`, then additional information will be printed to the R terminal.
- If `error_on_failure` is `FALSE` and `print_information` is also `FALSE`, then this function will throw a warning with a short message.
- If `error_on_failure` is `FALSE` and `print_information` is true, information about the problem will be printed to the R terminal.

This function will (optionally) perform several checks:

- Checking for infinite values: If `col_to_ignore_for_inf` is not `NULL`, no numeric columns in `exdf_obj` should have infinite values, with the exception of columns designated in `col_to_ignore_for_inf`.
- Checking required columns: All elements of `identifier_columns` should be present as columns in `exdf_obj`. If `driving_column` is not `NULL`, it should also be present as a column in `exdf_obj`. If `constant_col` is not empty, then these columns must also be present in `exdf_obj`.
- Checking the number of points in each curve: The general idea is to ensure that each curve has the expected number of points. Several options can be specified via the value of `expected_npts`, as discussed below.
- Checking the driving column: If `driving_column` is not `NULL`, then each curve should have the same sequence of values in this column. To allow for small variations, a nonzero `driving_column_tolerance` can be specified.
- Checking the constant columns: If `constant_col` is not empty, then each specified column should either be constant, or only vary by a specified amount. See details below.

By default, most of these are not performed (except the simplest ones like checking for infinite values or checking that key columns are present). This enables an "opt-in" use style, where users can specify just the checks they wish to make.

More Details:

There are several options for checking the number of points in each curve:

- If `expected_npts` is a single negative number, no check will be performed.
- If `expected_npts` is 0, then each curve is expected to have the same number of points.
- If `expected_npts` is a single positive number, then each curve is expected to have that many points. For example, if `expected_npts` is 7, then each curve must have 7 points.
- If `expected_npts` is a pair of positive numbers, then each curve is expected to have a number of points lying within the range defined by `expected_npts`. For example, if `expected_npts` is `c(6, 8)`, then each curve must have no fewer than 6 points and no more than 8 points.

- If `expected_npts` is a pair of numbers, one of which is zero and one of which is positive, then the positive number specifies a range; each curve must differ from the average number of points by less than the range. For example, if `expected_npts` is `c(0, 3)`, then every curve must have a number of points within 3 of the average number of points.

There are two options for checking columns that should be constant:

- A value of `NA` indicates that all values of that column must be exactly identical; this check applies for numeric and character columns.
- A numeric value indicates that the range of values of that column must be smaller than the specified range; this range applies for numeric columns only.

For example, setting `constant_col = list(species = NA, Qin = 10)` means that each curve must have only a single value of the `species` column, and that the value of the `Qin` column cannot vary by more than 10 across each curve.

Use Cases:

Using `check_response_curve_data` is not strictly necessary, but it can be helpful both to you and to anyone else reading your analysis code. Here are a few typical use cases:

- **Average response curves:** It is common to calculate and plot average response curves, either manually or by using `xyplot_avg_rc`. But, it only makes sense to do this if each curve followed the same sequence of the driving variable. In this case, `check_response_curve_data` can be used to confirm that each curve used the same values of `CO2_r_sp` (for an A-Ci curve) or `Qin` (for an A-Q curve).
- **Removing recovery points:** It is common to wish to remove one or more recovery points from a set of curves. The safest way to do this is to confirm that all the curves use the same sequence of setpoints; then you can be sure that, for example, points 9 and 10 are the recovery points in every curve.
- **Making a statement of expectations:** If you measured a set of A-Ci curves where each curve has 16 points and used the same sequence of `CO2_r` setpoints, you could record this somewhere in your notes. But it would be even more meaningful to use `check_response_curve_data` in your script with `expected_npts` set to 16. If this check passes, then it means not only that your claim is correct, but also that the identifier columns are being interpreted properly.
- **Checking identifiers:** If the data set includes some identifying metadata, such as a species or location, it may be helpful to confirm that each curve has a single value of these "identifier" columns. Otherwise, the data set may be difficult to interpret.
- **Checking measurement conditions:** If the response curves are expected to be measured under constant temperature, humidity, light, or other environmental variables, it may be helpful to confirm that these variables do not vary too much across each individual curve. Otherwise, parameter values estimated from the curves may not be meaningful.

Sometimes the response curves in a large data set were not all measured with the same sequence of setpoints. If only a few different sequences were used, it is possible to split them into groups and separately run `check_response_curve_data` on each group. This scenario is discussed in the Frequently Asked Questions vignette.

Even if none of the above situations are relevant to you, it may still be helpful to run `run check_response_curve_data` but with `expected_npts` set to 0 and `error_on_failure` set to `FALSE`. With these settings, if there are curves with different numbers of points, the function will print the number of points in each

curve to the R terminal, but won't stop the rest of the script from running. This can be useful for detecting problems with the `curve_identifier` column. For example, if the longest curves in the set are known to have 17 points, but `check_response_curve_data` identifies a curve with 34 points, it is clear that the same identifier was accidentally used for two different curves.

Value

The `check_response_curve_data` function does not return anything.

Examples

```
# Read an example Licor file included in the PhotoGEA package and check it.
# This file includes several 7-point light-response curves that can be uniquely
# identified by the values of its 'species' and 'plot' columns. Since these are
# light-response curves, each one follows a pre-set sequence of `Qin` values.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Make sure there are no infinite values and that all curves have the same
# number of points
check_response_curve_data(licor_file, c('species', 'plot'))

# Make sure there are no inifinite values and that all curves have 7 points
check_response_curve_data(licor_file, c('species', 'plot'), expected_npts = 7)

# Make sure there are no infinite values, that all curves have 7 points, and
# that the values of the `Qin` column follow the same sequence in all curves
# (to within 1.0 micromol / m^2 / s)
check_response_curve_data(
  licor_file,
  c('species', 'plot'),
  expected_npts = 7,
  driving_column = 'Qin',
  driving_column_tolerance = 1.0
)

# Make sure that there are no infinite values and that all curves have between
# 8 and 10 points; this will intentionally fail
check_response_curve_data(
  licor_file,
  c('species', 'plot'),
  expected_npts = c(8, 10),
  error_on_failure = FALSE
)

# Split the data set by `species` and make sure all curves have similar numbers
# of points (within 3 of the mean value); this will intentionally fail
check_response_curve_data(
  licor_file,
  'species',
  expected_npts = c(0, 3),
```

```
  error_on_failure = FALSE
)

# Split the data set by `species` and make sure all curves have a constant value
# of `plot` and a limited range of `TLeafCnd`; this will intentionally fail
check_response_curve_data(
  licor_file,
  'species',
  constant_col = list(plot = NA, TleafCnd = 0.001),
  error_on_failure = FALSE
)
```

`choose_input_files` *Choosing input files*

Description

Tools for choosing input files via dialog windows.

Usage

```
choose_input_files()

choose_input_licor_files()

choose_input_tdl_files()
```

Details

These functions are only available in interactive sessions; moreover, `choose_input_licor_files` and `choose_input_tdl_files` are only available in Microsoft Windows.

- `choose_input_files` will prompt the user to select a single file, and will return full file paths for all files in the same directory that have the same extension.
- `choose_input_licor_files` can be used to select one or more Microsoft Excel files (with extension `*.xlsx`) or plaintext files (with no extension).
- `choose_input_tdl_files` can be used to select one or more TDL data files (with extension `*.dat`).

The outputs from these functions are typically passed to `read_gasex_file` via `lapply`.

Value

A character vector of full file paths.

Examples

```
# Interactively select a single file and get full file paths to all
# other files in the same directory that have the same extension
if (interactive()) {
  file_paths <- choose_input_files()
}


# Interactively select one or more Licor Excel files and read each one to create
# a list of exdf objects
if (interactive() && .Platform$OS.type == "windows") {
  lapply(choose_input_licor_files(), function(fname) {
    read_gasex_file(fname, 'time')
  })
}

# Interactively select one or more TDL data files and read each one to create a
# list of exdf objects
if (interactive() && .Platform$OS.type == "windows") {
  lapply(choose_input_tdl_files(), function(fname) {
    read_gasex_file(fname, 'TIMESTAMP')
  })
}
```

---

confidence_intervals_c3_aci

Calculate confidence intervals for C3 A-Ci fitting parameters

---

Description

Calculates confidence intervals for parameters estimated by a C3 A-Ci curve fit. It is rare for users to call this function directly, because it can be automatically applied to each curve when calling fit_c3_aci.

Usage

```
confidence_intervals_c3_aci(
  replicate_exdf,
  best_fit_parameters,
  lower = list(),
  upper = list(),
  fit_options = list(),
  sd_A = 1,
  relative_likelihood_threshold = 0.147,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
```

```
  gamma_star_norm_column_name = 'Gamma_star_norm',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  cj_crossover_min = NA,
  cj_crossover_max = NA,
  hard_constraints = 0,
  ...
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one CO2 response curve.
`best_fit_parameters`	An `exdf` object representing best-fit parameters for the CO2 response curve in `replicate_exdf`, as calculated by `fit_c3_aci`.
`lower`	The same value that was passed to `fit_c3_aci` when generating `best_fit_parameters`.
`upper`	The same value that was passed to `fit_c3_aci` when generating `best_fit_parameters`.
`fit_options`	The same value that was passed to `fit_c3_aci` when generating `best_fit_parameters`.
`sd_A`	The same value that was passed to `fit_c3_aci` when generating `best_fit_parameters`.
`relative_likelihood_threshold`	The threshold value of relative likelihood used to define the boundaries of the confidence intervals; see details below.
`Wj_coef_C`	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see `calculate_c3_assimilation` for more information.
`Wj_coef_Gamma_star`	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see `calculate_c3_assimilation` for more information.
`a_column_name`	The name of the column in `replicate_exdf` that contains the net assimilation in `micromol m^(-2) s^(-1)`.
`ci_column_name`	The name of the column in `replicate_exdf` that contains the intercellular CO2 concentration in `micromol mol^(-1)`.
`gamma_star_norm_column_name`	The name of the column in `replicate_exdf` that contains the normalized `Gamma_star` values (with units of `normalized to Gamma_star at 25 degrees C`).
`gmc_norm_column_name`	The name of the column in `replicate_exdf` that contains the normalized mesophyll conductance values (with units of `normalized to gmc at 25 degrees C`).

j_norm_column_name
: The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).

ko_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
: The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
: The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

cj_crossover_min
: The minimum value of Cc (in ppm) where Aj is allowed to become the overall rate-limiting factor. If cj_crossover_min is set to NA, this restriction will not be applied.

cj_crossover_max
: The maximim value of Cc (in ppm) where Wj is allowed to be smaller than Wc. If cj_crossover_max is set to NA, this restriction will not be applied.

hard_constraints
: To be passed to calculate_c3_assimilation; see that function for more details.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

In maximum likelihood fitting, each set of parameter values has an associated likelihood value. If the maximum likelihood is known, then it is also possible to define a relative likelihood p according to p = L / L_max. The set of all parameter values where p exceeds a threshold value p_0 defines a region in parameter space called like a "relative likelihood region." When taking one-dimensional cuts through parameter space, the boundaries of the relative likelihood region define a relative likelihood interval.

Here we calculate the upper and lower limits of the relative likelihood intervals for each parameter. This is done by fixing the other parameters to their best-fit values, and varying a single parameter to find the interval where the relative likelihood is above the threshold value (set by the

relative_likelihood_threshold input argument). If the threshold is set to 0.147, then these intervals are equivalent to 95% confidence intervals in most situations. See the Wikipedia page about relative likelihood for more information.

Internally, this function uses error_function_c3_aci to calculate the negative logarithm of the likelihood (-ln(L)). It varies each fitting parameter independendently to find values where ln(L) - ln(p_0) - ln(L_max) = 0.

If the upper limit of a confidence interval is found to exceed ten times the upper limit specified when fitting that parameter, then the upper limit of the condfidence interval is taken to be infinity.

Value

An exdf object based on best_fit_parameters that contains lower and upper bounds for each parameter; for example, if Vcmax_at_25 was fit, best_fit_parameters will contain new columns called Vcmax_at_25_lower and Vcmax_at_25_upper.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Fit just one curve from the data set
one_result <- fit_c3_aci(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  calculate_confidence_intervals = FALSE
)

# Calculate confidence limits for the fit parameters
parameters_with_limits <- confidence_intervals_c3_aci(
    licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
    one_result$parameters
)
```

```
# View confidence limits and best estimate for Vcmax_at_25
parameters_with_limits[, c('Vcmax_at_25_lower', 'Vcmax_at_25', 'Vcmax_at_25_upper')]
```

`confidence_intervals_c3_variable_j` *Calculate confidence intervals for C3 Variable J fitting parameters*

Description

Calculates confidence intervals for parameters estimated by a C3 A-Ci curve fit. It is rare for users to call this function directly, because it can be automatically applied to each curve when calling `fit_c3_variable_j`.

Usage

```
confidence_intervals_c3_variable_j(
  replicate_exdf,
  best_fit_parameters,
  lower = list(),
  upper = list(),
  fit_options = list(),
  sd_A = 1,
  relative_likelihood_threshold = 0.147,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  phips2_column_name = 'PhiPS2',
  qin_column_name = 'Qin',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  cj_crossover_min = NA,
  cj_crossover_max = NA,
  hard_constraints = 0,
  require_positive_gmc = 'positive_a',
  gmc_max = Inf,
  check_j = TRUE,
  ...
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
best_fit_parameters	An exdf object representing best-fit parameters for the CO2 response curve in replicate_exdf, as calculated by fit_c3_variable_j.
lower	The same value that was passed to fit_c3_variable_j when generating best_fit_parameters.
upper	The same value that was passed to fit_c3_variable_j when generating best_fit_parameters.
fit_options	The same value that was passed to fit_c3_variable_j when generating best_fit_parameters.
sd_A	The same value that was passed to fit_c3_variable_j when generating best_fit_parameters.
relative_likelihood_threshold	The threshold value of relative likelihood used to define the boundaries of the confidence intervals; see details below.
Wj_coef_C	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
Wj_coef_Gamma_star	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_norm_column_name	The name of the column in replicate_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).
j_norm_column_name	The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).
kc_norm_column_name	The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).
ko_norm_column_name	The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).
oxygen_column_name	The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.
phips2_column_name	The name of the column in replicate_exdf that contains values of the operating efficiency of photosystem II (dimensionless).
qin_column_name	The name of the column in replicate_exdf that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).

rl_norm_column_name
: The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

cj_crossover_min
: To be passed to error_function_c3_variable_j.

cj_crossover_max
: To be passed to error_function_c3_variable_j.

hard_constraints
: To be passed to calculate_c3_assimilation and calculate_c3_variable_j; see those functions for more details.

require_positive_gmc
: To be passed to error_function_c3_variable_j.

gmc_max
: To be passed to error_function_c3_variable_j.

check_j
: To be passed to error_function_c3_variable_j.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

In maximum likelihood fitting, each set of parameter values has an associated likelihood value. If the maximum likelihood is known, then it is also possible to define a relative likelihood p according to p = L / L_max. The set of all parameter values where p exceeds a threshold value p_0 defines a region in parameter space called like a "relative likelihood region." When taking one-dimensional cuts through parameter space, the boundaries of the relative likelihood region define a relative likelihood interval.

Here we calculate the upper and lower limits of the relative likelihood intervals for each parameter. This is done by fixing the other parameters to their best-fit values, and varying a single parameter to find the interval where the relative likelihood is above the threshold value (set by the relative_likelihood_threshold input argument). If the threshold is set to 0.147, then these intervals are equivalent to 95% confidence intervals in most situations. See the Wikipedia page about relative likelihood for more information.

Internally, this function uses error_function_c3_variable_j to calculate the negative logarithm of the likelihood (-ln(L)). It varies each fitting parameter independendently to find values where ln(L) - ln(p_0) - ln(L_max) = 0.

If the upper limit of a confidence interval is found to exceed ten times the upper limit specified when fitting that parameter, then the upper limit of the condfidence interval is taken to be infinity.

Value

An exdf object based on best_fit_parameters that contains lower and upper bounds for each parameter; for example, if Vcmax_at_25 was fit, best_fit_parameters will contain new columns called Vcmax_at_25_lower and Vcmax_at_25_upper.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Fit just one curve from the data set, using a less reliable optimizer so the
# example runs faster
one_result <- fit_c3_variable_j(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  optim_fun = optimizer_nmkb(1e-7),
  calculate_confidence_intervals = FALSE
)

# Calculate confidence limits for the fit parameters
parameters_with_limits <- confidence_intervals_c3_variable_j(
    licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
    one_result$parameters
)

# View confidence limits and best estimate for Vcmax_at_25
parameters_with_limits[, c('Vcmax_at_25_lower', 'Vcmax_at_25', 'Vcmax_at_25_upper')]
```

---

confidence_intervals_c4_aci

Calculate confidence intervals for C4 A-Ci fitting parameters

---

Description

Calculates confidence intervals for parameters estimated by a C4 A-Ci curve fit. It is rare for users to call this function directly, because it can be automatically applied to each curve when calling `fit_c4_aci`.

Usage

```
confidence_intervals_c4_aci(
  replicate_exdf,
  best_fit_parameters,
  lower = list(),
  upper = list(),
  fit_options = list(),
  sd_A = 1,
  relative_likelihood_threshold = 0.147,
  x_etr = 0.4,
  a_column_name = 'A',
  ao_column_name = 'ao',
  ci_column_name = 'Ci',
  gamma_star_column_name = 'gamma_star',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kc_column_name = 'Kc',
  ko_column_name = 'Ko',
  kp_column_name = 'Kp',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  vcmax_norm_column_name = 'Vcmax_norm',
  vpmax_norm_column_name = 'Vpmax_norm',
  hard_constraints = 0
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one CO2 response curve.
`best_fit_parameters`	An `exdf` object representing best-fit parameters for the CO2 response curve in `replicate_exdf`, as calculated by `fit_c4_aci`.
`lower`	The same value that was passed to `fit_c4_aci` when generating `best_fit_parameters`.
`upper`	The same value that was passed to `fit_c4_aci` when generating `best_fit_parameters`.
`fit_options`	The same value that was passed to `fit_c4_aci` when generating `best_fit_parameters`.
`sd_A`	The same value that was passed to `fit_c4_aci` when generating `best_fit_parameters`.
`relative_likelihood_threshold`	The threshold value of relative likelihood used to define the boundaries of the confidence intervals; see details below.

x_etr The fraction of whole-chain electron transport occurring in the mesophyll (dimensionless). See Equation 29 from S. von Caemmerer (2021).

a_column_name The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).

ao_column_name The name of the column in replicate_exdf that contains the dimensionless ratio of solubility and diffusivity of O2 to CO2.

ci_column_name The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).

gamma_star_column_name
The name of the column in replicate_exdf that contains the dimensionless gamma_star values.

gmc_norm_column_name
The name of the column in replicate_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).

j_norm_column_name
The name of the column in exdf_obj that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for rubisco carboxylation in microbar.

ko_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for rubisco oxygenation in mbar.

kp_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for PEP carboxylase carboxylation in microbar.

oxygen_column_name
The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
The name of the column in exdf_obj that contains the total pressure in bar.

vcmax_norm_column_name
The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

vpmax_norm_column_name
The name of the column in replicate_exdf that contains the normalized Vpmax values (with units of normalized to Vpmax at 25 degrees C).

hard_constraints
To be passed to calculate_c4_assimilation; see that function for more details.

Details

In maximum likelihood fitting, each set of parameter values has an associated likelihood value. If the maximum likelihood is known, then it is also possible to define a relative likelihood p according

to p = L / L_max. The set of all parameter values where p exceeds a threshold value p_0 defines a region in parameter space called like a "relative likelihood region." When taking one-dimensional cuts through parameter space, the boundaries of the relative likelihood region define a relative likelihood interval.

Here we calculate the upper and lower limits of the relative likelihood intervals for each parameter. This is done by fixing the other parameters to their best-fit values, and varying a single parameter to find the interval where the relative likelihood is above the threshold value (set by the relative_likelihood_threshold input argument). If the threshold is set to 0.147, then these intervals are equivalent to 95% confidence intervals in most situations. See the Wikipedia page about relative likelihood for more information.

Internally, this function uses error_function_c4_aci to calculate the negative logarithm of the likelihood (-ln(L)). It varies each fitting parameter independendently to find values where ln(L) - ln(p_0) - ln(L_max) = 0.

If the upper limit of a confidence interval is found to exceed ten times the upper limit specified when fitting that parameter, then the upper limit of the condfidence interval is taken to be infinity.

Value

An exdf object based on best_fit_parameters that contains lower and upper bounds for each parameter; for example, if Vcmax_at_25 was fit, best_fit_parameters will contain new columns called Vcmax_at_25_lower and Vcmax_at_25_upper.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate temperature-dependent values of C4 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c4_temperature_param_vc)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Fit just one curve from the data set
one_result <- fit_c4_aci(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE],
```

```
  calculate_confidence_intervals = FALSE
)

# Calculate confidence limits for the fit parameters
parameters_with_limits <- confidence_intervals_c4_aci(
    licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE],
    one_result$parameters
)

# View confidence limits and best estimate for Vcmax_at_25
parameters_with_limits[, c('Vcmax_at_25_lower', 'Vcmax_at_25', 'Vcmax_at_25_upper')]
```

`confidence_intervals_c4_aci_hyperbola` *Calculate confidence intervals for C4 A-Ci hyperbola fitting parameters*

Description

Calculates confidence intervals for parameters estimated by a C4 A-Ci curve fit. It is rare for users to call this function directly, because it can be automatically applied to each curve when calling `fit_c4_aci_hyperbola`.

Usage

```
confidence_intervals_c4_aci_hyperbola(
  replicate_exdf,
  best_fit_parameters,
  lower = list(),
  upper = list(),
  fit_options = list(),
  sd_A = 1,
  relative_likelihood_threshold = 0.147,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  hard_constraints = 0
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one CO2 response curve.
`best_fit_parameters`	An `exdf` object representing best-fit parameters for the CO2 response curve in `replicate_exdf`, as calculated by `fit_c4_aci_hyperbola`.
`lower`	The same value that was passed to `fit_c4_aci_hyperbola` when generating `best_fit_parameters`.
`upper`	The same value that was passed to `fit_c4_aci_hyperbola` when generating `best_fit_parameters`.

`fit_options` The same value that was passed to `fit_c4_aci_hyperbola` when generating `best_fit_parameters`.

`sd_A` The same value that was passed to `fit_c4_aci_hyperbola` when generating `best_fit_parameters`.

`relative_likelihood_threshold`
The threshold value of relative likelihood used to define the boundaries of the confidence intervals; see details below.

`a_column_name` The name of the column in `replicate_exdf` that contains the net assimilation in `micromol m^(-2) s^(-1)`.

`ci_column_name` The name of the column in `exdf_obj` that contains the intercellular CO2 concentration, expressed in `micromol mol^(-1)`.

`hard_constraints`
To be passed to `calculate_c4_assimilation_hyperbola`; see that function for more details.

Details

In maximum likelihood fitting, each set of parameter values has an associated likelihood value. If the maximum likelihood is known, then it is also possible to define a relative likelihood `p` according to `p = L / L_max`. The set of all parameter values where `p` exceeds a threshold value `p_0` defines a region in parameter space called like a "relative likelihood region." When taking one-dimensional cuts through parameter space, the boundaries of the relative likelihood region define a relative likelihood interval.

Here we calculate the upper and lower limits of the relative likelihood intervals for each parameter. This is done by fixing the other parameters to their best-fit values, and varying a single parameter to find the interval where the relative likelihood is above the threshold value (set by the `relative_likelihood_threshold` input argument). If the threshold is set to 0.147, then these intervals are equivalent to 95% confidence intervals in most situations. See the Wikipedia page about relative likelihood for more information.

Internally, this function uses `error_function_c4_aci_hyperbola` to calculate the negative logarithm of the likelihood (`-ln(L)`). It varies each fitting parameter independendently to find values where `ln(L) - ln(p_0) - ln(L_max) = 0`.

If the upper limit of a confidence interval is found to exceed ten times the upper limit specified when fitting that parameter, then the upper limit of the condfidence interval is taken to be infinity.

Value

An `exdf` object based on `best_fit_parameters` that contains lower and upper bounds for each parameter; for example, if `Vmax` was fit, `best_fit_parameters` will contain new columns called `Vmax_lower` and `Vmax_upper`.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)
```

```
# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Fit just one curve from the data set
one_result <- fit_c4_aci_hyperbola(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE],
  calculate_confidence_intervals = FALSE
)

# Calculate confidence limits for the fit parameters
parameters_with_limits <- confidence_intervals_c4_aci_hyperbola(
    licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE],
    one_result$parameters
)

# View confidence limits and best estimate for Vmax
parameters_with_limits[, c('Vmax_lower', 'Vmax', 'Vmax_upper')]
```

`consolidate` *Consolidate a list of lists*

Description

Consolidates a list of lists into a regular list by combining like-named elements.

Usage

```
  consolidate(x)

  ## S3 method for class 'data.frame'
consolidate(x)

  ## S3 method for class 'exdf'
consolidate(x)
```

Arguments

`x`	A list of lists `list_1`, `list_2`, ..., `list_N`, where each sub-list `list_i` has elements named `name_1`, `name_2`, ..., `name_M`.

Details

consolidate is generic, with methods defined for nested lists of data frames and exdf objects.

Value

A list with elements named name_1, name_2, ..., name_M, where each element was created by combining all elements of x with the same name using rbind; for example, the element with name name_1 will be created by calling rbind(list_1$name_1, list_2$name_1, ..., list_N$name_1). Before calling rbind, each element will be limited to the columns that are common to all elements with the same name.

See Also

exdf

Examples

```
# Example 1: Create a nested list of data frames and then consolidate them into
# a regular list by combining the like-named elements
nested_df_list <- list(
  list_1 = list(
    name_1 = data.frame(A = c(1, 2), B = c(0, 0)),
    name_2 = data.frame(A = c(3, 4), B = c(0, 0)),
    name_3 = data.frame(A = c(5, 6), B = c(0, 0))
  ),
  list_2 = list(
    name_1 = data.frame(A = c(7, 8), B = c(0, 0)),
    name_2 = data.frame(A = c(9, 10), B = c(0, 0)),
    name_3 = data.frame(A = c(11, 12), B = c(0, 0))
  ),
  list_3 = list(
    name_1 = data.frame(A = c(13, 14), B = c(0, 0)),
    name_2 = data.frame(A = c(15, 16), B = c(0, 0)),
    name_3 = data.frame(A = c(17, 18), B = c(0, 0))
  )
)

str(nested_df_list)

consolidated_df_list <- consolidate(nested_df_list)

str(consolidated_df_list)

# Example 2: Create a nested list of `exdf` objects and then consolidate them
# into a regular list by combining the like-named elements. Here, some of the
# elements have columns not present in the others (for example,
# `nested_exdf_list$list_3$name_1`). However, these "extra" columns are removed
# before calling `rbind` and they do not appear in `consolidated_exdf_list`.
nested_exdf_list <- list(
  list_1 = list(
    name_1 = exdf(data.frame(A = c(1, 2), B = c(0, 0))),
```

```
    name_2 = exdf(data.frame(A = c(3, 4), B = c(0, 0))),
    name_3 = exdf(data.frame(A = c(5, 6), B = c(0, 0)))
  ),
  list_2 = list(
    name_1 = exdf(data.frame(A = c(7, 8), B = c(0, 0))),
    name_2 = exdf(data.frame(A = c(9, 10), B = c(0, 0))),
    name_3 = exdf(data.frame(A = c(11, 12), B = c(0, 0)))
  ),
  list_3 = list(
    name_1 = exdf(data.frame(A = c(13, 14), B = c(0, 0), C = c(-1, -2))),
    name_2 = exdf(data.frame(A = c(15, 16), B = c(0, 0), C = c(-1, -2))),
    name_3 = exdf(data.frame(A = c(17, 18), B = c(0, 0), C = c(-1, -2)))
  )
)

str(nested_exdf_list)

consolidated_exdf_list <- consolidate(nested_exdf_list)

str(consolidated_exdf_list)
```

`csv.exdf` *Read and write CSV files representing an exdf object*

Description

Functions for reading and writing CSV files that represent an `exdf` object.

Usage

```
read.csv.exdf(file, ...)

write.csv.exdf(x, file, ...)
```

Arguments

`file`	The name of the file which the data are to be read from; to be passed to `read.csv`.
`...`	Additional arguments to be passed to `read.csv` or `write.csv`. Note that some arguments cannot be specified; an error message will be sent if a used attempts to set one of these forbidden arguments.
`x`	An `exdf` object to be written to a CSV file.

Details

An `exdf` object can be written to a CSV file by directly calling `write.csv`, but this approach causes some column names to be unintentionally modified. For example, any spaces will be replaced by periods. This can potentially cause problems when reloading the data from the CSV file.

Instead, it is preferred to use `write.csv.exdf`, which will not modify any column names. When writing the CSV file, it is saved with the column names in the first row, the categories in the second row, the units in the third row, and the data in the remaining rows.

The resulting file can be read using `read.csv.exdf`. Here, the names, categories, and units are read from the first three rows of the specified file, and the data values from the remaining rows. An `exdf` object is then created from this information.

Value

The `write.csv.exdf` function does not return anything. The `read.csv.exdf` function returns an `exdf` object representing the contents of `file`.

Examples

```
# Read a CSV file included with the PhotoGEA package; this file was created
# using `write.csv.exdf`.
licor_file <- read.csv.exdf(
  PhotoGEA_example_file_path('ball_berry_1.csv')
)

# Now rewrite this to a temporary CSV file
tf <- tempfile(fileext = ".csv")
tf

write.csv.exdf(licor_file, tf)
```

---

deprecated *Deprecated functions*

Description

Deprecated functions that will be fully removed in future releases. Each of these functions will produce an error when called that will redirect the user to a suitable replacement.

Usage

```
read_tdl_file(...)

read_licor_file(...)

check_licor_data(...)

calculate_arrhenius(...)

calculate_peaked_gaussian(...)
```

Arguments

`...`	Additional arguments; currently unused.

Value

None of the deprecated functions return anything.

Examples

```
# These functions all throw errors, so we will wrap them in `tryCatch` here

tryCatch(
  read_tdl_file(),
  error = function(e) {print(e)}
)

tryCatch(
  read_licor_file(),
  error = function(e) {print(e)}
)

tryCatch(
  check_licor_data(),
  error = function(e) {print(e)}
)

tryCatch(
  calculate_arrhenius(),
  error = function(e) {print(e)}
)

tryCatch(
  calculate_peaked_gaussian(),
  error = function(e) {print(e)}
)
```

---

dim.exdf *Retrieve the dimension of an exdf object*

Description

Returns the dimensions of an `exdf` object's `main_data`. Also enables `nrow` and `ncol` for `exdf` objects.

Usage

```
  ## S3 method for class 'exdf'
dim(x)
```

Arguments

`x`	An `exdf` object.

Value

Returns `dim(x[['main_data']])`.

See Also

exdf

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))

dim(simple_exdf)
dim(simple_exdf[['main_data']]) # An equivalent command

nrow(simple_exdf)
ncol(simple_exdf)
```

---

dimnames.exdf *Retrieve or set the dimension names of an exdf object*

Description

Returns or sets the dimension names of an `exdf` object's `main_data`. When setting names, the column names of the `exdf` object's `units` and `categories` are also set. Also enables `colnames` and `rownames` for `exdf` objects.

Usage

```
  ## S3 method for class 'exdf'
dimnames(x)

  ## S3 replacement method for class 'exdf'
dimnames(x) <- value
```

Arguments

`x`	An `exdf` object.
`value`	A possible value for `dimnames(x)`

Value

Returns `dimnames(x[['main_data']])`.

See Also

exdf

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))

dimnames(simple_exdf)
dimnames(simple_exdf[['main_data']]) # An equivalent command

colnames(simple_exdf) <- "B"
rownames(simple_exdf) <- 2

colnames(simple_exdf)
rownames(simple_exdf)
```

`document_variables` *Document exdf columns by specifying units and categories*

Description

Adds new columns to a table-like object, and sets/modifies the units or categories of columns in an `exdf` object.

Usage

```
  document_variables(x, ...)

  ## S3 method for class 'data.frame'
document_variables(x, ...)

  ## S3 method for class 'exdf'
document_variables(x, ...)
```

Arguments

`x`	A table-like R object such as a data frame or an `exdf`.
`...`	Each optional argument should be a character vector with three elements that describe a column, where the first element is the category, the second is the name, and the third is the units. For example, `c('GasEx', 'A', 'micromol m^(-2) s^(-1)')` specifies that the category and units for the `A` column are `GasEx` and `micromol m^(-2) s^(-1)`, respectively. If the column name is not in `x`, it will be added with all values initialized to `NA`. Categories and units will be ignored when `x` is a data frame.

Value

An object based on `x` with new and/or modified columns.

See Also

`exdf`

Examples

```
# Create a simple exdf object with two columns (`A` and `B`) and default values
# for its units and categories.
simple_exdf <- exdf(data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)))

print(simple_exdf)

# Specify units and categories for the `A` and `B` columns, and add a new `C`
# column.
document_variables(
  simple_exdf,
  c('cat1', 'A', 'm'), # The category of `A` is `cat1` and its units are `m`
  c('cat2', 'B', 's'), # The category of `B` is `cat2` and its units are `s`
  c('cat3', 'C', 'g')  # The category of `C` is `cat3` and its units are `g`
)

# Do the same but for a data frame; in this case columns A and B will not be
# altered, but a new column C will be added (and initialized to NA)
document_variables(
  simple_exdf$main_data,
  c('cat1', 'A', 'm'), # The category of `A` is `cat1` and its units are `m`
  c('cat2', 'B', 's'), # The category of `B` is `cat2` and its units are `s`
  c('cat3', 'C', 'g')  # The category of `C` is `cat3` and its units are `g`
)
```

---

error_function_c3_aci *Generate an error function for C3 A-Ci curve fitting*

Description

Creates a function that returns an error value (the negative of the natural logarithm of the likelihood) representing the amount of agreement between modeled and measured An values. When this function is minimized, the likelihood is maximized.

Internally, this function uses apply_gm to calculate Cc, and then uses link{calculate_c3_assimilation} to calculate assimilation rate values that are compared to the measured ones.

Usage

```
error_function_c3_aci(
  replicate_exdf,
  fit_options = list(),
  sd_A = 1,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  gmc_norm_column_name = 'gmc_norm',
```

```
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  cj_crossover_min = NA,
  cj_crossover_max = NA,
  hard_constraints = 0,
  debug_mode = FALSE,
  ...
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
fit_options	A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in replicate_exdf of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(alpha_g = 0, Vcmax_at_25 = 'fit', Tp_at_25 = 'column') means that alpha_g will be set to 0, Vcmax_at_25 will be fit, and Tp_at_25 will be set to the values in the Tp_at_25 column of replicate_exdf.
sd_A	The standard deviation of the measured values of the net CO2 assimilation rate, expressed in units of micromol m^(-2) s^(-1). If sd_A is not a number, then there must be a column in replicate_exdf called sd_A with appropriate units. A numeric value supplied here will overwrite the values in the sd_A column of replicate_exdf if it exists.
Wj_coef_C	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
Wj_coef_Gamma_star	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_norm_column_name	The name of the column in replicate_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).

gmc_norm_column_name
: The name of the column in replicate_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).

j_norm_column_name
: The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).

ko_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
: The name of the column in replicate_exdf that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
: The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

cj_crossover_min
: The minimum value of Cc (in ppm) where Aj is allowed to become the overall rate-limiting factor. If cj_crossover_min is set to NA, this restriction will not be applied.

cj_crossover_max
: The maximim value of Cc (in ppm) where Wj is allowed to be smaller than Wc. If cj_crossover_max is set to NA, this restriction will not be applied.

hard_constraints
: To be passed to calculate_c3_assimilation; see that function for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the guess is printed each time the error function is called; this can be helpful when troubleshooting issues with an optimizer.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

When fitting A-Ci curves using a maximum likelihood approach, it is necessary to define a function that calculates the likelihood of a given set of alpha_g, alpha_old, alpha_s, alpha_t,

Gamma_star_at_25, gmc_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 values by comparing a model prediction to a measured curve. This function will be passed to an optimization algorithm which will determine the values that produce the largest likelihood.

The error_function_c3_aci returns such a function, which is based on a particular A-Ci curve and a set of fitting options. It is possible to just fit a subset of the available fitting parameters; by default, the fitting parameters are alpha_old, J_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25. This behavior can be changed via the fit_options argument.

For practical reasons, the function actually returns values of -ln(L), where L is the likelihood. The logarithm of L is simpler to calculate than L itself, and the minus sign converts the problem from a maximization to a minimization, which is important because most optimizers are designed to minimize a value.

Sometimes an optimizer will choose biologically unreasonable parameter values that nevertheless produce good fits to the supplied assimilation values. A common problem is that the fit result may not indicate Ac-limited assimilation at low CO2 values, which should be the case for any A-Ci curves measured at saturating light. In this case, the optional cj_crossover_min and cj_crossover_max can be used to constrain the range of Cc values (in ppm) where Aj is allowed to be the overall rate limiting factor. If the crossover from Rubisco-limited to RuBP-regeneration limited assimilation occurs outside these bounds (when they are supplied), a heavy penalty will be added to the error function, preventing the optimizer from choosing those parameter values.

A penalty is also added for any parameter combination where An is not a number, or where calculate_c3_assimilation produces an error.

Value

A function with one input argument guess, which should be a numeric vector representing values of the parameters to be fitted (which are specified by the fit_options input argument.) Each element of guess is the value of one parameter (arranged in alphabetical order.) For example, with the default settings, guess should contain values of alpha_old, J_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)
```

```
# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Define an error function for one curve from the set
error_fcn <- error_function_c3_aci(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE]
)

# Evaluate the error for:
#  alpha_old = 0
#  J_at_25 = 236
#  RL_at_25 = 4e-8
#  Tp_at_25 = 22.7
#  Vcmax_at_25 = 147
error_fcn(c(0, 236, 4e-8, 22.7, 147))

# Make a plot of likelihood vs. Vcmax when other parameters are fixed to the
# values above.
vcmax_error_fcn <- function(Vcmax) {error_fcn(c(0, 236, 4e-8, 22.7, Vcmax))}
vcmax_seq <- seq(135, 152, length.out = 41)

lattice::xyplot(
  exp(-sapply(vcmax_seq, vcmax_error_fcn)) ~ vcmax_seq,
  type = 'b',
  xlab = 'Vcmax_at_25 (micromol / m^2 / s)',
  ylab = 'Negative log likelihood (dimensionless)'
)
```

`error_function_c3_variable_j` *Generate an error function for C3 Variable J curve fitting*

Description

Creates a function that returns an error value (the negative of the natural logarithm of the likelihood) representing the amount of agreement between modeled and measured `An` values. When this function is minimized, the likelihood is maximized.

Internally, this function uses `link{calculate_c3_variable_j}` and `link{calculate_c3_assimilation}` to calculate assimilation rate values that are compared to the measured ones.

Usage

```
error_function_c3_variable_j(
  replicate_exdf,
  fit_options = list(),
  sd_A = 1,
```

```
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  phips2_column_name = 'PhiPS2',
  qin_column_name = 'Qin',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  cj_crossover_min = NA,
  cj_crossover_max = NA,
  hard_constraints = 0,
  require_positive_gmc = 'positive_a',
  gmc_max = Inf,
  check_j = TRUE,
  debug_mode = FALSE,
  ...
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
fit_options	A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in replicate_exdf of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(alpha_g = 0, Vcmax_at_25 = 'fit', Tp_at_25 = 'column') means that alpha_g will be set to 0, Vcmax_at_25 will be fit, and Tp_at_25 will be set to the values in the Tp_at_25 column of replicate_exdf.
sd_A	The standard deviation of the measured values of the net CO2 assimilation rate, expressed in units of micromol m^(-2) s^(-1). If sd_A is not a number, then there must be a column in replicate_exdf called sd_A with appropriate units. A numeric value supplied here will overwrite the values in the sd_A column of replicate_exdf if it exists.
Wj_coef_C	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.
Wj_coef_Gamma_star	A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of

RuBP regeneration; see calculate_c3_assimilation for more information.

a_column_name The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).

ci_column_name The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).

gamma_star_norm_column_name
The name of the column in replicate_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).

j_norm_column_name
The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_norm_column_name
The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).

ko_norm_column_name
The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
The name of the column in replicate_exdf that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

phips2_column_name
The name of the column in replicate_exdf that contains values of the operating efficiency of photosystem II (dimensionless).

qin_column_name
The name of the column in replicate_exdf that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).

rl_norm_column_name
The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

cj_crossover_min
The minimum value of Cc (in ppm) where Aj is allowed to become the overall rate-limiting factor. If cj_crossover_min is set to NA, this restriction will not be applied.

cj_crossover_max
The maximim value of Cc (in ppm) where Wj is allowed to be smaller than Wc. If cj_crossover_max is set to NA, this restriction will not be applied.

hard_constraints
: To be passed to calculate_c3_assimilation and calculate_c3_variable_j; see those functions for more details.

require_positive_gmc
: A character string specifying the method to be used for requiring positive values of mesophyll conductance. Can be 'none', 'all', or 'positive_a'. See below for more details.

gmc_max
: The maximum value of mesophyll conductance that should be considered to be acceptable. See below for more details.

check_j
: A logical (TRUE/FALSE) value indicating whether to check whether J_F > J_tl. See below for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the guess is printed each time the error function is called; this can be helpful when troubleshooting issues with an optimizer.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

When fitting A-Ci + chlorophyll fluorescence curves using the Variable J method, it is necessary to define a function that calculates the likelihood of a given set of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star, J_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 values by comparing a model prediction to a measured curve. This function will be passed to an optimization algorithm which will determine the values that produce the smallest error.

The error_function_c3_variable_j returns such a function, which is based on a particular replicate and a set of fitting options. It is possible to just fit a subset of the available fitting parameters; by default, the fitting parameters are alpha_old, J_at_25, RL_at_25, Tp_at_25, tau, and Vcmax_at_25. This behavior can be changed via the fit_options argument.

For practical reasons, the function actually returns values of -ln(L), where L is the likelihood. The logarithm of L is simpler to calculate than L itself, and the minus sign converts the problem from a maximization to a minimization, which is important because most optimizers are designed to minimize a value.

Sometimes an optimizer will choose biologically unreasonable parameter values that nevertheless produce good fits to the supplied assimilation values. There are several options for preventing an optimizer from choosing such parameter values:

- Enforcing Rubisco limitations: A common problem is that the fit result may not indicate Rubisc-limited assimilation at low CO2 values, which should be the case for any A-Ci curves measured at saturating light. In this case, the optional cj_crossover_min and cj_crossover_max can be used to constrain the range of Cc values (in ppm) where Wj is allowed to be the overall rate limiting factor. If the crossover from Rubisco-limited to RuBP-regeneration limited carboxylation occurs outside these bounds (when they are supplied), a heavy penalty will be added to the error function, preventing the optimizer from choosing those parameter values.
- Requiring positive gmc: The Variable J method sometimes predicts negative values of the mesophyll conductance (gmc). Such values are probably not realistic, especially when Cc is above the CO2 compensation point. The require_positive_gmc input argument can be used to penalize negative values of gmc. When require_positive_gmc is 'all', a heavy

penalty will be added to the error function if there are any negative gmc values. When require_positive_gmc is 'positive_a', a heavy penalty will be added to the error function if there are any negative gmc values when A is positive; negative gmc for negative A will be allowed. When require_positive_gmc is 'none', these restrictions are disabled and no penalties are added for negative gmc.

- Preventing large values of gmc: The Variable J method sometimes produces unreasonably high values of gmc. The gmc_max argument can be used to address this. If any predicted gmc values exceed gmc_max when A is positive, a heavy penalty will be added to the error function.
- Enforcing consistent RuBP regeneration rates: In principle, the actual RuBP regeneration rate (J_F) should always be less than or equal to its maximum value for a given Qin and leaf temperature (J_tl), with equality only occuring when assimilation is RuBP-regeneration-limited. When check_j is TRUE, a heavy penalty will be added to the error function for any parameter values where J_F is greater than J_tl at any point in the curve.

A penalty is also added for any parameter combination where An is not a number, or where calculate_c3_variable_j or calculate_c3_assimilation produce an error.

Value

A function with one input argument guess, which should be a numeric vector representing values of the parameters to be fitted (which are specified by the fit_options input argument.) Each element of guess is the value of one parameter (arranged in alphabetical order.) For example, with the default settings, guess should contain values of alpha_old, J_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Define an error function for one curve from the set
```

```
error_fcn <- error_function_c3_variable_j(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE]
)

# Evaluate the error for:
#  alpha_old = 1.9
#  J_at_25 = 270
#  RL_at_25 = 1.9
#  tau = 0.42
#  Tp_at_25 = 8.7
#  Vcmax_at_25 = 258
error_fcn(c(1.9, 270, 1.9, 0.42, 8.7, 258))

# Make a plot of error vs. Tp_at_25 when the other parameters are fixed to the
# values above. As Tp_at_25 increases, it eventually stops limiting the
# assimilation rate and its value stops influencing the error.
tpu_error_fcn <- function(Tp_at_25) {error_fcn(c(1.9, 270, 1.9, 0.42, Tp_at_25, 258))}
tpu_seq <- seq(5, 12, by = 0.25)

lattice::xyplot(
  sapply(tpu_seq, tpu_error_fcn) ~ tpu_seq,
  type = 'b',
  xlab = 'Tp at 25 degrees C (micromol / m^2 / s)',
  ylab = 'Negative log likelihood (dimensionless)'
)
```

`error_function_c4_aci` *Generate an error function for C4 A-Ci curve fitting*

Description

Creates a function that returns an error value (the negative of the natural logarithm of the likelihood) representing the amount of agreement between modeled and measured `An` values. When this function is minimized, the likelihood is maximized.

Internally, this function uses `apply_gm` to calculate `Cc`, and then uses `link{calculate_c4_assimilation}` to calculate assimilation rate values that are compared to the measured ones.

Usage

```
error_function_c4_aci(
  replicate_exdf,
  fit_options = list(),
  sd_A = 1,
  x_etr = 0.4,
  a_column_name = 'A',
  ao_column_name = 'ao',
  ci_column_name = 'Ci',
  gamma_star_column_name = 'gamma_star',
  gmc_norm_column_name = 'gmc_norm',
```

```
    j_norm_column_name = 'J_norm',
    kc_column_name = 'Kc',
    ko_column_name = 'Ko',
    kp_column_name = 'Kp',
    oxygen_column_name = 'Oxygen',
    rl_norm_column_name = 'RL_norm',
    total_pressure_column_name = 'total_pressure',
    vcmax_norm_column_name = 'Vcmax_norm',
    vpmax_norm_column_name = 'Vpmax_norm',
    hard_constraints = 0,
    debug_mode = FALSE
  )
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
fit_options	A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in replicate_exdf of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(RL_at_25 = 0, Vcmax_at_25 = 'fit', Vpmax_at_25 = 'column') means that RL_at_25 will be set to 0, Vcmax_at_25 will be fit, and Vpmax_at_25 will be set to the values in the Vpmax_at_25 column of replicate_exdf.
sd_A	The standard deviation of the measured values of the net CO2 assimilation rate, expressed in units of micromol m^(-2) s^(-1). If sd_A is not a number, then there must be a column in exdf_obj called sd_A with appropriate units. A numeric value supplied here will overwrite the values in the sd_A column of exdf_obj if it exists.
x_etr	The fraction of whole-chain electron transport occurring in the mesophyll (dimensionless). See Equation 29 from S. von Caemmerer (2021).
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ao_column_name	The name of the column in replicate_exdf that contains the dimensionless ratio of solubility and diffusivity of O2 to CO2.
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_column_name	The name of the column in replicate_exdf that contains the dimensionless gamma_star values.
gmc_norm_column_name	The name of the column in replicate_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).
j_norm_column_name	The name of the column in exdf_obj that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for rubisco carboxylation in microbar.

ko_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for rubisco oxygenation in mbar.

kp_column_name The name of the column in replicate_exdf that contains the Michaelis-Menten constant for PEP carboxylase carboxylation in microbar.

oxygen_column_name
The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
The name of the column in exdf_obj that contains the total pressure in bar.

vcmax_norm_column_name
The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

vpmax_norm_column_name
The name of the column in replicate_exdf that contains the normalized Vpmax values (with units of normalized to Vpmax at 25 degrees C).

hard_constraints
To be passed to calculate_c4_assimilation; see that function for more details.

debug_mode A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the guess is printed each time the error function is called; this can be helpful when troubleshooting issues with an optimizer.

Details

When fitting A-Ci curves, it is necessary to define a function that calculates the likelihood of a given set of alpha_psii, gbs, gmc_at_25, J_at_25, RL_at_25, Rm_frac, Vcmax_at_25, Vpmax_at_25, and Vpr values by comparing a model prediction to a measured curve. This function will be passed to an optimization algorithm which will determine the values that produce the smallest error.

The error_function_c4_aci returns such a function, which is based on a particular A-Ci curve and a set of fitting options. It is possible to just fit a subset of the available fitting parameters; by default, the fitting parameters are RL_at_25, Vcmax_at_25, and Vpmax_at_25. This behavior can be changed via the fit_options argument.

For practical reasons, the function actually returns values of -ln(L), where L is the likelihood. The logarithm of L is simpler to calculate than L itself, and the minus sign converts the problem from a maximization to a minimization, which is important because most optimizers are designed to minimize a value.

A penalty is added to the error value for any parameter combination where An is not a number, or where calculate_c4_assimilation produces an error.

Value

A function with one input argument guess, which should be a numeric vector representing values of the parameters to be fitted (which are specified by the fit_options input argument.) Each element of guess is the value of one parameter (arranged in alphabetical order.) For example, with the default settings, guess should contain values of RL_at_25, Vcmax_at_25, and Vpmax_at_25 (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate temperature-dependent values of C4 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c4_temperature_param_vc)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Define an error function for one curve from the set
error_fcn <- error_function_c4_aci(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE]
)

# Evaluate the error for RL_at_25 = 0, Vcmax_at_25 = 35, Vpmax_at_25 = 180
error_fcn(c(0, 35, 180))

# Make a plot of error vs. Vcmax_at_25 when the other parameters are fixed to
# the values above.
vcmax_error_fcn <- function(Vcmax_at_25) {error_fcn(c(0, Vcmax_at_25, 180))}
vcmax_seq <- seq(20, 50)

lattice::xyplot(
  sapply(vcmax_seq, vcmax_error_fcn) ~ vcmax_seq,
  type = 'b',
  xlab = 'Vcmax at 25 degrees C (micromol / m^2 / s)',
  ylab = 'Negative log likelihood (dimensionless)'
)
```

`error_function_c4_aci_hyperbola` *Generate an error function for C4 A-Ci curve fitting with a hyperbola*

Description

Creates a function that returns an error value (the negative of the natural logarithm of the likelihood) representing the amount of agreement between modeled and measured `An` values. When this function is minimized, the likelihood is maximized.

Internally, this function uses `link{calculate_c4_assimilation_hyperbola}` to calculate assimilation rate values that are compared to the measured ones.

Usage

```
error_function_c4_aci_hyperbola(
  replicate_exdf,
  fit_options = list(),
  sd_A = 1,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  hard_constraints = 0,
  debug_mode = FALSE
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one CO2 response curve.
`fit_options`	A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be `'fit'`, `'column'`, or a numeric value. A value of `'fit'` means that the parameter will be fit; a value of `'column'` means that the value of the parameter will be taken from a column in `replicate_exdf` of the same name; and a numeric value means that the parameter will be set to that value. For example, `fit_options = list(rL = 0, Vmax = 'fit', c4_curvature = 'column')` means that `rL` will be set to 0, `Vmax` will be fit, and `c4_curvature` will be set to the values in the `c4_curvature` column of `replicate_exdf`.
`sd_A`	The standard deviation of the measured values of the net CO2 assimilation rate, expressed in units of `micromol m^(-2) s^(-1)`. If `sd_A` is not a number, then there must be a column in `exdf_obj` called `sd_A` with appropriate units. A numeric value supplied here will overwrite the values in the `sd_A` column of `exdf_obj` if it exists.
`a_column_name`	The name of the column in `replicate_exdf` that contains the net assimilation in `micromol m^(-2) s^(-1)`.
`ci_column_name`	The name of the column in `exdf_obj` that contains the intercellular CO2 concentration, expressed in `micromol mol^(-1)`.

hard_constraints
: To be passed to calculate_c4_assimilation_hyperbola; see that function for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the guess is printed each time the error function is called; this can be helpful when troubleshooting issues with an optimizer.

Details

When fitting A-Ci curves, it is necessary to define a function that calculates the likelihood of a given set of c4_curvature, c4_slope, rL, and Vmax values by comparing a model prediction to a measured curve. This function will be passed to an optimization algorithm which will determine the values that produce the smallest error.

The error_function_c4_aci_hyperbola returns such a function, which is based on a particular A-Ci curve and a set of fitting options. It is possible to just fit a subset of the available fitting parameters; by default, all are fit. This behavior can be changed via the fit_options argument.

For practical reasons, the function actually returns values of -ln(L), where L is the likelihood. The logarithm of L is simpler to calculate than L itself, and the minus sign converts the problem from a maximization to a minimization, which is important because most optimizers are designed to minimize a value.

A penalty is added to the error value for any parameter combination where An is not a number, or where calculate_c4_assimilation_hyperbola produces an error.

Value

A function with one input argument guess, which should be a numeric vector representing values of the parameters to be fitted (which are specified by the fit_options input argument.) Each element of guess is the value of one parameter (arranged in alphabetical order.) For example, with the default settings, guess should contain values of c4_curvature, c4_slope, rL, and Vmax (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)
```

```
# Define an error function for one curve from the set
error_fcn <- error_function_c4_aci_hyperbola(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE]
)

# Evaluate the error for c4_curvature = 0.8, c4_slope = 0.5, rL = 1.0, Vmax = 65
error_fcn(c(0.8, 0.5, 1.0, 65))

# Make a plot of error vs. Vmax when the other parameters are fixed to
# the values above.
vmax_error_fcn <- function(Vmax) {error_fcn(c(0.8, 0.5, 1.0, Vmax))}
vmax_seq <- seq(55, 75)

lattice::xyplot(
  sapply(vmax_seq, vmax_error_fcn) ~ vmax_seq,
  type = 'b',
  xlab = 'Vmax (micromol / m^2 / s)',
  ylab = 'Negative log likelihood (dimensionless)'
)
```

`estimate_licor_variance` *Estimate variance of measured Licor values*

Description

Estimates variance and standard deviation of the net CO2 assimilation rate as measured by a Licor Li-6800 or similar portable photosynthesis system.

Usage

```
estimate_licor_variance(
  exdf_obj,
  sd_CO2_r,
  sd_CO2_s,
  sd_flow,
  sd_H2O_r,
  sd_H2O_s,
  a_column_name = 'A',
  co2_r_column_name = 'CO2_r',
  co2_s_column_name = 'CO2_s',
  corrfact_column_name = 'CorrFact',
  flow_column_name = 'Flow',
  h2o_r_column_name = 'H2O_r',
  h2o_s_column_name = 'H2O_s',
  s_column_name = 'S'
)
```

Arguments

exdf_obj	An exdf object containing gas exchange data.
sd_CO2_r	The standard deviation of reference CO2 concentrations (CO2_r) in units of micromol mol^(-1).
sd_CO2_s	The standard deviation of sample CO2 concentrations (CO2_s) in units of micromol mol^(-1).
sd_flow	The standard deviation of flow rates (Flow) in units of micromol s^(-1).
sd_H2O_r	The standard deviation of reference H2O concentrations (H2O_r) in units of mmol mol^(-1).
sd_H2O_s	The standard deviation of reference H2O concentrations (H2O_r) in units of mmol mol^(-1).
a_column_name	The name of the column in exdf_obj that contains the net CO2 assimilation rate in micromol m^(-2) s^(-1).
co2_r_column_name	The name of the column in exdf_obj that contains the CO2 concentration in the reference line in micromol mol^(-1).
co2_s_column_name	The name of the column in exdf_obj that contains the CO2 concentration in the reference line in micromol mol^(-1).
corrfact_column_name	The name of the column in exdf_obj that contains the leak correction factor (dimensionless)
flow_column_name	The name of the column in exdf_obj that contains the flow rate of air entering the leaf chamber in micromol s^(-1).
h2o_r_column_name	The name of the column in exdf_obj that contains the H2O concentration in the reference line in mmol mol^(-1).
h2o_s_column_name	The name of the column in exdf_obj that contains the H2O concentration in the sample line in mmol mol^(-1).
s_column_name	The name of the column in exdf_obj that contains the leaf chamber area in cm^2.

Details

Uses the error propogation formula to calculate the influence of the variance in CO2_r, CO2_s, etc on the variance of A, as calculated by a Licor LI-6800.

Value

An exdf object based on exdf_obj that includes additional columns representing the standard deviation of A measurements (sd_A), and the individual terms comprising the total variance of A, such as var_CO2_r, var_CO2_s, etc.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Estimate variance in measured A values
licor_file <- estimate_licor_variance(
  licor_file,
  sd_CO2_r = 1,
  sd_CO2_s = 0.1,
  sd_flow = 0.2,
  sd_H2O_r = 0.5,
  sd_H2O_s = 0.1
)

# Plot each component of the total variance of A
lattice::xyplot(
    var_CO2_r + var_CO2_s + var_flow + var_H2O_r + var_H2O_s + var_A ~ Ci | species_plot,
    data = licor_file$main_data,
    type = 'b',
    pch = 16,
    auto = TRUE
)

# Plot the standard deviation of A
lattice::xyplot(
    sd_A ~ Ci,
    group = species_plot,
    data = licor_file$main_data,
    type = 'b',
    pch = 16,
    auto = TRUE
)
```

`estimate_operating_point` *Estimate the operating point from an A-Ci curve*

Description

Uses linear interpolation to estimate Cc, Ci, and An at atmospheric CO2 concentration from the data in the exdf object, which should represent a single A-Ci curve. This function can accomodate alternative column names for the variables taken from the data file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
estimate_operating_point(
  aci_exdf,
  Ca_atmospheric,
  type = 'c3',
  a_column_name = 'A',
  ca_column_name = 'Ca',
  cc_column_name = 'Cc',
  ci_column_name = 'Ci',
  pcm_column_name = 'PCm',
  return_list = FALSE
)
```

Arguments

aci_exdf	An exdf object representing one CO2 response curve.
Ca_atmospheric	The atmospheric CO2 concentration (with units of micromol mol^(-1)); this will be used to estimate the operating point. For example, the approximate global average during the 2023 is 420 ppm, which would correspond to Ca_atmospheric = 420.
type	The type of photosynthesis: either 'c3' or 'c4'.
a_column_name	The name of the column in aci_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ca_column_name	The name of the column in aci_exdf that contains the ambient CO2 concentration in micromol mol^(-1).
cc_column_name	The name of the column in aci_exdf that contains the chloroplastic CO2 concentration in micromol mol^(-1).
ci_column_name	The name of the column in aci_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
pcm_column_name	The name of the column in aci_exdf that contains the partial pressure of CO2 in the mesophyll, expressed in microbar.
return_list	A logical value indicating whether or not to return the results as a list. Most users will only need to use return_list = TRUE; return_list = FALSE is used internally by other functions in the PhotoGEA package.

Details

When analyzing or interpreting A-Ci curves, it is often useful to determine the values of Ci and An that correspond to typical growth conditions (where Ca is set to the atmospheric value). Together, these special values of Ci and An specify the "operating point" of the leaf.

However, for a variety of practical reasons, most A-Ci curves do not actually contain a measurement point where Ca is at the atmospheric value. Nevertheless, it is possible to apply linear interpolation to the observed Ci ~ Ca and An ~ Ca relations to estimate the operating point. This function automates that procedure. It also calculates the operating values of Cc (for c3 A-Ci curves) and PCm (for c4 A-Ci curves).

This function assumes that aci_exdf represents a single A-Ci curve. Typically, this function is not directly called by users because the fitting functions fit_c3_aci and fit_c4_aci automatically use this function to determine the operating point.

Value

The return value depends on return_list and type.

When return_list is FALSE, this function returns an exdf object based on aci_exdf that includes its identifier columns as well as values of Ca_atmospheric, operating_Ci, operating_An, and operating_Cc (or operating_PCm) in columns with those names.

When return_list is TRUE, this function returns a list with the following named elements: Ca_atmospheric, operating_Ci, operating_An, operating_Cc (or operating_PCm), and operating_exdf. The first four are numeric values as described above, while operating_exdf is an exdf object with one row that can be passed to calculate_c3_assimilation or calculate_c4_assimilation in order to estimate the operating An from a photosynthesis model.

If Ca_atmospheric is outside the range of Ca values in aci_exdf, or if all provided values of Ca are NA, then the operating point cannot be reasonably estimated; in this case, an explanation is returned as the operating_point_msg column or list element, and all other calculated return values are set to NA. Otherwise, the operating_point_msg is an empty string.

If Ca_atmospheric is NA, all calculated return values are set to NA without any additional explanation.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
```

```
)

# Calculate temperature-dependent values of photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_sharkey)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate Cc, assuming an infinite mesophyll conductance (so `Cc` = `Ci`)
licor_file <- apply_gm(licor_file, Inf)

# Determine the operating point for just one curve from the data set
one_result <- estimate_operating_point(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  Ca_atmospheric = 420
)

one_result[, 'operating_Cc']
one_result[, 'operating_Ci']
one_result[, 'operating_An']
one_result[, 'operating_point_msg']
```

`example_data_files` *Example data files*

Description

The PhotoGEA package includes several data files that can be used to demonstrate different functions and analysis techniques.

Details

The following files are included with the package:

- `ball_berry_1.xlsx` and `ball_berry_2.xlsx`: Two log files created by Licor Li-6800 portable gas exchange measurement systems. These log files each contain several Ball-Berry curves. Several user constants were defined in these logs that can be used to identify individual curves or subsets of curves: `species`, `plot`, and `instrument`. These files are used in the "Analyzing Ball-Berry Data" vignette and in other examples.
- `ball_berry_1.csv`: A CSV version of `ball_berry_1.xlsx`, which was created by reading the Excel file with `read_gasex_file` and then saving it using `write.csv.exdf`. This can be done as follows: `tmp <- read_gasex_file(PhotoGEA_example_file_path('ball_berry_1.xlsx')); write.csv.exdf(tmp, 'ball_berry_1.csv')`
- `c3_aci_1.xlsx` and `c3_aci_2.xlsx`: Two log files created by Licor Li-6800 portable gas exchange measurement systems. These log files each contain several C3 CO2 response (or A-Ci) curves. Several user constants were defined in these logs that can be used to identify individual curves or subsets of curves: `species`, `plot`, and `instrument`. These files are used in the "Analyzing C3 A-Ci Curves" vignette and in other examples. The `Remarks` sheet of `c3_aci_2.xlsx` was deleted from the original version, and an updated value of Oxygen was added after observation 10, as a test for `read_licor_6800_Excel`.

- `c4_aci_1.xlsx` and `c4_aci_2.xlsx`: Two log files created by Licor Li-6800 portable gas exchange measurement systems. These log files each contain several C4 CO2 response (or A-Ci) curves. Several user constants were defined in these logs that can be used to identify individual curves or subsets of curves: `species`, `plot`, and `instrument`. These files are used in the "Analyzing C4 A-Ci Curves" vignette and in other examples.
- `tdl_sampling_1.dat` and `tdl_sampling_2.dat`: Two log files created by a Campbell Scientific CR3000 data logger, representing data from a tunable diode laser (TDL) system. These files are used in the "Analyzing TDL Data" vignette and in other examples.
- `plaintext_licor_file`: A log file created by a Licor Li-6800 portable gas exchange measurement system. This file contains several CO2 response (or A-Ci) curves. Several user constants were defined in this log that can be used to identify individual curves or subsets of curves: `species`, `plot`, and `instrument`. Several updated values of Oxygen were added as a test of `read_licor_6800_plaintext`.
- `plaintext_licor_file_v2`: A log file based on `plaintext_licor_file` that has two separate `[Data]` and `[Header]` sections, as if the log file had been closed and reopened halfway through the measurement sequence. It also has an extra blank line at the end. Several updated values of Oxygen were added as a test of `read_licor_6800_plaintext`.
- `licor_for_gm_site11.xlsx`, `licor_for_gm_site13.xslsx`, and `tdl_for_gm`: Two Licor Li-6800 log files and a CR3000 TDL log file, respectively. These files are used as an example of loading and processing combined gas exchange and isotope discrimination measurements. Each Licor log file includes 6 points measured with the `CO2_r` setpoint set to 715 ppm and 6 points with the setpoint set to 450 ppm.

Since none of these data files have been published, noise has been added to the original data. Thus, they are similar to real measurements, but no useful conclusions can be drawn from them.

After installing 'PhotoGEA', copies of these files will be stored in the R package directory (in the `PhotoGEA/extdata` subdirectory). This location will be unique to your computer, but full paths to these files can be obtained using the `PhotoGEA_example_file_path` function.

Examples

```
# Print full paths to the example files
PhotoGEA_example_file_path('ball_berry_1.xlsx')
PhotoGEA_example_file_path('ball_berry_2.xlsx')
PhotoGEA_example_file_path('c3_aci_1.xlsx')
PhotoGEA_example_file_path('c3_aci_2.xlsx')
PhotoGEA_example_file_path('c4_aci_1.xlsx')
PhotoGEA_example_file_path('c4_aci_2.xlsx')
PhotoGEA_example_file_path('licor_for_gm_site11.xlsx')
PhotoGEA_example_file_path('licor_for_gm_site13.xlsx')
PhotoGEA_example_file_path('plaintext_licor_file')
PhotoGEA_example_file_path('plaintext_licor_file_v2')
PhotoGEA_example_file_path('tdl_for_gm.dat')
PhotoGEA_example_file_path('tdl_sampling_1.dat')
PhotoGEA_example_file_path('tdl_sampling_2.dat')
```

`exclude_outliers` *Exclude outliers from a data set*

Description

Excludes outliers from a data set using the "1.5 interquartile range" rule.

Usage

```
  exclude_outliers(x, col_for_analysis, INDICES, method = 'exclude')

  ## S3 method for class 'data.frame'
exclude_outliers(x, col_for_analysis, INDICES, method = 'exclude')

  ## S3 method for class 'exdf'
exclude_outliers(x, col_for_analysis, INDICES, method = 'exclude')
```

Arguments

`x`	A data table
`col_for_analysis`	The name of a column of `x` that should be used to determine outliers.
`INDICES`	A factor or list of factors that each `nrow(x)` elements.
`method`	Specify whether to remove rows from `x` (`'remove'`) or to replace outlier values of `col_for_analysis` with `NA` (`'exclude'`).

Details

`exclude_outliers` is generic, with methods defined for data frames and `exdf` objects. This function uses a simple rule to detect outliers, where any point that deviates from the mean by more than `1.5 * IQR`, where `IQR` is the interquartile range, is said to be an outlier. This method is also sometimes referred to as "Tukey's Fences," as seen in the Wikipedia page about outliers.

For data sets with extreme outliers, it may be necessary to exclude outliers more than once to actually remove them all.

Value

This function returns an object formed from `x`, where the results depend on on the value of `method`.

When `method` is `'remove'`, the returned object is a modified copy of `x` where all rows in which the value of `col_for_analysis` is an outlier have been removed.

When `method` is `'exclude'`, the returned object is a modified copy of `x` where all outlier values of `col_for_analysis` have been replaced with `NA`.

See Also

`exdf`

Examples

```
# Read a Licor file included with the PhotoGEA package; this file includes
# several light response curves that can be identified by the 'species' and
# 'plot' columns.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Remove points from each response curve in the data where the leaf temperature
# is determined to be an outlier
licor_file_clean <- exclude_outliers(
  licor_file,
  'TleafCnd',
  list(licor_file[, 'species'], licor_file[, 'plot']),
  method = 'remove'
)

# Check to see how many points remain after removing outliers
str(list('original' = nrow(licor_file), 'clean' = nrow(licor_file_clean)))
```

---

exdf *Extended data frame*

Description

An "extended data frame" (`exdf`) is an object similar to a data frame, but which also contains information about the units and categories of each column.

Usage

```
exdf(
  main_data = data.frame(),
  units = NULL,
  categories = NULL,
  ...
)
```

Arguments

`main_data`	A data frame.
`units`	A data frame with the same columns as `main_data` (or a subset of the columns in `main_data`) but with just one row, where each entry describes the units for the corresponding column of `main_data`. If `units` is `NULL`, it will be initialized with `NA` for each column. The units of any columns in `main_data` that are not present in `units` will also be initialized to `NA`.

categories — A data frame with the same columns as main_data (or a subset of the columns in main_data) but with just one row, where each entry describes the category for the corresponding column of main_data. If categories is NULL, it will be initialized with NA for each column. The categories of any columns in main_data that are not present in catgories will also be initialized to NA.

... — Any additional properties to include as entries in the resulting exdf object; these must be passed as named arguments.

Details

The exdf class was originally created as a way to represent the contents of a Licor Excel file in an R structure. In Licor Excel files, each column has a name, units, and a category; for example, the column for values of net assimilation rate is called A, has units of micromol / m^2 / s, and is categorized as a GasEx variable.

From a technical point of view, an exdf object is simply a list with three required elements: main_data, units, and categories. Each of these should be a data frame with the same column names, as described above. It is also possible for an exdf object to have additional entries such as a filename that stores the name of the file that was used to create the exdf.

Several S3 methods have been defined for exdf objects, following the general guidance from Advanced R on S3 classes:

- is.exdf
- as.data.frame.exdf
- print.exdf
- str.exdf
- length.exdf
- dim.exdf
- dimnames.exdf
- [.exdf
- [<-.exdf
- rbind.exdf
- cbind.exdf
- split.exdf
- by.exdf

Note that the column names of main_data, units, and categories must be unique; the make.unique function can be useful for ensuring this.

Value

An exdf object as described above.

Examples

```
# Example 1: Creating a simple exdf object with two columns (`A` and `B`) and
# default values for its units and categories. There are four values of each
# variable.
exdf(data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)))

# Example 2: Creating a simple exdf object with two columns (`A` and `B`) that
# have units of `m` and `s`, respectively, and categories of `Cat1` and `Cat2`,
# respectively. There are four values of each variable.
exdf(
  data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)),
  data.frame(A = 'm', B = 's'),
  data.frame(A = 'Cat1', B = 'Cat2')
)
```

`extract.exdf` *Access or modify exdf elements*

Description

Returns or sets the values of elements in an `exdf` object.

Usage

```
  ## S3 method for class 'exdf'
x[i, j, return_exdf = FALSE]

  ## S3 replacement method for class 'exdf'
x[i, j] <- value
```

Arguments

`x`	An `exdf` object.
`i, j`	Indices specifying elements to extract or replace. Indices are `numeric` or `character` vectors or empty (missing) or `NULL`.
`return_exdf`	A logical value indicating whether the return value should be an `exdf` object.
`value`	Typically an array-like R object of a similar class as `x`.

Details

Since an `exdf` object is actually a list of named elements, those elements can be accessed using the `[[` or `$` operators, and a list of all named elements can be obtained by calling `names`.

Elements of the `main_data` data frame of an `exdf` object can be accessed and set using the `[` and `[<-` operators. When applied to an `exdf` object, these operators are essentially shortcuts to calling the same operators on the object's `main_data` data frame.

To create a new `exdf` object with a subset of the data contained in another `exdf` object, the `[` operator with `return_exdf = TRUE` can be used.

Value

When `return_exdf` is `FALSE`, the access operator will return either a vector or a data frame, depending on the dimension of `j`. When `return_exdf` is `TRUE`, the access operator will return an `exdf` object.

See Also

`exdf`

Examples

```
# Create a small exdf object that includes an extra element in addition to the
# required ones (`main_data`, `units`, and `categories`).
small_exdf <- exdf(
  data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)),
  data.frame(A = 'm', B = 's'),
  data.frame(A = 'Cat1', B = 'Cat2'),
  extra_exdf_element = "This is an example of an extra exdf element"
)

# Accessing elements of `small_exdf`
names(small_exdf)     # Get the names of all elements of small_exdf
small_exdf[['units']] # View the units using the `[[` operator
small_exdf$categories # View the categories using the `$` operator

# Accessing elements of `small_exdf$main_data`
small_exdf[,1]   # Access the first column
small_exdf[1,]   # Access the first row
small_exdf[,'B'] # Access the column named 'B'
small_exdf[1,2]  # Access element 1 of column 2

# Equivalent (but longer) commands for accessing elements of `small_exdf$main_data`
small_exdf$main_data[,1]   # Access the first column
small_exdf$main_data[1,]   # Access the first row
small_exdf$main_data[,'B'] # Access the column named 'B'
small_exdf$main_data[1,2]  # Access element 1 of column 2

# Replacing elements of `small_exdf$main_data`
small_exdf[,'A'] <- seq_len(4)             # Replace column A with new values
small_exdf[small_exdf[,'A'] > 2, 'B'] <- 0 # Replace some rows of column B with new values

# Creating a new exdf object with a subset of the data from small_exdf. Here we
# specify `return_exdf = TRUE` so that the `[` operator returns an exdf object
# instead of a data frame
new_exdf <- small_exdf[small_exdf[,'A'] > 2, , TRUE]
names(new_exdf) # Check that the `extra_exdf_element` is still present
print(new_exdf) # Check that only the rows with A > 2 are included
```

`factorize_id_column` *Convert ID column to a factor with a suitable ordering*

Description

Converts an ID column to a factor with a suitable ordering. In particular, this function will ensure that any IDs beginning with `WT` (or any other control group name, case-insensitive) will be ordered before other values. This is helpful when plotting results according to genotype.

Usage

```
  factorize_id_column(x, ...)

  ## S3 method for class 'character'
factorize_id_column(x, control_group_name = 'WT', ...)

  ## S3 method for class 'data.frame'
factorize_id_column(x, id_column_name, control_group_name = 'WT', ...)

  ## S3 method for class 'exdf'
factorize_id_column(x, id_column_name, control_group_name = 'WT', ...)
```

Arguments

`x`	Object to be ordered.
`id_column_name`	When `x` is a `data.frame` or `exdf`, this argument specifies the column within the table that should be ordered.
`control_group_name`	A string specifying the name of the control group, such as `'WT'` or `'control'`.
`...`	Additional arguments (currently unused).

Details

To choose an ordering, each unique identifier is split into three components: an initial `control_group_name` (if present), a final numeric value, and any other content in between these two. Then, the identifiers are sorted according to these three values, in order of `control_group_name` -> other content -> numeric value. Note that capitalization of any initial `control_group_name` values will be standardized to match the user-specified version.

This system works well with identifiers that represent genotypes/events, or that combine genotype/event with a replicate number.

Value

`factorize_id_column.character` returns the character vector as a `factor` with an appropriate ordering.

factorize_id_column.data.frame and factorize_id_column.exdf return a copy of the original table, where one column (specified by id_column_name) has been converted to a factor with an appropriate ordering.

See Also

exdf

Examples

```
# Identifiers that represent genotypes
genotype_ids <- c('4', 'control', '2', 'CONTROL', '8')

factorize_id_column(genotype_ids, control_group_name = 'control')

# Identifiers that represent `genotype - replicate` values
replicate_ids <- c('4 - 4', 'wT - 2', 'a - 2', 'WT - 1', '4 - 8', 'wt - 9')

factorize_id_column(replicate_ids)

# Data frame
dat <- data.frame(replicate_id = replicate_ids, val = seq_along(replicate_ids))

# Display data in bar chart - note the order of the replicates
lattice::barchart(val ~ replicate_id, data = dat)

# Display factorized data in bar chart - note the order of the replicates
lattice::barchart(val ~ replicate_id, data = factorize_id_column(dat, 'replicate_id'))

# Extended data frame
exdf_obj <- exdf(dat, units = data.frame(replicate_id = '', val = 'm / s'))

exdf_obj <- factorize_id_column(exdf_obj, 'replicate_id')

exdf_obj[, 'replicate_id']
```

---

fit_ball_berry *Fits the Ball-Berry model to an experimental curve*

Description

Calculates a linear fit of stomatal conductance vs. the Ball-Berry index using the data in the exdf object. This function can accomodate alternative column names for the variables taken from the Licor file in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
fit_ball_berry(
  replicate_exdf,
  bb_index_column_name = 'bb_index',
  gsw_column_name = 'gsw'
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one Ball-Berry curve.
`bb_index_column_name`	The name of the column in `replicate_exdf` that contains the Ball-Berry index in `mol m^(-2) s^(-1)`.
`gsw_column_name`	The name of the column in `replicate_exdf` that contains the stomatal conductance to water vapor in `mol m^(-2) s^(-1)`.

Details

The Ball-Berry model is a simple way to describe the response of a leaf's stomata to its assimilation rate and local environmental conditions. Specifically, it predicts stomatal conductance to water vapor using the following equation:

`gsw = bb_0 + bb_1 * A * h_s / C_s`

where `gsw` is the stomatal conductance, `A` is the net assimilation rate, `h_s` is the relative humidity at the leaf surface, and `C_s` is the CO2 concentration at the leaf surface. The term `A * h_s / C_s` is commonly referred to as the Ball-Berry index, while the intercept (`bb_0`) and slope (`bb_1`) of the linear relationship are the Ball-Berry parameters which describe the stomatal response.

Although this model is certainly an oversimplification, it does encode some important stomatal responses. For example, when humidity is low, the stomata close, reducing stomatal conductance. Likewise, if the CO2 concentration around the leaf is depleted, the stomata open to allow more CO2 to diffuse into the leaf's interior, increasing somatal conductance. For more information about this model and some possible alternatives, see the following papers:

- Ball, J. T., Woodrow, I. E. and Berry, J. A. "A Model Predicting Stomatal Conductance and its Contribution to the Control of Photosynthesis under Different Environmental Conditions." in "Progress in Photosynthesis Research: Volume 4" (1986) [doi:10.1007/9789401705196_48].
- Tardieu, F. and Davies, W. J. "Integration of hydraulic and chemical signalling in the control of stomatal conductance and water status of droughted plants." Plant, Cell & Environment 16, 341–349 (1993). [doi:10.1111/j.13653040.1993.tb00880.x].
- Leuning, R. "A critical appraisal of a combined stomatal-photosynthesis model for C3 plants." Plant, Cell & Environment 18, 339–355 (1995) [doi:10.1111/j.13653040.1995.tb00370.x].
- Dewar, R. C. "The Ball–Berry–Leuning and Tardieu–Davies stomatal models: synthesis and extension within a spatially aggregated picture of guard cell function." Plant, Cell & Environment 25, 1383–1398 (2002). [doi:10.1046/j.13653040.2002.00909.x].

Ball-Berry parameters are typically determined by measuring a Ball-Berry curve, where one or more of the factors that influence the Ball-Berry index is systematically varied across a range of values. At

each value, care is taken that net assimilation and stomatal conductance have reached their steady-state values, and then those values are recorded. Then, a linear fit of the experimentally observed stomatal conductances as a function of the Ball-Berry index is performed to extract estimates for the Ball-Berry intercept and slope.

This function uses `lm` to perform the fit.

This function assumes that `replicate_exdf` represents a single Ball-Berry curve. To fit multiple curves at once, this function is often used along with `by.exdf` and `consolidate`.

Value

A list with two elements:

- `fits`: An `exdf` object including the measured values and the fitted values of stomatal conductance. The fitted values will be stored in a column whose name is determined by appending `'_fits'` to the end of `gsw_column_name`; typically, this will be `'gsw_fits'`. Also includes residuals in the `gsw_residuals` column and values of the Ball-Berry slope and intercept.
- `parameters`: An `exdf` object including the fitting parameters and R-squared values. The Ball-Berry intercept is stored in the `bb_intercept` column and the Ball-Berry slope is stored in the `bb_slope` column. Their standard errors are stored in the `bb_intercept_err` and `bb_slope_err` columns. The R-squared value and p-value for the fit are stored in the `r_squared` and `p_value` columns. Other statistical descriptors of the fit as calculated by `residual_stats` are also included.

Examples

```
# Read an example Licor file included in the PhotoGEA package, calculate
# additional gas properties, calculate the Ball-Berry index, define a new column
# that uniquely identifies each curve, and then perform a fit to extract the
# Ball-Berry parameters from each curve.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_gas_properties(licor_file)

licor_file[,'species_plot'] <-
  paste(licor_file[,'species'], '-', licor_file[,'plot'])

licor_file <- calculate_ball_berry_index(licor_file)

# Fit just one curve from the data set (it is rare to do this)
one_result <- fit_ball_berry(
  licor_file[licor_file[, 'species_plot'] == 'soybean - 1a', , TRUE]
)

# Fit all curves in the data set (it is more common to do this)
bb_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
```

```
  fit_ball_berry
))

# View the fitting parameters for each species / plot
col_to_keep <- c('species', 'plot', 'species_plot', 'bb_intercept', 'bb_slope', 'r_squared')
bb_results$parameters[ , col_to_keep]

# View the fits for each species / plot
plot_ball_berry_fit(bb_results, 'species_plot')
```

`fit_c3_aci` *Fits a C3 assimilation model to an A-Ci curve*

Description

Fits the Farquhar-von-Caemmerer-Berry model to an experimentally measured C3 A-Ci curve.

It is possible to fit the following parameters: `alpha_g`, `alpha_old`, `alpha_s`, `alpha_t`, `Gamma_star_at_25`, `gmc_at_25`, `J_at_25`, `Kc_at_25`, `Ko_at_25`, `RL_at_25`, `Tp_at_25`, and `Vcmax_at_25`.

By default, only a subset of these parameters are actually fit: `alpha_old`, `J_at_25`, `RL_at_25`, `Tp_at_25`, and `Vcmax_at_25`. This can be altered using the `fit_options` argument, as described below.

Best-fit parameters are found using maximum likelihood fitting, where the optimizer (`optim_fun`) is used to minimize the error function (defined by `error_function_c3_aci`).

Once best-fit parameters are found, confidence intervals are calculated using `confidence_intervals_c3_aci`, and unreliable parameter estimates are removed.

For temperature-dependent parameters, best-fit values and confidence intervals are returned at 25 degrees C and at leaf temperature.

See below for more details.

Usage

```
fit_c3_aci(
  replicate_exdf,
  Ca_atmospheric = NA,
  a_column_name = 'A',
  ca_column_name = 'Ca',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
```

```
    vcmax_norm_column_name = 'Vcmax_norm',
    sd_A = 'RMSE',
    Wj_coef_C = 4.0,
    Wj_coef_Gamma_star = 8.0,
    optim_fun = optimizer_deoptim(200),
    lower = list(),
    upper = list(),
    fit_options = list(),
    cj_crossover_min = NA,
    cj_crossover_max = NA,
    relative_likelihood_threshold = 0.147,
    hard_constraints = 0,
    calculate_confidence_intervals = TRUE,
    remove_unreliable_param = 2,
    debug_mode = FALSE,
    ...
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
Ca_atmospheric	The atmospheric CO2 concentration (with units of micromol mol^(-1)); this will be used by estimate_operating_point to estimate the operating point. A value of NA disables this feature.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ca_column_name	The name of the column in replicate_exdf that contains the ambient CO2 concentration in micromol mol^(-1). If values of Ca are not available, they can be set to NA. In this case, it will not be possible to estimate the operating point, and apply_gm will not be able to calculate the CO2 drawdown across the stomata.
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_norm_column_name	The name of the column in replicate_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).
gmc_norm_column_name	The name of the column in replicate_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).
j_norm_column_name	The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).
kc_norm_column_name	The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).
ko_norm_column_name	The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
: The name of the column in replicate_exdf that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
: The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

sd_A
: A value of the standard deviation of measured A values, or the name of a method for determining the deviation; currently, the only supported option is 'RMSE'.

Wj_coef_C
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

Wj_coef_Gamma_star
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

optim_fun
: An optimization function that accepts the following input arguments: an initial guess, an error function, lower bounds, and upper bounds. It should return a list with the following elements: par, convergence, feval, and convergence_msg. See optimizers for a list of available options.

lower
: A list of named numeric elements representing lower bounds to use when fitting. Values supplied here override the default values (see details below). For example, lower = list(Vcmax_at_25 = 10) sets the lower limit for Vcmax_at_25 to 10 micromol / m^2 / s.

upper
: A list of named numeric elements representing upper bounds to use when fitting. Values supplied here override the default values (see details below). For example, upper = list(Vcmax_at_25 = 200) sets the upper limit for Vcmax_at_25 to 200 micromol / m^2 / s.

fit_options
: A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in replicate_exdf of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(alpha_g = 0, Vcmax_at_25 = 'fit', Tp_at_25 = 'column') means that alpha_g will be set to 0, Vcmax_at_25 will be fit, and Tp_at_25 will be set to the values in the Tp_at_25 column of replicate_exdf.

cj_crossover_min
: The minimum value of Cc (in ppm) where Aj is allowed to become the overall rate-limiting factor. If cj_crossover_min is set to NA, this restriction will not be applied.

cj_crossover_max
: The maximim value of Cc (in ppm) where Wj is allowed to be smaller than Wc. If cj_crossover_max is set to NA, this restriction will not be applied.

relative_likelihood_threshold
: To be passed to confidence_intervals_c3_aci when calculate_confidence_intervals is TRUE.

hard_constraints
: To be passed to calculate_c3_assimilation; see that function for more details.

calculate_confidence_intervals
: A logical value indicating whether or not to estimate confidence intervals for the fitting parameters using confidence_intervals_c3_aci.

remove_unreliable_param
: An integer value indicating the rules to use when identifying and removing unreliable parameter estimates. A value of 2 is the most conservative option. A value of 0 disables this feature, which is not typically recommended. It is also possible to directly specify the trust values to remove; for example, 'unreliable (process never limiting)' is equivalent to 1. See below for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about replicate_exdf, the initial guess, each guess supplied from the optimizer, and the final outcome is printed; this can be helpful when troubleshooting issues with a particular curve.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

This function calls apply_gm and calculate_c3_assimilation to calculate values of net assimilation. The user-supplied optimization function is used to vary the values of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, gmc_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 to find ones that best reproduce the experimentally measured values of net assimilation. By default, the following options are used for the fits:

- alpha_g: lower = 0, upper = 10, fit_option = 0
- alpha_old: lower = 0, upper = 10, fit_option = 'fit'
- alpha_s: lower = 0, upper = 10, fit_option = 0
- alpha_t: lower = 0, upper = 10, fit_option = 0
- Gamma_star_at_25: lower = -20, upper = 200, fit_option = 'column'
- gmc_at_25: lower = -1, upper = 10, fit_option = Inf
- J_at_25: lower = -50, upper = 1000, fit_option = 'fit'
- Kc_at_25: lower = -50, upper = 1000, fit_option = 'column'
- Ko_at_25: lower = -50, upper = 1000, fit_option = 'column'

- RL_at_25: lower = -10, upper = 100, fit_option = 'fit'
- Tp_at_25: lower = -10, upper = 100, fit_option = 'fit'
- Vcmax_at_25: lower = -50, upper = 1000, fit_option = 'fit'

With these settings, the "new" alpha parameters are set to 0; values of Gamma_star_at_25, Kc_at_25, and Ko_at_25 are taken from the Gamma_star_at_25, Kc_at_25, and Ko_at_25 columns of replicate_exdf; mesophyll conductance (gmc_at_25) is set to inifinity (so Cc = Ci); and the other parameters are fit during the process (see fit_options above). The bounds are chosen liberally to avoid any bias.

An initial guess for the parameters is generated by calling initial_guess_c3_aci as follows:

- cc_threshold_rl is set to 100 micromol / mol.
- If alpha_g is being fit, the alpha_g argument of initial_guess_c3_aci is set to 0.5; otherwise, the argument is set to the value specified by the fit options.
- If alpha_old is being fit, the alpha_old argument of initial_guess_c3_aci is set to 0.5; otherwise, the argument is set to the value specified by the fit options.
- if alpha_s is being fit, the alpha_s argument of initial_guess_c3_aci is set to 0.3 * (1 - alpha_g); otherwise, the argument is set to the value specified by the fit options.
- if alpha_t is being fit, the alpha_t argument of initial_guess_c3_aci is set to 0; otherwise, the argument is set to the value specified by the fit options.
- If Gamma_star_at_25 is being fit, the Gamma_star_at_25 argument of initial_guess_c3_aci is set to 40; otherwise, the argument is set to the value specified by the fit options.
- If gmc_at_25 is being fit, the gmc_at_25 argument of initial_guess_c3_aci is set to 1; otherwise, the argument is set to the value specified by the fit options.
- If Kc_at_25 is being fit, the Kc_at_25 argument of initial_guess_c3_aci is set to 400; otherwise, the argument is set to the value specified by the fit options.
- If Ko_at_25 is being fit, the Ko_at_25 argument of initial_guess_c3_aci is set to 275; otherwise, the argument is set to the value specified by the fit options.

Note that any fixed values specified in the fit options will override the values returned by the guessing function.

The fit is made by creating an error function using error_function_c3_aci and minimizing its value using optim_fun, starting from the initial guess described above. The optimizer_deoptim optimizer is used by default since it has been found to reliably return great fits. However, it is a slow optimizer. If speed is important, consider reducing the number of generations or using optimizer_nmkb, but be aware that this optimizer is more likely to get stuck in a local minimum.

The photosynthesis model represented by calculate_c3_assimilation is not smooth in the sense that small changes in the input parameters do not necessarily cause changes in its outputs. This is related to the final step in the calculations, where the overall assimilation rate is taken to be the minimum of three enzyme-limited rates. For example, if the assimilation rate is never TPU-limited, modifying Tp_at_25 will not change the model's outputs. For this reason, derivative-based optimizers tend to struggle when fitting C3 A-Ci curves. Best results are obtained using derivative-free methods.

Sometimes the optimizer may choose a set of parameter values where one or more of the potential limiting carboxylation rates (Wc, Wj, or Wp) is never the smallest rate. In this case, the corresponding parameter estimates (Vcmax, J, or alpha_old & Tp) will be severely unreliable. This will be indicated by a value of 'unreliable (process never limiting)' in the corresponding trust column

(for example, Vcmax_trust). If remove_unreliable_param is 1 or larger, then such parameter estimates (and the corresponding rates) will be replaced by NA in the fitting results.

It is also possible that the upper limit of the confidence interval for a parameter is infinity; this indicates a potentially unreliable parameter estimate. This will be indicated by a value of 'unreliable (infinite upper limit)' in the corresponding trust column (for example, Vcmax_trust). If remove_unreliable_param is 2 or larger, then such parameter estimates (but not the corresponding rates) will be replaced by NA in the fitting results.

The trust value for fully reliable parameter estimates is set to 'reliable' and they will never be replaced by NA.

Once the best-fit parameters have been determined, this function also estimates the operating value of Cc from the atmospheric CO2 concentration atmospheric_ca using estimate_operating_point, and then uses that value to estimate the modeled An at the operating point via calculate_c3_assimilation. It also estimates the Akaike information criterion (AIC).

This function assumes that replicate_exdf represents a single C3 A-Ci curve. To fit multiple curves at once, this function is often used along with by.exdf and consolidate.

Value

A list with three elements:

- fits: An exdf object including the original contents of replicate_exdf along with several new columns:
  - The fitted values of net assimilation will be stored in a column whose name is determined by appending '_fit' to the end of a_column_name; typically, this will be 'A_fit'.
  - Residuals (measured - fitted) will be stored in a column whose name is determined by appending '_residuals' to the end of a_column_name; typically, this will be 'A_residuals'.
  - Values of fitting parameters at 25 degrees C will be stored in the Gamma_star_at_25, gmc_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 columns.
  - The other outputs from calculate_c3_assimilation will be stored in columns with the usual names: alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_tl, gmc_tl, Kc_tl, Ko_tl, Tp_tl, Vcmax_tl, RL_tl, J_tl, Wc, Wj, Wp, Vc, Ac, Aj, and Ap.
- fits_interpolated: An exdf object including the calculated assimilation rates at a fine spacing of Ci values (step size of 1 micromol mol^(-1)).
- parameters: An exdf object including the identifiers, fitting parameters, and convergence information for the A-Ci curve:
  - The number of points where An = Ac, An = Aj, and An = Ap are stored in the n_Ac_limiting, n_Aj_limiting, and n_Ap_limiting columns.
  - The best-fit values are stored in the alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, gmc_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 columns. If calculate_confidence_intervals is TRUE, upper and lower limits for each of these parameters will also be included.
  - For parameters that depend on leaf temperature, the average leaf-temperature-dependent values are stored in Gamma_star_tl_avg, gmc_tl_avg, J_tl_avg, Kc_tl_avg, Ko_tl_avg, RL_tl_avg, Tp_tl_avg, and Vcmax_tl_avg.

- Information about the operating point is stored in `operating_Cc`, `operating_Ci`, `operating_An`, and `operating_An_model`.
- The `convergence` column indicates whether the fit was successful (`==0`) or if the optimizer encountered a problem (`!=0`).
- The `feval` column indicates how many cost function evaluations were required while finding the optimal parameter values.
- The residual stats as returned by `residual_stats` are included as columns with the default names: `dof`, `RSS`, `RMSE`, etc.
- The Akaike information criterion is included in the `AIC` column.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)

# We can fit just one curve from the data set, although it is rare to do this
one_result <- fit_c3_aci(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  Ca_atmospheric = 420,
  optim_fun = optimizer
)

# We can fit the same curve, but allow alpha_old and Gamma_star_at_25 to vary
one_result_v2 <- fit_c3_aci(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  Ca_atmospheric = 420,
  fit_options = list(Gamma_star_at_25 = 'fit', alpha_old = 'fit'),
  optim_fun = optimizer
```

```
)

# Fit all curves in the data set (it is more common to do this)
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer
))

# View the fitting parameters for each species / plot
col_to_keep <- c(
  'species', 'plot',                                          # identifiers
  'n_Ac_limiting', 'n_Aj_limiting', 'n_Ap_limiting',       # number of points where
                                                              #   each process is limiting
 'Tp_at_25', 'J_at_25', 'RL_at_25', 'Vcmax_at_25',      # parameters scaled to 25 degrees C
 'J_tl_avg', 'RL_tl_avg', 'Vcmax_tl_avg',            # average temperature-dependent values
  'operating_Ci', 'operating_An', 'operating_An_model',    # operating point info
  'dof', 'RSS', 'MSE', 'RMSE', 'RSE',                      # residual stats
  'convergence', 'convergence_msg', 'feval', 'optimum_val' # convergence info
)

aci_results$parameters[ , col_to_keep, TRUE]

# View the fits for each species / plot
plot_c3_aci_fit(aci_results, 'species_plot', 'Ci')

# View the residuals for each species / plot
lattice::xyplot(
  A_residuals ~ Ci | species_plot,
  data = aci_results$fits$main_data,
  type = 'b',
  pch = 16,
  auto = TRUE,
  grid = TRUE,
  xlab = paste0('Intercellular CO2 concentration (', aci_results$fits$units$Ci, ')'),
  ylab = paste0('Assimilation rate residuals (', aci_results$fits$units$A_residuals, ')')
)

# In some of the curves above, there are no points where carboxylation is TPU
# limited. Estimates of Tp are therefore unreliable and are removed.
```

`fit_c3_variable_j` *Fits a C3 assimilation model to an A-Ci + CF curve*

Description

Fits the Farquhar-von-Caemmerer-Berry + Variable J model to an experimentally measured C3 A-Ci + CF curve.

It is possible to fit the following parameters: alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, J_at_25, Kc_at_25, Ko_at_25 RL_at_25, tau, Tp_at_25, and Vcmax_at_25.

By default, only a subset of these parameters are actually fit: alpha_old, J_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25. This can be altered using the fit_options argument, as described below.

Best-fit parameters are found using maximum likelihood fitting, where the optimizer (optim_fun) is used to minimize the error function (defined by error_function_c3_variable_j).

Once best-fit parameters are found, confidence intervals are calculated using confidence_intervals_c3_variable_j, and unreliable parameter estimates are removed.

For temperature-dependent parameters, best-fit values and confidence intervals are returned at 25 degrees C and at leaf temperature.

See below for more details.

Usage

```
fit_c3_variable_j(
  replicate_exdf,
  Ca_atmospheric = NA,
  a_column_name = 'A',
  ca_column_name = 'Ca',
  ci_column_name = 'Ci',
  etr_column_name = 'ETR',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  phips2_column_name = 'PhiPS2',
  qin_column_name = 'Qin',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  sd_A = 'RMSE',
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  optim_fun = optimizer_deoptim(400),
  lower = list(),
  upper = list(),
  fit_options = list(),
  cj_crossover_min = NA,
  cj_crossover_max = NA,
  require_positive_gmc = 'positive_a',
  gmc_max = Inf,
  check_j = TRUE,
  relative_likelihood_threshold = 0.147,
  hard_constraints = 0,
```

```
  calculate_confidence_intervals = TRUE,
  remove_unreliable_param = 2,
  debug_mode = FALSE,
  ...
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
Ca_atmospheric	The atmospheric CO2 concentration (with units of micromol mol^(-1)); this will be used by estimate_operating_point to estimate the operating point. A value of NA disables this feature.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ca_column_name	The name of the column in replicate_exdf that contains the ambient CO2 concentration in micromol mol^(-1). If values of Ca are not available, they can be set to NA. In this case, it will not be possible to estimate the operating point, and apply_gm will not be able to calculate the CO2 drawdown across the stomata.
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
etr_column_name	The name of the column in rc_exdf that contains the electron transport rate as estimated by the measurement system in micromol m^(-2) s^(-1).
gamma_star_norm_column_name	The name of the column in replicate_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).
j_norm_column_name	The name of the column in replicate_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).
kc_norm_column_name	The name of the column in replicate_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).
ko_norm_column_name	The name of the column in replicate_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).
oxygen_column_name	The name of the column in replicate_exdf that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.
phips2_column_name	The name of the column in replicate_exdf that contains values of the operating efficiency of photosystem II (dimensionless).
qin_column_name	The name of the column in replicate_exdf that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).

rl_norm_column_name
: The name of the column in replicate_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in replicate_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in replicate_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

sd_A
: A value of the standard deviation of measured A values, or the name of a method for determining the deviation; currently, the only supported option is 'RMSE'.

Wj_coef_C
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

Wj_coef_Gamma_star
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

optim_fun
: An optimization function that accepts the following input arguments: an initial guess, an error function, lower bounds, and upper bounds. It should return a list with the following elements: par, convergence, feval, and convergence_msg. The default option is an evolutionary optimizer that runs slow but tends to find good fits for most curves. optimizer_nmkb can also be used; it is faster, but doesn't always find a good fit.

lower
: A list of named numeric elements representing lower bounds to use when fitting. Values supplied here override the default values (see details below). For example, lower = list(Vcmax_at_25 = 10) sets the lower limit for Vcmax_at_25 to 10 micromol / m^2 / s.

upper
: A list of named numeric elements representing upper bounds to use when fitting. Values supplied here override the default values (see details below). For example, upper = list(Vcmax_at_25 = 200) sets the upper limit for Vcmax_at_25 to 200 micromol / m^2 / s.

fit_options
: A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in replicate_exdf of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(alpha_g = 0, Vcmax_at_25 = 'fit', Tp_at_25 = 'column') means that alpha_g will be set to 0, Vcmax_at_25 will be fit, and Tp_at_25 will be set to the values in the Tp_at_25 column of replicate_exdf.

cj_crossover_min
: To be passed to error_function_c3_variable_j.

cj_crossover_max
: To be passed to error_function_c3_variable_j.

require_positive_gmc
: To be passed to error_function_c3_variable_j.

gmc_max
: To be passed to error_function_c3_variable_j.

check_j
: To be passed to error_function_c3_variable_j.

relative_likelihood_threshold
: To be passed to confidence_intervals_c3_variable_j when calculate_confidence_intervals is TRUE.

hard_constraints
: To be passed to calculate_c3_assimilation and calculate_c3_variable_j; see those functions for more details.

calculate_confidence_intervals
: A logical value indicating whether or not to estimate confidence intervals for the fitting parameters using confidence_intervals_c3_variable_j.

remove_unreliable_param
: An integer value indicating the rules to use when identifying and removing unreliable parameter estimates. A value of 2 is the most conservative option. A value of 0 disables this feature, which is not typically recommended. It is also possible to directly specify the trust values to remove; for example, 'unreliable (process never limiting)' is equivalent to 1. See below for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about replicate_exdf, the initial guess, each guess supplied from the optimizer, and the final outcome is printed; this can be helpful when troubleshooting issues with a particular curve.

...
: Additional arguments to be passed to calculate_c3_assimilation.

Details

This function calls calculate_c3_variable_j and calculate_c3_assimilation to calculate values of net assimilation. The user-supplied optimization function is used to vary the values of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 to find ones that best reproduce the experimentally measured values of net assimilation. By default, the following options are used for the fits:

- alpha_g: lower = 0, upper = 10, fit_option = 0
- alpha_old: lower = 0, upper = 10, fit_option = 'fit'
- alpha_s: lower = 0, upper = 10, fit_option = 0
- alpha_t: lower = 0, upper = 10, fit_option = 0
- Gamma_star_at_25: lower = -20, upper = 200, fit_option = 'column'
- J_at_25: lower = -50, upper = 1000, fit_option = 'fit'
- Kc_at_25: lower = -50, upper = 1000, fit_option = 'column'
- Ko_at_25: lower = -50, upper = 1000, fit_option = 'column'
- RL_at_25: lower = -10, upper = 100, fit_option = 'fit'
- tau: lower = -10, upper = 10, fit_option = 'fit'

- Tp_at_25: lower = -10, upper = 100, fit_option = 'fit'
- Vcmax_at_25: lower = -50, upper = 1000, fit_option = 'fit'

With these settings, all "new" alpha parameters are set to 0; values of Gamma_star_at_25, Kc_at_25, and Ko_at_25 are taken from the Gamma_star_at_25, Kc_at_25, and Ko_at_25 columns of replicate_exdf; and the other parameters are fit during the process (see fit_options above). The bounds are chosen liberally to avoid any bias.

An initial guess for the parameters is generated by calling initial_guess_c3_variable_j as follows:

- cc_threshold_rl is set to 100 micromol / mol.
- If alpha_g is being fit, the alpha_g argument of initial_guess_c3_variable_j is set to 0.5; otherwise, the argument is set to the value specified by the fit options.
- If alpha_old is being fit, the alpha_old argument of initial_guess_c3_variable_j is set to 0.5; otherwise, the argument is set to the value specified by the fit options.
- if alpha_s is being fit, the alpha_s argument of initial_guess_c3_variable_j is set to 0.3 * (1 - alpha_g); otherwise, the argument is set to the value specified by the fit options.
- if alpha_t is being fit, the alpha_t argument of initial_guess_c3_variable_j is set to 0; otherwise, the argument is set to the value specified by the fit options.
- If Gamma_star_at_25 is being fit, the Gamma_star_at_25 argument of initial_guess_c3_variable_j is set to 40; otherwise, the argument is set to the value specified by the fit options.
- If Kc_at_25 is being fit, the Kc_at_25 argument of initial_guess_c3_variable_j is set to 400; otherwise, the argument is set to the value specified by the fit options.
- If Ko_at_25 is being fit, the Ko_at_25 argument of initial_guess_c3_variable_j is set to 275; otherwise, the argument is set to the value specified by the fit options.

Note that any fixed values specified in the fit options will override the values returned by the guessing function.

The fit is made by creating an error function using error_function_c3_variable_j and minimizing its value using optim_fun, starting from the initial guess described above. The optimizer_deoptim optimizer is used by default since it has been found to reliably return great fits. However, it is a slow optimizer. If speed is important, consider reducing the number of generations or using optimizer_nmkb, but be aware that this optimizer is more likely to get stuck in a local minimum.

The photosynthesis model used here is not smooth in the sense that small changes in the input parameters do not necessarily cause changes in its outputs. This is related to the final step in the calculations, where the overall assimilation rate is taken to be the minimum of three enzyme-limited rates. For example, if the assimilation rate is never phosphate-limited, modifying Tp_at_25 will not change the model's outputs. For this reason, derivative-based optimizers tend to struggle when fitting C3 A-Ci curves. Best results are obtained using derivative-free methods.

Sometimes the optimizer may choose a set of parameter values where one or more of the potential limiting carboxylation rates (Wc, Wj, or Wp) is never the smallest rate. In this case, the corresponding parameter estimates (Vcmax, J, or alpha_old & Tp) will be severely unreliable. This will be indicated by a value of 'unreliable (process never limiting)' in the corresponding trust column (for example, Vcmax_trust). If remove_unreliable_param is 1 or larger, then such parameter estimates (and the corresponding rates) will be replaced by NA in the fitting results.

It is also possible that the upper limit of the confidence interval for a parameter is infinity; this indicates a potentially unreliable parameter estimate. This will be indicated by a value of 'unreliable (infinite upper limit)' in the corresponding trust column (for example, Vcmax_trust). If remove_unreliable_param is 2 or larger, then such parameter estimates (but not the corresponding rates) will be replaced by NA in the fitting results.

The trust value for fully reliable parameter estimates is set to 'reliable' and they will never be replaced by NA.

Once the best-fit parameters have been determined, this function also estimates the operating value of 'Cc from the atmospheric CO2 concentration atmospheric_ca using estimate_operating_point, and then uses that value to estimate the modeled An at the operating point via calculate_c3_assimilation. It also estimates the Akaike information criterion (AIC).

This function assumes that replicate_exdf represents a single C3 A-Ci curve. To fit multiple curves at once, this function is often used along with by.exdf and consolidate.

Value

A list with two elements:

- fits: An exdf object including the original contents of replicate_exdf along with several new columns:
  - The fitted values of net assimilation will be stored in a column whose name is determined by appending '_fit' to the end of a_column_name; typically, this will be 'A_fit'.
  - Residuals (measured - fitted) will be stored in a column whose name is determined by appending '_residuals' to the end of a_column_name; typically, this will be 'A_residuals'.
  - Values of fitting parameters at 25 degrees C will be stored in the Gamma_star_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 columns.
  - The other outputs from calculate_c3_variable_j and calculate_c3_assimilation will be stored in columns with the usual names: alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_tl, J_tl, Kc_tl, Ko_tl, RL_tl, tau, Tp_tl, Vcmax_tl, Ac, Aj, Ap, gmc, J_F, and Cc.
- fits_interpolated: An exdf object including the calculated assimilation rates at a fine spacing of Ci values (step size of 1 micromol mol^(-1)).
- parameters: An exdf object including the identifiers, fitting parameters, and convergence information for the A-Ci curve:
  - The number of points where An = Ac, An = Aj, and An = Ap are stored in the n_Ac_limiting, n_Aj_limiting, and n_Ap_limiting columns.
  - The best-fit values are stored in the alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 columns. If calculate_confidence_intervals is TRUE, upper and lower limits for each of these parameters will also be included.
  - For parameters that depend on leaf temperature, the average leaf-temperature-dependent values are stored in Gamma_star_tl_avg, J_tl_avg, Kc_tl_avg, Ko_tl_avg, RL_tl_avg, Tp_tl_avg, and Vcmax_tl_avg.
  - Information about the operating point is stored in operating_Cc, operating_Ci, operating_An, and operating_An_model.

- The convergence column indicates whether the fit was successful (==0) or if the optimizer encountered a problem (!=0).
- The feval column indicates how many cost function evaluations were required while finding the optimal parameter values.
- The residual stats as returned by residual_stats are included as columns with the default names: dof, RSS, RMSE, etc.
- The Akaike information criterion is included in the AIC column.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)

# Fit just one curve from the data set (it is rare to do this).
one_result <- fit_c3_variable_j(
  licor_file[licor_file[, 'species_plot'] == 'tobacco - 1', , TRUE],
  Ca_atmospheric = 420,
  optim_fun = optimizer
)

# Fit all curves in the data set (it is more common to do this).
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_variable_j,
  Ca_atmospheric = 420,
  optim_fun = optimizer
))
```

```
# View the fitting parameters for each species / plot
col_to_keep <- c(
  'species', 'plot',                                        # identifiers
  'n_Ac_limiting', 'n_Aj_limiting', 'n_Ap_limiting',         # number of points where
                                                            #   each process is limiting
  'tau', 'Tp_at_25',                                  # parameters with temperature response
  'J_at_25', 'RL_at_25', 'Vcmax_at_25',                  # parameters scaled to 25 degrees C
  'J_tl_avg', 'RL_tl_avg', 'Vcmax_tl_avg',             # average temperature-dependent values
  'operating_Ci', 'operating_An', 'operating_An_model',     # operating point info
  'dof', 'RSS', 'MSE', 'RMSE', 'RSE',                       # residual stats
  'convergence', 'convergence_msg', 'feval', 'optimum_val' # convergence info
)

aci_results$parameters[ , col_to_keep, TRUE]

# View the fits for each species / plot
plot_c3_aci_fit(aci_results, 'species_plot', 'Ci')

# View the residuals for each species / plot
lattice::xyplot(
  A_residuals ~ Ci | species_plot,
  data = aci_results$fits$main_data,
  type = 'b',
  pch = 16,
  auto = TRUE,
  grid = TRUE,
  xlab = paste0('Intercellular CO2 concentration (', aci_results$fits$units$Ci, ')'),
  ylab = paste0('Assimilation rate residuals (', aci_results$fits$units$A_residuals, ')')
)

# View the estimated mesophyll conductance values for each species / plot
lattice::xyplot(
  gmc ~ Ci | species_plot,
  data = aci_results$fits$main_data,
  type = 'b',
  pch = 16,
  auto = TRUE,
  grid = TRUE,
  xlab = paste0('Intercellular CO2 concentration (', aci_results$fits$units$Ci, ')'),
  ylab = paste0('Mesophyll conductance to CO2 (', aci_results$fits$units$gmc, ')'),
  ylim = c(0, 2)
)

# In some of the curves above, there are no points where carboxylation is TPU
# limited. Estimates of Tp are therefore unreliable and are removed.
```

`fit_c4_aci` *Fits a C4 assimilation model to an A-Ci curve*

Description

Fits the von Caemmerer model to an experimentally measured C4 A-Ci curve.

It is possible to fit the following parameters: `alpha_psii`, `gbs`, `gmc_at_25`, `J_at_25`, `RL_at_25`, `Rm_frac`, `Vcmax_at_25`, `Vpmax_at_25`, and `Vpr`.

By default, only a subset of these parameters are actually fit: `RL_at_25`, `Vcmax_at_25`, and `Vpmax_at_25`. This can be altered using the `fit_options` argument, as described below.

Best-fit parameters are found using maximum likelihood fitting, where the optimizer (`optim_fun`) is used to minimize the error function (defined by `error_function_c4_aci`).

Once best-fit parameters are found, confidence intervals are calculated using `confidence_intervals_c4_aci`, and unreliable parameter estimates are removed.

For temperature-dependent parameters, best-fit values and confidence intervals are returned at 25 degrees C and at leaf temperature.

See below for more details.

Usage

```
fit_c4_aci(
  replicate_exdf,
  Ca_atmospheric = NA,
  ao_column_name = 'ao',
  a_column_name = 'A',
  ca_column_name = 'Ca',
  ci_column_name = 'Ci',
  gamma_star_column_name = 'gamma_star',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kc_column_name = 'Kc',
  ko_column_name = 'Ko',
  kp_column_name = 'Kp',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  vcmax_norm_column_name = 'Vcmax_norm',
  vpmax_norm_column_name = 'Vpmax_norm',
  sd_A = 'RMSE',
  x_etr = 0.4,
  optim_fun = optimizer_deoptim(200),
  lower = list(),
  upper = list(),
  fit_options = list(),
  relative_likelihood_threshold = 0.147,
  hard_constraints = 0,
  calculate_confidence_intervals = TRUE,
  remove_unreliable_param = 2,
  debug_mode = FALSE
)
```

Arguments

replicate_exdf	An exdf object representing one CO2 response curve.
Ca_atmospheric	The atmospheric CO2 concentration (with units of micromol mol^(-1)); this will be used by estimate_operating_point to estimate the operating point. A value of NA disables this feature.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
ao_column_name	The name of the column in exdf_obj that contains the dimensionless ratio of solubility and diffusivity of O2 to CO2.
ca_column_name	The name of the column in replicate_exdf that contains the ambient CO2 concentration in micromol mol^(-1). If values of Ca are not available, they can be set to NA. In this case, it will not be possible to estimate the operating point, and apply_gm will not be able to calculate the CO2 drawdown across the stomata.
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).
gamma_star_column_name	The name of the column in exdf_obj that contains the dimensionless gamma_star values.
gmc_norm_column_name	The name of the column in replicate_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).
j_norm_column_name	The name of the column in exdf_obj that contains the normalized J values (with units of normalized to J at 25 degrees C).
kc_column_name	The name of the column in exdf_obj that contains the Michaelis-Menten constant for rubisco carboxylation in microbar.
ko_column_name	The name of the column in exdf_obj that contains the Michaelis-Menten constant for rubisco oxygenation in mbar.
kp_column_name	The name of the column in exdf_obj that contains the Michaelis-Menten constant for PEP carboxylase carboxylation in microbar.
oxygen_column_name	The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.
rl_norm_column_name	The name of the column in exdf_obj that contains the normalized RL values (with units of normalized to RL at 25 degrees C).
total_pressure_column_name	The name of the column in exdf_obj that contains the total pressure in bar.
vcmax_norm_column_name	The name of the column in exdf_obj that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

vpmax_norm_column_name
: The name of the column in exdf_obj that contains the normalized Vpmax values (with units of normalized to Vpmax at 25 degrees C).

sd_A
: A value of the standard deviation of measured A values, or the name of a method for determining the deviation; currently, the only supported option is 'RMSE'.

x_etr
: The fraction of whole-chain electron transport occurring in the mesophyll (dimensionless). See Equation 29 from S. von Caemmerer (2021).

optim_fun
: An optimization function that accepts the following input arguments: an initial guess, an error function, lower bounds, and upper bounds. It should return a list with the following elements: par, convergence, feval, and convergence_msg. See optimizers for a list of available options.

lower
: A list of named numeric elements representing lower bounds to use when fitting. Values supplied here override the default values (see details below). For example, lower = list(Vcmax_at_25 = 10) sets the lower limit for Vcmax_at_25 to 10 micromol / m^2 / s.

upper
: A list of named numeric elements representing upper bounds to use when fitting. Values supplied here override the default values (see details below). For example, upper = list(Vcmax_at_25 = 200) sets the upper limit for Vcmax_at_25 to 200 micromol / m^2 / s.

fit_options
: A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be 'fit', 'column', or a numeric value. A value of 'fit' means that the parameter will be fit; a value of 'column' means that the value of the parameter will be taken from a column in exdf_obj of the same name; and a numeric value means that the parameter will be set to that value. For example, fit_options = list(RL_at_25 = 0, Vcmax_at_25 = 'fit', Vpr = 'column') means that RL_at_25 will be set to 0, Vcmax_at_25 will be fit, and Vpr will be set to the values in the Vpr column of exdf_obj.

relative_likelihood_threshold
: To be passed to confidence_intervals_c4_aci when calculate_confidence_intervals is TRUE.

hard_constraints
: To be passed to calculate_c4_assimilation; see that function for more details.

calculate_confidence_intervals
: A logical value indicating whether or not to estimate confidence intervals for the fitting parameters using confidence_intervals_c4_aci.

remove_unreliable_param
: An integer value indicating the rules to use when identifying and removing unreliable parameter estimates. A value of 2 is the most conservative option. A value of 0 disables this feature, which is not typically recommended. It is also possible to directly specify the trust values to remove; for example, 'unreliable (process never limiting)' is equivalent to 1. See below for more details.

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about replicate_exdf, the initial guess, each guess supplied from the optimizer, and the final outcome is printed; this can be helpful when troubleshooting issues with a particular curve.

Details

This function calls calculate_c4_assimilation to calculate values of net assimilation. The user-supplied optimization function is used to vary the values of alpha_psii, gbs, gmc_at_25, J_at_25, RL_at_25, Rm_frac, Vcmax_at_25, Vpmax_at_25, and Vpr to find ones that best reproduce the experimentally measured values of net assimilation. By default, the following options are used for the fits:

- alpha_psii: lower = -1, upper = 10, fit_option = 0
- gbs: lower = -1, upper = 10, fit_option = 0.003
- gmc_at_25: lower = -1, upper = 10, fit_option = 1
- J_at_25: lower = -50, upper = 1000, fit_option = 1000
- RL_at_25: lower = -10, upper = 100, fit_option = 'fit'
- Rm_frac: lower = -10, upper = 10, fit_option = 0.5
- Vcmax_at_25: lower = -50, upper = 1000, fit_option = 'fit'
- Vpmax_at_25: lower = -50, upper = 1000, fit_option = 'fit'
- Vpr: lower = -50, upper = 1000, fit_option = 1000

With these settings, J_at_25 and Vpr are set to 1000 (so net assimilation is essentially never limited by light or PEP carboxylase regeneration), alpha_psii, gbs, gmc_at_25, and Rm_frac are set to default values used in von Caemmerer (2000), and the other parameters are fit during the process (see fit_options above). The bounds are chosen liberally to avoid any bias.

An initial guess for the parameters is generated by calling initial_guess_c4_aci as follows:

- pcm_threshold_rlm is set to 40 microbar.
- If alpha_psii is being fit, the alpha_psii argument of initial_guess_c4_aci is set to 0.1; otherwise, the argument is set to the value specified by the fit options.
- If gbs is being fit, the gbs argument of initial_guess_c4_aci is set to 0.003; otherwise, the argument is set to the value specified by the fit options.
- If gmc_at_25 is being fit, the gmc_at_25 argument of initial_guess_c4_aci is set to 1; otherwise, the argument is set to the value specified by the fit options.
- If Rm_frac is being fit, the Rm_frac argument of initial_guess_c4_aci is set to 0.5; otherwise, the argument is set to the value specified by the fit options.

Note that any fixed values specified in the fit options will override the values returned by the guessing function.

The fit is made by creating an error function using error_function_c4_aci and minimizing its value using optim_fun, starting from the initial guess described above. The optimizer_deoptim optimizer is used by default since it has been found to reliably return great fits. However, it is a slow optimizer. If speed is important, consider reducing the number of generations or using optimizer_nmkb, but be aware that this optimizer is more likely to get stuck in a local minimum.

The photosynthesis model represented by calculate_c4_assimilation is not smooth in the sense that small changes in the input parameters do not necessarily cause changes in its outputs. This is related to the calculation of the PEP carboxylase activity Vp, which is taken to be the minimum of Vpr and Vpc. For example, if Vpr is high and Vp = Vpc at all points along the curve, modifying Vpr by a small amount will not change the model's outputs. Similar issues can occur when calculating

An as the minimum of Ac and Aj. Because of this, derivative-based optimizers tend to struggle when fitting C4 A-Ci curves. Best results are obtained using derivative-free methods.

Sometimes the optimizer may choose a set of parameter values where one of the potential limiting rates Vpc or Vpr is never the smallest rate. In this case, the corresponding parameter estimates (Vpmax or Vpr) will be severely unreliable. Likewise, it may happen that one of Ac or Aj is never the smallest rate. In this case the corresponding parameter estimates (Vpmax, Vpr, and Vcmax, or J) will be severely unreliable. This will be indicated by a value of 'unreliable (process never limiting)' in the corresponding trust column (for example, Vcmax_trust). If remove_unreliable_param is 1 or larger, then such parameter estimates (and the corresponding rates) will be replaced by NA in the fitting results.

It is also possible that the upper limit of the confidence interval for a parameter is infinity; this indicates a potentially unreliable parameter estimate. This will be indicated by a value of 'unreliable (infinite upper limit)' in the corresponding trust column (for example, Vcmax_trust). If remove_unreliable_param is 2 or larger, then such parameter estimates (but not the corresponding rates) will be replaced by NA in the fitting results.

The trust value for fully reliable parameter estimates is set to 'reliable' and they will never be replaced by NA.

Once the best-fit parameters have been determined, this function also estimates the operating value of 'PCm from the atmospheric CO2 concentration atmospheric_ca using estimate_operating_point, and then uses that value to estimate the modeled An at the operating point via calculate_c4_assimilation. It also estimates the Akaike information criterion (AIC).

This function assumes that replicate_exdf represents a single C4 A-Ci curve. To fit multiple curves at once, this function is often used along with by.exdf and consolidate.

Value

A list with two elements:

- fits: An exdf object including the original contents of replicate_exdf along with several new columns:
  - The fitted values of net assimilation will be stored in a column whose name is determined by appending '_fit' to the end of a_column_name; typically, this will be 'A_fit'.
  - Residuals (measured - fitted) will be stored in a column whose name is determined by appending '_residuals' to the end of a_column_name; typically, this will be 'A_residuals'.
  - Values of fitting parameters at 25 degrees C will be stored in the gmc_at_25, J_at_25, RL_at_25, Vcmax_at_25, Vpmax_at_25, and Vpr columns.
  - The other outputs from calculate_c4_assimilation will be stored in columns with the usual names: alpha_psii, gbs, gmc_tl, Rm_Frac, Vcmax_tl, Vpmax_tl, RL_tl, RLm_tl, Vp, Apc, Apr, Ap, Ar, Ajm, Ajbs, Ac, and Aj.
- fits_interpolated: An exdf object including the calculated assimilation rates at a fine spacing of Ci values (step size of 1 micromol mol^(-1)).
- parameters: An exdf object including the identifiers, fitting parameters, and convergence information for the A-Ci curve:
  - The number of points where Vpc and Vpr are each the smallest potential carboxylation rate are stored in the n_Vpc_smallest and n_Vpr_smallest columns.

- The best-fit values are stored in the alpha_psii, gbs, gmc_at_25, J_at_25, RL_at_25, Rm_frac, Vcmax_at_25, Vpmax_at_25, and Vpr columns. If calculate_confidence_intervals is TRUE, upper and lower limits for each of these parameters will also be included.
- For parameters that depend on leaf temperature, the average leaf-temperature-dependent values are stored in X_tl_avg columns: gmc_tl_avg, J_tl_avg, RL_tl_avg, Vcmax_tl_avg, and Vpmax_tl_avg.
- The average leaf temperature is also stored in the Tleaf_avg column.
- Information about the operating point is stored in operating_PCm, operating_Ci, operating_An, and operating_An_model.
- The convergence column indicates whether the fit was successful (==0) or if the optimizer encountered a problem (!=0).
- The feval column indicates how many cost function evaluations were required while finding the optimal parameter values.
- The residual stats as returned by residual_stats are included as columns with the default names: dof, RSS, RMSE, etc.
- The Akaike information criterion is included in the AIC column.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate temperature-dependent values of C4 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c4_temperature_param_vc)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)

# Fit just one curve from the data set (it is rare to do this).
one_result <- fit_c4_aci(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE],
  Ca_atmospheric = 420,
```

```
  optim_fun = optimizer
)

# Fit all curves in the data set (it is more common to do this)
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer
))

# View the fitting parameters for each species / plot
col_to_keep <- c(
  'species', 'plot',                                      # identifiers
 'RL_at_25', 'Vcmax_at_25', 'Vpmax_at_25', 'Vpr',      # parameters scaled to 25 degrees C
 'RL_tl_avg', 'Vcmax_tl_avg', 'Vpmax_tl_avg',         # average temperature-dependent values
  'operating_Ci', 'operating_An', 'operating_An_model',    # operating point info
  'dof', 'RSS', 'MSE', 'RMSE', 'RSE',                     # residual stats
  'convergence', 'convergence_msg', 'feval', 'optimum_val' # convergence info
)

aci_results$parameters[ , col_to_keep, TRUE]

# View the fits for each species / plot
plot_c4_aci_fit(aci_results, 'species_plot', 'Ci', ylim = c(0, 100))

# View the residuals for each species / plot
lattice::xyplot(
  A_residuals ~ Ci | species_plot,
  data = aci_results$fits$main_data,
  type = 'b',
  pch = 16,
  auto = TRUE,
  grid = TRUE,
  xlab = paste('Intercellular CO2 concentration [', aci_results$fits$units$Ci, ']'),
  ylab = paste('Assimilation rate residuals [', aci_results$fits$units$A_residuals, ']')
)
```

`fit_c4_aci_hyperbola` *Fits a hyperbolic C4 assimilation model to an experimental curve*

Description

Fits an empirical hyperbola model to an experimentally measured C4 A-Ci curve.

It is possible to fit the following parameters: `c4_curvature`, `c4_slope`, `rL`, and `Vmax`.

By default, all of these parameters are fit.

Best-fit parameters are found using maximum likelihood fitting, where the optimizer (`optim_fun`) is used to minimize the error function (defined by `error_function_c4_aci_hyperbola`).

Once best-fit parameters are found, confidence intervals are calculated using `confidence_intervals_c4_aci_hyperbola`.

See below for more details.

Usage

```
fit_c4_aci_hyperbola(
  replicate_exdf,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  sd_A = 'RMSE',
  optim_fun = optimizer_nmkb(1e-7),
  lower = list(),
  upper = list(),
  fit_options = list(),
  relative_likelihood_threshold = 0.147,
  hard_constraints = 0,
  calculate_confidence_intervals = TRUE,
  debug_mode = FALSE
)
```

Arguments

`replicate_exdf`	An `exdf` object representing one CO2 response curve.
`a_column_name`	The name of the column in `replicate_exdf` that contains the net assimilation in `micromol m^(-2) s^(-1)`.
`ci_column_name`	The name of the column in `replicate_exdf` that contains the intercellular CO2 concentration in `micromol mol^(-1)`.
`sd_A`	A value of the standard deviation of measured `A` values, or the name of a method for determining the deviation; currently, the only supported option is `'RMSE'`.
`optim_fun`	An optimization function that accepts the following input arguments: an initial guess, an error function, lower bounds, and upper bounds. It should return a list with the following elements: `par`, `convergence`, `feval`, and `convergence_msg`. See `optimizers` for a list of available options.
`lower`	A list of named numeric elements representing lower bounds to use when fitting. Values supplied here override the default values (see details below). For example, `lower = list(Vmax = 10)` sets the lower limit for `Vmax` to 10 micromol / m^2 / s.
`upper`	A list of named numeric elements representing upper bounds to use when fitting. Values supplied here override the default values (see details below). For example, `upper = list(Vmax = 200)` sets the upper limit for `Vmax` to 200 micromol / m^2 / s.
`fit_options`	A list of named elements representing fit options to use for each parameter. Values supplied here override the default values (see details below). Each element must be `'fit'`, `'column'`, or a numeric value. A value of `'fit'` means that the parameter will be fit; a value of `'column'` means that the value of the parameter will be taken from a column in `exdf_obj` of the same name; and a numeric value

means that the parameter will be set to that value. For example, `fit_options = list(rL = 0, Vmax = 'fit', c4_curvature = 'column')` means that `rL` will be set to 0, `Vmax` will be fit, and `c4_curvature` will be set to the values in the `c4_curvature` column of `replicate_exdf`.

`relative_likelihood_threshold`
: To be passed to `confidence_intervals_c4_aci_hyperbola` when `calculate_confidence_interval` is `TRUE`.

`hard_constraints`
: To be passed to `calculate_c4_assimilation_hyperbola`; see that function for more details.

`calculate_confidence_intervals`
: A logical value indicating whether or not to estimate confidence intervals for the fitting parameters using `confidence_intervals_c4_aci_hyperbola`.

`debug_mode`
: A logical (`TRUE` or `FALSE`) variable indicating whether to operate in debug mode. In debug mode, information about `replicate_exdf`, the initial guess, each guess supplied from the optimizer, and the final outcome is printed; this can be helpful when troubleshooting issues with a particular curve.

Details

This function calls `calculate_c4_assimilation_hyperbola` to calculate values of net assimilation. The user-supplied optimization function is used to vary the values of `c4_curvature`, `c4_slope`, `rL`, and `Vmax` to find ones that best reproduce the experimentally measured values of net assimilation. By default, the following options are used for the fits:

- `c4_curvature`: lower = -10, upper = 10, fit_option = `'fit'`
- `c4_slope`: lower = -50, upper = 1000, fit_option = `'fit'`
- `rL`: lower = -10, upper = 100, fit_option = `'fit'`
- `Vmax`: lower = -50, upper = 1000, fit_option = `'fit'`

With these settings, all of the parameters are fit during the process (see `fit_options` above). The bounds are chosen liberally to avoid any bias.

An initial guess for the parameters is generated by calling `initial_guess_c4_aci_hyperbola`. Note that any fixed values specified in the fit options will override the values returned by the guessing function.

The fit is made by creating an error function using `error_function_c4_aci_hyperbola` and minimizing its value using `optim_fun`, starting from the initial guess described above. The `optimizer_nmkb` optimizer is used by default since it has been found to reliably return great fits. However, it is a fast optimizer that can get stuck in local minima. If it seems to be returning bad fits, consider using the `optimizer_deoptim` optimizer instead, but be aware that the fits will take more time to complete.

Unlike the model represented by `calculate_c4_assimilation`, the model in `calculate_c4_assimilation_hyperbola` is smooth in the sense that small changes in the input parameters cause small changes in its outputs. Because of this, it is a fairly easy model to fit.

This function assumes that `replicate_exdf` represents a single C4 A-Ci curve. To fit multiple curves at once, this function is often used along with `by.exdf` and `consolidate`.

Value

A list with two elements:

- `fits`: An `exdf` object including the original contents of `replicate_exdf` along with several new columns:
  - The fitted values of net assimilation will be stored in a column whose name is determined by appending `'_fit'` to the end of `a_column_name`; typically, this will be `'A_fit'`.
  - Residuals (measured - fitted) will be stored in a column whose name is determined by appending `'_residuals'` to the end of `a_column_name`; typically, this will be `'A_residuals'`.
  - Values of fitting parameters will be stored in the `c4_curvature`, `c4_slope`, `rL`, and `Vmax` columns.
  - The other outputs from `calculate_c4_assimilation_hyperbola` will be stored in columns with the usual names: `Ag`, `Ainitial`, `Amax`, `An`, `c4_curvature`, `c4_slope`, `rL`, `Vinitial`, `Vmax`, and `c4_assimilation_hyperbola_msg`.
- `fits_interpolated`: An `exdf` object including the calculated assimilation rates at a fine spacing of `Ci` values (step size of 1 `micromol mol^(-1)`).
- `parameters`: An `exdf` object including the identifiers, fitting parameters, and convergence information for the A-Ci curve:
  - The best-fit values are stored in the `c4_curvature`, `c4_slope`, `rL`, and `Vmax`. If `calculate_confidence_interval` is `TRUE`, upper and lower limits for each of these parameters will also be included.
  - The `convergence` column indicates whether the fit was successful (`==0`) or if the optimizer encountered a problem (`!=0`).
  - The `feval` column indicates how many cost function evaluations were required while finding the optimal parameter values.
  - The residual stats as returned by `residual_stats` are included as columns with the default names: `dof`, `RSS`, `RMSE`, etc.
  - The Akaike information criterion is included in the `AIC` column.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Fit just one curve from the data set (it is rare to do this).
```

```
one_result <- fit_c4_aci_hyperbola(
  licor_file[licor_file[, 'species_plot'] == 'maize - 5', , TRUE]
)

# Fit all curves in the data set (it is more common to do this)
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci_hyperbola
))

# View the fitting parameters for each species / plot
col_to_keep <- c(
  'species', 'plot',                                      # identifiers
 'c4_curvature', 'c4_slope', 'rL', 'Vmax',           # best estimates for parameter values
  'dof', 'RSS', 'MSE', 'RMSE', 'RSE',                     # residual stats
  'convergence', 'convergence_msg', 'feval', 'optimum_val' # convergence info
)

aci_results$parameters[ , col_to_keep, TRUE]

# View the fits for each species / plot
plot_c4_aci_hyperbola_fit(aci_results, 'species_plot', ylim = c(0, 100))

# View the residuals for each species / plot
lattice::xyplot(
  A_residuals ~ Ci | species_plot,
  data = aci_results$fits$main_data,
  type = 'b',
  pch = 16,
  auto = TRUE,
  grid = TRUE,
  xlab = paste('Intercellular CO2 concentration [', aci_results$fits$units$Ci, ']'),
  ylab = paste('Assimilation rate residuals [', aci_results$fits$units$A_residuals, ']')
)
```

`fit_laisk` *Calculate RL and Ci_star using the Laisk method*

Description

Uses the Laisk method to estimate `Ci_star` and `RL`. This function can accomodate alternative colum names for the variables taken from log files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
fit_laisk(
  replicate_exdf,
```

```
  ci_lower = 40,  # ppm
  ci_upper = 120, # ppm
  a_column_name = 'A',
  ci_column_name = 'Ci',
  ppfd_column_name = 'PPFD'
)
```

Arguments

replicate_exdf	An exdf object containing multiple A-Ci curves measured at different levels of incident photosynthetically active photon flux density (PPFD).
ci_lower	Lower end of Ci range used for linear fits of An vs. Ci.
ci_upper	Upper end of Ci range used for linear fits of An vs. Ci.
a_column_name	The name of the column in replicate_exdf that contains the net CO2 assimilation rate An in micromol m^(-2) s^(-1).
ci_column_name	The name of the column in replicate_exdf that contains the intercellular CO2 concentration Ci in micromol mol^(-1).
ppfd_column_name	The name of the column in replicate_exdf that can be used to split it into individual response curves. Typically the individial curves are measured at different values of incident light, but the log entries for 'Qin' are not all exactly the same. It is advised to create a new column called 'PPFD' with rounded values. For example, licor_data[, 'PPFD'] <- round(licor_data[, 'Qin']).

Details

The Laisk method is a way to estimate RL and Ci_star for a C3 plant. Definitions of these quantities and a description of the theory underpinning this method is given below.

For a C3 plant, the net CO2 assimilation rate An is given by

An = Vc - Rp - RL,

where Vc is the rate of RuBP carboxylation, Rp is the rate of carbon loss due to photorespiration, and RL is the rate of carbon loss due to non-photorespiratory respiration (also known as the rate of day respiration, the rate of mitochondrial respiration, or the rate of respiration in the light). Because RuBP carboxylation and photorespiration both occur due to Rubisco activity, these rates are actually proportional to each other:

Rp = Vc * Gamma_star / Cc,

where Cc is the CO2 concentration in the chloroplast (where Rubisco is located) and Gamma_star will be discussed below. Using this expression, the net CO2 assimilation rate can be written as

An = Vc * (1 - Gamma_star / Cc) - RL.

When Cc is equal to Gamma_star, the net assimilation rate is equal to -RL. For this reason, Gamma_star is usually referred to as the CO2 compensation point in the absence of mitochondrial respiration.

In general, Cc is related to the intercellular CO2 concentration Ci according to

Ci = Cc + An / gmc,

where gmc is the mesophyll conductance to CO2 diffusion. When Cc is equal to Gamma_star, we therefore have Ci = Gamma_star - RL / gmc. This special value of Ci is referred to as Ci_star, and

can be understood as the value of Ci where Cc = Gamma_star and An = -RL. Note that the values of Gamma_star and Ci_star depend on Rubisco properties, mesophyll conductance, and the ambient O2 concentration, but not on the incident light intensity.

These observations suggest a method for estimating RL from a leaf: Measure An vs. Ci curves at several light intensities, and find the value of Ci where the curves intersect with each other. This will be Ci_star, and the corresponding value of An will be equal to -RL.

In practice, it is unlikely that the measured curves will all exactly intersect at a single point. A method for dealing with this issue was developed in Walker & Ort (2015) and described in more detail in Busch et al. (2024). Briefly, a linear fit is first made to each A-Ci curve, enabling the calculation of an intercept-slope curve. Then another linear fit is made to the intercept-slope curve. The intercept of this fit is equal to -RL and its slope is equal to -Ci_star.

Note: it is possible that RL depends on incident light intensity, an issue which complicates the application of the Laisk method. See the references for more details.

References:

- Yin, X., Sun, Z., Struik, P. C. & Gu, J. "Evaluating a new method to estimate the rate of leaf respiration in the light by analysis of combined gas exchange and chlorophyll fluorescence measurements." Journal of Experimental Botany 62, 3489–3499 (2011) [doi:10.1093/jxb/err038].
- Walker, B. J. & Ort, D. R. "Improved method for measuring the apparent CO2 photocompensation point resolves the impact of multiple internal conductances to CO2 to net gas exchange." Plant, Cell & Environment 38, 2462–2474 (2015) [doi:10.1111/pce.12562].
- Busch, F. A. et al. "A guide to photosynthetic gas exchang measurements: Fundamental principles, best practice and potential pitfalls." Plant, Cell & Environment 47, 3344–3364 (2024) [doi:10.1111/pce.14815].

Value

This function returns a list with the following named elements:

- first_fit_parameters: An exdf object with the slope (and its standard error), intercept (and its standard error), R-squared value, and p-value for each linear fit of A vs. Ci. These are included as the laisk_slope, laisk_slope_err, laisk_intercept, laisk_intercept_err, r_squared, and p_value columns.
- first_fits: An exdf object based on replicate_exdf that also includes the fitted values of An in a new column whose name is a_column_name followed by _fit (for example, A_fit). The fits are extrapolated to Ci = 0 so they can be visually checked for a common intersection point.
- second_fit_parameters: An exdf object with RL (and its standard error), Ci_Star (and its standard error) as estimated from a linear fit of laisk_intercept vs. laisk_slope. Also includes the R-squared and p-value of the fit.
- second_fit_parameters: An exdf object based on first_fit_parameters that also includes fitted values of laisk_intercept in the laisk_intercept_fit column.

As noted above, the estimated values of RL and Ci_star are included in the second_fit_parameters element of the returned list.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Apply the Laisk method. Note: this is a bad example because these curves were
# measured at the same light intensity, but from different species. Because of
# this, the results are not meaningful.
laisk_results <- fit_laisk(
  licor_file, 20, 150,
  ppfd_column_name = 'species_plot'
)

# Get estimated values
print(laisk_results$second_fit_parameters[, 'RL'])
print(laisk_results$second_fit_parameters[, 'Ci_star'])

# Plot the linear fits of A vs. Ci
plot_laisk_fit(laisk_results, 'instrument', 'first', ppfd_column_name = 'species_plot')

# Plot the linear fits of Laisk intercept vs. Laisk slope
plot_laisk_fit(laisk_results, 'instrument', 'second', ppfd_column_name = 'species_plot')
```

---

fit_medlyn *Fits the Medlyn model to an experimental curve*

Description

Fits measured values of stomatal conductance using the Medlyn model. This function can accomodate alternative column names for the variables taken from gas exchange log files in case they change at some point in the future. This function also checks the units of each required column and will produce an error if any units are incorrect.

Usage

```
fit_medlyn(
```

```
  replicate_exdf,
  a_column_name = 'A',
  csurface_column_name = 'Csurface',
  gsw_column_name = 'gsw',
  vpdleaf_column_name = 'VPDleaf'
)
```

Arguments

replicate_exdf	An exdf object representing one Ball-Berry curve.
a_column_name	The name of the column in replicate_exdf that contains the net assimilation in micromol m^(-2) s^(-1).
csurface_column_name	The name of the column in replicate_exdf that contains the CO2 concentration at the leaf surface in micromol mol^(-1).
gsw_column_name	The name of the column in replicate_exdf that contains the stomatal conductance to water vapor in mol m^(-2) s^(-1).
vpdleaf_column_name	The name of the column in replicate_exdf that contains the vapor pressure deficit at the leaf surface in kPa.

Details

The Medlyn model is a simple way to describe the response of a leaf's stomata to its assimilation rate and local environmental consitions. Specifically, it predicts that the stomatal conductance to water vapor (gsw) using the following equation:

gsw = g0 + 1.6 * (1 + g1 / sqrt(VPDleaf)) * A / Csurface,

where VPDleaf is the vapor pressure deficit at the leaf surface, A is the net CO2 assimilation rate, Csurface is the CO2 concentration at the leaf surface, g0 is the stomatal conductance when A is zero, and g1 is a parameter describing the leaf's combined response to environmental parameters.

Fits from this model are typically plotted with gsw on the Y-axis and A / (Csurface * sqrt(VPDleaf)) on the X-axis. Because g1 is typically close to or larger than 1, the model exhibits an almost linear response of gsw to A / (Csurface * sqrt(VPDleaf)), which we refer to as the "Medlyn index" in analogy with the Ball-Berry index (see calculate_ball_berry_index).

Although this model is certainly an oversimplification, it does encode some important stomatal responses. For example, when humidity is low, the stomata close, reducing stomatal conductance. Likewise, if the CO2 concentration around the leaf is depleted, the stomata open to allow more CO2 to diffuse into the leaf's interior, increasing somatal conductance.

The Medlyn model was originally described in Medlyn, B. E. et al. "Reconciling the optimal and empirical approaches to modelling stomatal conductance." Global Change Biology 17, 2134–2144 (2011) [doi:10.1111/j.13652486.2010.02375.x].

Medlyn parameters are typically determined using the same type of response curve measured for parameterizing the Ball-Berry model. See fit_ball_berry for more details.

This function uses nls to perform the fit, beginning from an initial guess of g0 = 0.005 and g1 = 4.

This function assumes that replicate_exdf represents a single response curve. To fit multiple curves at once, this function is often used along with by.exdf and consolidate.

Value

A list with two elements:

- `fits`: An `exdf` object including the measured values and the fitted values of stomatal conductance. The fitted values will be stored in a column whose name is determined by appending `'_fits'` to the end of `gsw_column_name`; typically, this will be `'gsw_fits'`. Also includes residuals in the `gsw_residuals` column and values of the Medlyn model parameters `medlyn_g0` and `medlyn_g1`.
- `parameters`: An `exdf` object including the fitting parameters and R-squared value. The Medlyn model parameters are stored in the `medlyn_g0` and `medlyn_g1` columns, their standard errors are stored in the `medlyn_g0_err` and `medlyn_g1_err` columns. Other statistical descriptors of the fit as calculated by `residual_stats` are also included.

Examples

```
# Read an example Licor file included in the PhotoGEA package, calculate
# additional gas properties, calculate the Ball-Berry index, define a new column
# that uniquely identifies each curve, and then perform a fit to extract the
# Ball-Berry parameters from each curve.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_gas_properties(licor_file)

licor_file[,'species_plot'] <-
  paste(licor_file[,'species'], '-', licor_file[,'plot'])

# Fit just one curve from the data set (it is rare to do this)
one_result <- fit_medlyn(
  licor_file[licor_file[, 'species_plot'] == 'soybean - 1a', , TRUE]
)

# Fit all curves in the data set (it is more common to do this)
medlyn_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_medlyn
))

# View the fitting parameters for each species / plot
col_to_keep <- c('species', 'plot', 'species_plot', 'medlyn_g0', 'medlyn_g1')
medlyn_results$parameters[ , col_to_keep]

# View the fits for each species / plot
lattice::xyplot(
  gsw + gsw_fit ~ medlyn_index | species_plot,
  data = medlyn_results$fits$main_data,
  type = 'b',
```

```
  pch = 16,
  auto = TRUE,
  xlab = paste('Medlyn index [', medlyn_results$fits$units$medlyn_index, ']'),
  ylab = paste('Stomatal conductance to H2O [', medlyn_results$fits$units$gsw, ']')
)
```

`get_oxygen_from_preamble` *Extract oxygen information from a Licor file*

Description

Ensures that the oxygen percentage (contained in the preamble of a Licor file) is properly documented (including units).

Usage

```
get_oxygen_from_preamble(licor_exdf)
```

Arguments

`licor_exdf`	An `exdf` object representing data from a photosynthetic gas exchange measurement system. It should contain a column called `Oxygen`; this should automatically be the case if `licor_exdf` was created by `read_gasex_file`.

Details

Licor LI-6800 log files include the oxygen concentration as an entry in the preamble, which is automatically converted to a column by `read_licor_6800_Excel` and `read_licor_6800_plaintext`. However, the Licor log files do not specify units for this value; they should be `percent`.

This function adds the correct units to the `Oxygen` column, or creates one (with values initialized to `NA`) if the `Oxygen` column is not present.

Also note that this function is called automatically by `read_licor_6800_Excel` and `read_licor_6800_plaintext` whenever `get_oxygen` is `TRUE`; this happens by default, so it is rare for a user to call this function directly.

Value

An `exdf` object based on `licor_exdf` that includes the `Oxygen` column units.

Examples

```
# Example: Read data from a Licor log file and get the oxygen information from
# the preamble

# Read the file without automatically specifying oxygen units
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
```

```
  get_oxygen = FALSE
)

# Here we can see that the oxygen column is included, but without units
print(licor_data$categories$Oxygen)
print(licor_data$units$Oxygen)
print(licor_data[, 'Oxygen'])

# Include the oxygen units
licor_data <- get_oxygen_from_preamble(licor_data)

# The units have changed, but the category and values have not
print(licor_data$categories$Oxygen)
print(licor_data$units$Oxygen)
print(licor_data[, 'Oxygen'])
```

`get_sample_valve_from_filename` *Extract TDL valve information from file name*

Description

Determines the TDL valve number from a photosynthetic gas exchange system log file name.

Usage

```
get_sample_valve_from_filename(
  exdf_obj,
  reference_table = NULL
)
```

Arguments

`exdf_obj`	An `exdf` object representing data from a photosynthetic gas exchange measurement system. The `exdf_obj$file_name` field must be defined and contain the file name; this will automatically be the case if `exdf_obj` was created by `read_gasex_file`.
`reference_table`	An optional list of named elements, where the name of each element is a Licor sample line valve number (as a character) and the value of each element is the corresponding Licor reference line valve number.

Details

When making combined gas exchange and isotope discrimination measurements using a portable photosynthetic gas exchange system (such as a Licor LI-6800) coupled with a tunable diode laser (TDL) absorption spectroscopy system, the TDL's gas handling system cycles through several gas lines (or sites) by opening and closing valves. When analyzing such data, a key step is to identify which TDL valve numbers correspond to the sample and reference gas lines of the Licor.

At UIUC, there is a convention for designating the sample line valve numbers in the Licor file names, where "siteNN" or "site NN" means that the Licor's sample line is valve NN in the TDL data file. The get_sample_valve_from_filename function extracts the valve number from the file name and stores it in a new column in exdf_obj called valve_number_s.

Optionally, it is also possible to specify the reference line valve number corresponding to each sample line valve number using the reference_table input argument. Reference line valve numbers will be stored in the valve_number_r column.

Value

An exdf object based on exdf_obj that includes the Licor sample line valve number as a new column called valve_number_s and (optionally) the Licor reference line valve number as a new column called valve_number_r.

Examples

```
## In this example we load a gas exchange data file and determine the TDL valve
## numbers from its file name

# Read the gas exchange data
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
)

# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# View the results
licor_data[, c('obs', 'valve_number_s', 'valve_number_r')]
```

---

identifier_columns *Find columns that have a single value across all rows*

Description

Identifies columns that have a single value across all rows and returns them.

Usage

```
  identifier_columns(x)

  ## S3 method for class 'data.frame'
identifier_columns(x)

  ## S3 method for class 'exdf'
identifier_columns(x)
```

Arguments

x	A table-like R object such as a data frame or an exdf.

Details

identifier_columns is generic, with methods defined for data frames and exdf objects.

identifier_columns gets the names and values of any columns in a table-like object that have a single unique value. If the object represents a set of data from one replicate, then these special columns are taken to be "identifiers" that describe the replicate. This function is often used inside fitting functions that are passed to by.exdf as its FUN input argument. For example, see the code for fit_ball_berry by typing PhotoGEA::fit_ball_berry in the R terminal.

Value

The return value will be a subset of x, restricted to only include columns whose values are constant. Only one row will be returned.

See Also

exdf

Examples

```
# Create a simple exdf object
simple_exdf <- exdf(
  data.frame(A = c(3, 2, 7, 9), species = c('a', 'a', 'a', 'a'), plot = c(1, 1, 1, 1)),
  data.frame(A = 'm', species = '', plot = ''),
  data.frame(A = 'Cat1', species = '', plot = '')
)

# Find its identifier columns
identifier_columns(simple_exdf)

# Apply the data frame method to the exdf object's main data frame
identifier_columns(simple_exdf$main_data)
```

---

identify_c3_limiting_processes

Identify C3 Limiting Processes

Description

Identify limiting processes in a C3 curve, typically the result of a fit. It is rate for users to call this function directly because it is used internally by fit_c3_aci and fit_c3_variable_j.

Usage

```
identify_c3_limiting_processes(
  data_table,
  a_column_name = 'A_fit',
  ac_column_name = 'Ac',
  aj_column_name = 'Aj',
  ap_column_name = 'Ap',
  tol = 1e-3
)
```

Arguments

data_table	A table-like R object such as a data frame or an exdf.
a_column_name	The name of the column in data_table that contains the modeled net CO2 assimilation rate in micromol m^(-2) s^(-1).
ac_column_name	The name of the column in data_table that contains the modeled Rubisco-limited net CO2 assimilation rate in micromol m^(-2) s^(-1).
aj_column_name	The name of the column in data_table that contains the modeled RuBP-regeneration-limited net CO2 assimilation rate in micromol m^(-2) s^(-1).
ap_column_name	The name of the column in data_table that contains the modeled TPU-limited net CO2 assimilation rate in micromol m^(-2) s^(-1).
tol	A relative tolerance factor used to identify when two rates are equal.

Details

For a C3 leaf, An is given by either Ac, Aj, or Ap. See the documentation for calculate_c3_assimilation for more information.

This function first identifies points where An = Ac, An = Aj, and An = Ap. The results are stored in columns called Ac_limiting, Aj_limiting, and Ap_limiting, where a value of TRUE indicates that the corresponding process is limiting.

Then, the overall limiting state is specified in the limiting_process column. For example, points where An equals Ac but not Aj or Ap are designated by limiting_process = 'Ac', and likewise for the other potential limiting processes. If more than one process is limiting for a point, limiting_process is set to 'co-limited'.

Value

An exdf object based on licor_exdf that includes new columns as described above: Ac_limiting, Aj_limiting, Ap_limiting, and limiting_process. The categories of these new columns are set to identify_c3_limiting_processes to indicate that they were created using this function.

Examples

```
# Identify limiting processes in an example curve
example_curve <- exdf(
  data.frame(
    A_fit = c(1.0, 2.0, 3.0, 4.0, 4.0),
```

```
    Ac    = c(1.0, 2.0, 5.0, 8.0, 9.0),
    Aj    = c(2.0, 2.5, 3.0, 4.0, 8.0),
    Ap    = c(NA,  NA,  4.0, 4.0, 4.0)
  ),
  units = data.frame(
    A_fit = 'micromol m^(-2) s^(-1)',
    Ac    = 'micromol m^(-2) s^(-1)',
    Aj    = 'micromol m^(-2) s^(-1)',
    Ap    = 'micromol m^(-2) s^(-1)',
    stringsAsFactors = FALSE
  )
)

identify_c3_limiting_processes(example_curve)

# This function also works for data frames
identify_c3_limiting_processes(example_curve$main_data)
```

`identify_common_columns` *Identify columns that are common to multiple objects*

Description

Checks whether the input arguments have the same columns

Usage

```
  identify_common_columns(...)

  ## S3 method for class 'data.frame'
identify_common_columns(...)

  ## S3 method for class 'exdf'
identify_common_columns(...)
```

Arguments

`...`	One or more R objects that have column names.

Details

`identify_common_columns` is generic, with methods defined for data frames and `exdf` objects. In the case of `exdf` objects, a column will only be considered common if it has the same name, units, and category in all of the input objects.

Value

A character vector of the column names that are common to all the input objects.

See Also

exdf

Examples

```
# Here we create two exdf objects with the same column names and units, but
# where the categories of one column are not the same in both objects
exdf_1 <- exdf(
  data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)),
  data.frame(A = 'm', B = 's'),
  data.frame(A = 'Cat1', B = 'Cat2')
)

exdf_2 <- exdf(
  data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)),
  data.frame(A = 'm', B = 's'),
  data.frame(A = 'Cat1', B = 'Cat3')
)

# Calling `identify_common_columns` on the exdf objects will only identify one
# common column (A) because the category for column B is not common to all the
# exdf objects.
identify_common_columns(exdf_1, exdf_2)

# Calling `identify_common_columns` on the main_data data frames will identify
# two common columns because unit and category information will not be
# considered here.
identify_common_columns(exdf_1$main_data, exdf_2$main_data)
```

---

identify_tdl_cycles *Identifying cycles in TDL data*

Description

Tool for identifying complete measurement cycles in a set of tunable diode laser (TDL) data.

Usage

```
identify_tdl_cycles(
  tdl_exdf,
  valve_column_name,
  cycle_start_valve,
  expected_cycle_length_minutes,
  expected_cycle_num_valves,
  expected_cycle_num_time_pts = expected_cycle_num_valves,
  timestamp_colname
)
```

Arguments

`tdl_exdf`	An `exdf` object representing data from a TDL data logger.
`valve_column_name`	The name of the column in `tdl_exdf` that contains the valve number; typically, this is `'valve_number'`.
`cycle_start_valve`	The value of the valve column that indicates the start of a new cycle.
`expected_cycle_length_minutes`	The expected length of a full cycle (in minutes); here the length is determined by the difference in timestamp between the first and last measurements that compose the cycle. For example, if a cycle consists of 9 valves that each require 20 seconds to measure, the expected length of the cycle in minutes would be 8 * 20 / 60 = 2.7 minutes (approximately).
`expected_cycle_num_valves`	The total number of unique valves that are measured in each cycle. For example, if a cycle consists of measuements from valves 1, 3, 13, 6, and 13, then `expected_cycle_num_valves` should be 4.
`expected_cycle_num_time_pts`	The total number of time points that are recorded in each cycle. For example, if 10 measuements are logged per second and a cycle is 12 minutes long, `expected_cycle_num_time_pts` should be `12 * 60 * 10 = 7200`.
`timestamp_colname`	The name of the column in `tdl_exdf` that contains the timestamp of each measurement; typically, this is `'TIMESTAMP'`.

Details

Typically a TDL system periodically cycles between multiple gas lines during measurements. Some of the gas lines represent gas mixtures with known composition that can be used for calibration, while others are the "unknown" mixtures whose composition is being measured. A collection of valves are used to control which gas line is being measured at any given time, and the "active" valve for each recorded data point is included in a measurement file.

When using the calibration lines to apply corrections to the measured data, it is necessary to first identify complete measurements cycles within the data set. Here, complete cycles are identified using the following criteria:

- A cycle is said to begin when the value of `valve_column_name` is `cycle_start_valve`.
- A cycle ends after `expected_cycle_num_valves` valves have been measured.
- The time difference between the first and last points of a cycle cannot deviate from `expected_cycle_length_minutes` by more than +/- 30 seconds.

In addition to identifying valid measurement cycles within the data, `identify_tdl_cycles` also calculates the elapsed time at the beginning of each cycle (in minutes).

If no cycles are identified, the most likely explanation is that the values of some of the key input arguments (such as `cycle_start_valve`, `expected_cycle_length_minutes`, or `expected_cycle_num_valves`) may be incorrect for the data in `tdl_exdf`. In this case, an error message will be sent.

Value

An exdf object based on tdl_exdf that includes two new columns: the cycle_num column indicates the measurement cycle corresponding to each measurement, and the elapsed_time column indicates the elapsed time (in minutes) at the start of each cycle. Any rows in tdl_exdf that were not found to be part of a complete cycle will not be included in the return value.

Examples

```
# Example: reading a TDL file that is included with the PhotoGEA package and
# identifying its measurement cycles.
tdl_file <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_sampling_1.dat'),
  'TIMESTAMP'
)

tdl_file <- identify_tdl_cycles(
  tdl_file,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

str(tdl_file) # Notice the two new columns: `cycle_num` and `elapsed_time`
```

---

initial_guess_c3_aci *Make an initial guess of FvCB model parameter values for one curve*

Description

Creates a function that makes an initial guess of FvCB model parameter values for one curve. This function is used internally by fit_c3_aci.

Values estimated by this guessing function should be considered inaccurate, and should always be improved upon by an optimizer.

Usage

```
initial_guess_c3_aci(
  alpha_g,
  alpha_old,
  alpha_s,
  alpha_t,
  Gamma_star_at_25,
  gmc_at_25,
  Kc_at_25,
  Ko_at_25,
  cc_threshold_rl = 100,
```

```
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  debug_mode = FALSE
)
```

Arguments

alpha_g	A dimensionless parameter where 0 <= alpha_g <= 1, representing the proportion of glycolate carbon taken out of the photorespiratory pathway as glycine. alpha_g is often assumed to be 0. If alpha_g is not a number, then there must be a column in rc_exdf called alpha_g with appropriate units. A numeric value supplied here will overwrite the values in the alpha_g column of rc_exdf if it exists.
alpha_old	A dimensionless parameter where 0 <= alpha_old <= 1, representing the fraction of remaining glycolate carbon not returned to the chloroplast after accounting for carbon released as CO2. alpha_old is often assumed to be 0. If alpha_old is not a number, then there must be a column in rc_exdf called alpha_old with appropriate units. A numeric value supplied here will overwrite the values in the alpha_old column of rc_exdf if it exists.
alpha_s	A dimensionless parameter where 0 <= alpha_s <= 0.75 * (1 - alpha_g) representing the proportion of glycolate carbon taken out of the photorespiratory pathway as serine. alpha_s is often assumed to be 0. If alpha_s is not a number, then there must be a column in rc_exdf called alpha_s with appropriate units. A numeric value supplied here will overwrite the values in the alpha_s column of rc_exdf if it exists.
alpha_t	A dimensionless parameter where 0 <= alpha_t <= 1 representing the proportion of glycolate carbon taken out of the photorespiratory pathway as CH2-THF. alpha_t is often assumed to be 0. If alpha_t is not a number, then there must be a column in rc_exdf called alpha_t with appropriate units. A numeric value supplied here will overwrite the values in the alpha_t column of rc_exdf if it exists.
Gamma_star_at_25	The chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at 25 degrees C, expressed in micromol mol^(-1). If Gamma_star_at_25 is not a number, then there must be a column in rc_exdf called Gamma_star_at_25

with appropriate units. A numeric value supplied here will overwrite the values in the Gamma_star_at_25 column of rc_exdf if it exists.

gmc_at_25 The mesophyll conductance to CO2 diffusion at 25 degrees C, expressed in mol m^(-2) s^(-1) bar^(-1). In the absence of other reliable information, gmc_at_25 is often assumed to be infinitely large. If gmc_at_25 is not a number, then there must be a column in rc_exdf called gmc_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the gmc_at_25 column of rc_exdf if it exists.

Kc_at_25 The Michaelis-Menten constant for Rubisco carboxylation at 25 degrees C, expressed in micromol mol^(-1). If Kc_at_25 is not a number, then there must be a column in rc_exdf called Kc_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Kc_at_25 column of rc_exdf if it exists.

Ko_at_25 The Michaelis-Menten constant for Rubisco oxygenation at 25 degrees C, expressed in mmol mol^(-1). If Ko_at_25 is not a number, then there must be a column in rc_exdf called Ko_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Ko_at_25 column of rc_exdf if it exists.

cc_threshold_rl

An upper cutoff value for the chloroplast CO2 concentration in micromol mol^(-1) to be used when estimating RL.

Wj_coef_C A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

Wj_coef_Gamma_star

A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

a_column_name The name of the column in rc_exdf that contains the net assimilation in micromol m^(-2) s^(-1).

ci_column_name The name of the column in rc_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).

gamma_star_norm_column_name

The name of the column in rc_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).

gmc_norm_column_name

The name of the column in rc_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).

j_norm_column_name

The name of the column in rc_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_norm_column_name

The name of the column in rc_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).

ko_norm_column_name

The name of the column in rc_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
: The name of the column in rc_exdf that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

rl_norm_column_name
: The name of the column in rc_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in rc_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in rc_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in rc_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

debug_mode
: A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the linear fit used to estimate RL is printed; this can be helpful when troubleshooting issues with a particular curve.

Details

Here we estimate values of J_at_25, RL_at_25, Tp_at_25, and Vcmax_at_25 from a measured C3 CO2 response curve. It is difficult to estimate values of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, gmc_at_25, Kc_at_25, Ko_at_25 from a curve, so they must be supplied beforehand. For more information about these parameters, see the documentation for calculate_c3_assimilation.

- **Estimating RL**: Regardless of which process is limiting at low Cc, it is always true that An = -RL when Cc = Gamma_star_agt. Here we make a linear fit of the measured An vs. Cc values where Cc is below cc_threshold_rl, and evaluate it at at Cc = Gamma_star_agt to estimate RL. If there are fewer than two points with Cc <= cc_threshold_rl, the fit cannot be made, and we use a typical value instead (1.0 micromol m^(-2) s^(-1)). Likewise, if the linear fit predicts a negative or NA value for RL, we use the same typical value instead.
- **Estimating Vc**: Once an estimate for RL has been found, the RuBP carboxylation rate Vc can be estimated using Vc = (An + RL) / (1 - Gamma_star_agt / Cc). This is useful for the remaining parameter estimates.
- **Estimating Vcmax**: An estimate for Vcmax can be obtained by solving the equation for Wc for Vcmax, and evaluating it with Wc = Vc as estimated above. In the rubisco-limited part of the curve, Vc = Wc and the estimated values of Vcmax should be reasonable. In other parts of the curve, Wc is not the limiting rate, so Vc < Wc. Consequently, the estimated values of Vcmax in these parts of the curve will be smaller. So, to make an overall estimate, we choose the the largest estimated Vcmax value.
- **Estimating J and Tp**: Estimates for these parameters can be made using the equations for Wj and Wp, similar to the approach followed for Vcmax.

For the parameter values estimated above, the values of RL_norm, Vcmax_norm, and J_norm are used to convert the values at leaf temperature to the values at 25 degrees C.

Value

A function with one input argument `rc_exdf`, which should be an `exdf` object representing one C3 CO2 response curve. The return value of this function will be a numeric vector with twelve elements, representing the values of `alpha_g`, `alpha_old`, `alpha_s`, `alpha_t`, `Gamma_star_at_25`, `gmc_at_25`, `J_at_25`, `Kc_at_25`, `Ko_at_25`, `RL_at_25`, `Tp_at_25`, and `Vcmax_at_25` (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Create the guessing function; here we set:
# - All alpha values to 0
# - Gamma_star_at_25 to 40 micromol / mol
# - gmc to infinity
# - Kc_at_25 to 400 micromol / mol
# - Ko_at_25 to 275 mmol / mol
guessing_func <- initial_guess_c3_aci(
  alpha_g = 0,
  alpha_old = 0,
  alpha_s = 0,
  alpha_t = 0,
  Gamma_star = 40,
  gmc_at_25 = Inf,
  Kc_at_25 = 400,
  Ko_at_25 = 275
)

# Apply it and see the initial guesses for each curve
print(by(licor_file, licor_file[, 'species_plot'], guessing_func))
```

```
# A simple way to visualize the guesses is to "fit" the curves using the null
# optimizer, which simply returns the initial guess
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
  fit_options = list(alpha_old = 0),
  optim_fun = optimizer_null(),
  remove_unreliable_param = 0
))

plot_c3_aci_fit(aci_results, 'species_plot', 'Ci')
```

`initial_guess_c3_variable_j` *Make an initial guess of "Variable J" model parameter values for one curve*

Description

Creates a function that makes an initial guess of "variable J" model parameter values for one curve. This function is used internally by `fit_c3_variable_j`.

Values estimated by this guessing function should be considered inaccurate, and should always be improved upon by an optimizer.

Usage

```
initial_guess_c3_variable_j(
  alpha_g,
  alpha_old,
  alpha_s,
  alpha_t,
  Gamma_star_at_25,
  Kc_at_25,
  Ko_at_25,
  cc_threshold_rl = 100,
  Wj_coef_C = 4.0,
  Wj_coef_Gamma_star = 8.0,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  etr_column_name = 'ETR',
  gamma_star_norm_column_name = 'Gamma_star_norm',
  j_norm_column_name = 'J_norm',
  kc_norm_column_name = 'Kc_norm',
  ko_norm_column_name = 'Ko_norm',
  oxygen_column_name = 'Oxygen',
  phips2_column_name = 'PhiPS2',
  qin_column_name = 'Qin',
```

```
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  tp_norm_column_name = 'Tp_norm',
  vcmax_norm_column_name = 'Vcmax_norm',
  debug_mode = FALSE
)
```

Arguments

alpha_g	A dimensionless parameter where 0 <= alpha_g <= 1, representing the proportion of glycolate carbon taken out of the photorespiratory pathway as glycine. alpha_g is often assumed to be 0. If alpha_g is not a number, then there must be a column in rc_exdf called alpha_g with appropriate units. A numeric value supplied here will overwrite the values in the alpha_g column of rc_exdf if it exists.
alpha_old	A dimensionless parameter where 0 <= alpha_old <= 1, representing the fraction of remaining glycolate carbon not returned to the chloroplast after accounting for carbon released as CO2. alpha_old is often assumed to be 0. If alpha_old is not a number, then there must be a column in rc_exdf called alpha_old with appropriate units. A numeric value supplied here will overwrite the values in the alpha_old column of rc_exdf if it exists.
alpha_s	A dimensionless parameter where 0 <= alpha_s <= 0.75 * (1 - alpha_g) representing the proportion of glycolate carbon taken out of the photorespiratory pathway as serine. alpha_s is often assumed to be 0. If alpha_s is not a number, then there must be a column in rc_exdf called alpha_s with appropriate units. A numeric value supplied here will overwrite the values in the alpha_s column of rc_exdf if it exists.
alpha_t	A dimensionless parameter where 0 <= alpha_t <= 1 representing the proportion of glycolate carbon taken out of the photorespiratory pathway as CH2-THF. alpha_t is often assumed to be 0. If alpha_t is not a number, then there must be a column in rc_exdf called alpha_t with appropriate units. A numeric value supplied here will overwrite the values in the alpha_t column of rc_exdf if it exists.
Gamma_star_at_25	The chloroplastic CO2 concentration at which CO2 gains from Rubisco carboxylation are exactly balanced by CO2 losses from Rubisco oxygenation, at 25 degrees C, expressed in micromol mol^(-1). If Gamma_star_at_25 is not a number, then there must be a column in rc_exdf called Gamma_star_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Gamma_star_at_25 column of rc_exdf if it exists.
Kc_at_25	The Michaelis-Menten constant for Rubisco carboxylation at 25 degrees C, expressed in micromol mol^(-1). If Kc_at_25 is not a number, then there must be a column in rc_exdf called Kc_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Kc_at_25 column of rc_exdf if it exists.
Ko_at_25	The Michaelis-Menten constant for Rubisco oxygenation at 25 degrees C, expressed in mmol mol^(-1). If Ko_at_25 is not a number, then there must be a

column in rc_exdf called Ko_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the Ko_at_25 column of rc_exdf if it exists.

cc_threshold_rl
: An upper cutoff value for the chloroplast CO2 concentration in micromol mol^(-1) to be used when estimating RL.

Wj_coef_C
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

Wj_coef_Gamma_star
: A coefficient in the equation for RuBP-regeneration-limited carboxylation, whose value depends on assumptions about the NADPH and ATP requirements of RuBP regeneration; see calculate_c3_assimilation for more information.

a_column_name
: The name of the column in rc_exdf that contains the net assimilation in micromol m^(-2) s^(-1).

ci_column_name
: The name of the column in rc_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).

etr_column_name
: The name of the column in rc_exdf that contains the electron transport rate as estimated by the measurement system in micromol m^(-2) s^(-1).

gamma_star_norm_column_name
: The name of the column in rc_exdf that contains the normalized Gamma_star values (with units of normalized to Gamma_star at 25 degrees C).

j_norm_column_name
: The name of the column in rc_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kc_norm_column_name
: The name of the column in rc_exdf that contains the normalized Kc values (with units of normalized to Kc at 25 degrees C).

ko_norm_column_name
: The name of the column in rc_exdf that contains the normalized Ko values (with units of normalized to Ko at 25 degrees C).

oxygen_column_name
: The name of the column in exdf_obj that contains the concentration of O2 in the ambient air, expressed as a percentage (commonly 21% or 2%); the units must be percent.

phips2_column_name
: The name of the column in rc_exdf that contains values of the operating efficiency of photosystem II (dimensionless).

qin_column_name
: The name of the column in rc_exdf that contains values of the incident photosynthetically active flux density in micromol m^(-2) s^(-1).

rl_norm_column_name
: The name of the column in rc_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
: The name of the column in rc_exdf that contains the total pressure in bar.

tp_norm_column_name
: The name of the column in rc_exdf that contains the normalized Tp values (with units of normalized to Tp at 25 degrees C).

vcmax_norm_column_name
: The name of the column in rc_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

debug_mode
: Passed to initial_guess_c3_aci.

Details

The variable J method is a fitting procedure for estimating values of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, J_at_25, Kc_at_25, Kc_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 from a measured C3 CO2 response curve + chlorophyll fluorescence. For more information about these parameters, see the documentation at calculate_c3_variable_j and calculate_c3_assimilation.

Here, we make an estimate for tau by noting that gas exchange measurement systems equipped with chlorophyll fluorometers typically make an estimate for the electron transport rate (ETR), which is essentially synonymous with the actual RuBP regeneration rate. Thus, tau can be estimated by inverting the equation for J_actual:

tau = ETR / (Qin * PhiPSII)

Estimates of the remaining parameters are calculated by setting Cc = Ci and then calling initial_guess_c3_aci.

Value

A function with one input argument rc_exdf, which should be an exdf object representing one C3 CO2 response curve. The return value of this function will be a numeric vector with twelve elements, representing the values of alpha_g, alpha_old, alpha_s, alpha_t, Gamma_star_at_25, J_at_25, Kc_at_25, Ko_at_25, RL_at_25, tau, Tp_at_25, and Vcmax_at_25 (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)
```

```
# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# Create the guessing function; here we set:
# - All alpha values to 0
# - Gamma_star_at_25 to 40 micromol / mol
# - Kc_at_25 to 400 micromol / mol
# - Ko_at_25 to 275 mmol / mol
guessing_func <- initial_guess_c3_variable_j(
  alpha_g = 0,
  alpha_old = 0,
  alpha_s = 0,
  alpha_t = 0,
  Gamma_star = 40,
  Kc_at_25 = 400,
  Ko_at_25 = 275
)

# Apply it and see the initial guesses for each curve
print(by(licor_file, licor_file[, 'species_plot'], guessing_func))

# A simple way to visualize the guesses is to "fit" the curves using the null
# optimizer, which simply returns the initial guess
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_variable_j,
  fit_options = list(alpha_old = 0),
  optim_fun = optimizer_null(),
  remove_unreliable_param = 0
))

plot_c3_aci_fit(aci_results, 'species_plot', 'Ci')
```

`initial_guess_c4_aci` *Make an initial guess of C4 photosynthesis parameter values for one curve*

Description

Creates a function that makes an initial guess of C4 photosynthesis model parameter values for one curve. This function is used internally by `fit_c4_aci`.

Values estimated by this guessing function should be considered inaccurate, and should always be improved upon by an optimizer.

Usage

```
initial_guess_c4_aci(
  alpha_psii,
  gbs,
  gmc_at_25,
  Rm_frac,
  pcm_threshold_rlm = 40,
  x_etr = 0.4,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  gmc_norm_column_name = 'gmc_norm',
  j_norm_column_name = 'J_norm',
  kp_column_name = 'Kp',
  rl_norm_column_name = 'RL_norm',
  total_pressure_column_name = 'total_pressure',
  vcmax_norm_column_name = 'Vcmax_norm',
  vpmax_norm_column_name = 'Vpmax_norm',
  debug_mode = FALSE
)
```

Arguments

alpha_psii	The fraction of photosystem II activity in the bundle sheath (dimensionless). If alpha_psii is not a number, then there must be a column in rc_exdf called alpha_psii with appropriate units. A numeric value supplied here will overwrite the values in the alpha_psii column of rc_exdf if it exists.
gbs	The bundle sheath conductance to CO2 in mol m^(-2) s^(-1) bar^(-1). If gbs is not a number, then there must be a column in rc_exdf called gbs with appropriate units. A numeric value supplied here will overwrite the values in the gbs column of rc_exdf if it exists.
gmc_at_25	The mesophyll conductance to CO2 diffusion at 25 degrees C, expressed in mol m^(-2) s^(-1) bar^(-1). If gmc_at_25 is not a number, then there must be a column in rc_exdf called gmc_at_25 with appropriate units. A numeric value supplied here will overwrite the values in the gmc_at_25 column of rc_exdf if it exists.
Rm_frac	The fraction of the total mitochondrial respiration that occurs in the mesophyll. If Rm_frac is not a number, then there must be a column in rc_exdf called Rm_frac with appropriate units. A numeric value supplied here will overwrite the values in the Rm_frac column of rc_exdf if it exists.
pcm_threshold_rlm	An upper cutoff value for the partial pressure of CO2 in the mesophyll (in microbar) to be used when estimating RLm.
x_etr	The fraction of whole-chain electron transport occurring in the mesophyll (dimensionless). See Equation 29 from S. von Caemmerer (2021).
a_column_name	The name of the column in rc_exdf that contains the net assimilation in micromol m^(-2) s^(-1).

ci_column_name The name of the column in rc_exdf that contains the intercellular CO2 concentration in micromol mol^(-1).

gmc_norm_column_name
The name of the column in rc_exdf that contains the normalized mesophyll conductance values (with units of normalized to gmc at 25 degrees C).

j_norm_column_name
The name of the column in rc_exdf that contains the normalized J values (with units of normalized to J at 25 degrees C).

kp_column_name The name of the column in rc_exdf that contains the Michaelis-Menten constant for PEP carboxylase carboxylation in microbar.

rl_norm_column_name
The name of the column in rc_exdf that contains the normalized RL values (with units of normalized to RL at 25 degrees C).

total_pressure_column_name
The name of the column in rc_exdf that contains the total pressure in bar.

vcmax_norm_column_name
The name of the column in rc_exdf that contains the normalized Vcmax values (with units of normalized to Vcmax at 25 degrees C).

vpmax_norm_column_name
The name of the column in rc_exdf that contains the normalized Vpmax values (with units of normalized to Vpmax at 25 degrees C).

debug_mode A logical (TRUE or FALSE) variable indicating whether to operate in debug mode. In debug mode, information about the linear fit used to estimate RL is printed; this can be helpful when troubleshooting issues with a particular curve.

Details

Here we estimate values of J_at_25, RL_at_25, Vcmax_at_25, Vpmax_at_25, and Vpr from a measured C4 CO2 response curve. It is difficult to estimate values of alpha_psii, gbs, gmc_at_25, and Rm_frac from a curve, so they must be supplied beforehand. For more information about these parameters, see the documentation for calculate_c4_assimilation. To estimate these parameter values, we use several equations from S. von Caemmerer, "Biochemical Models of Leaf Photosynthesis" (CSIRO Publishing, 2000) [doi:10.1071/9780643103405]. Any equation numbers referenced below are from this book.

- **Estimating RL**: An estimate for RLm can be obtained using Equation 4.26, which applies for low values of PCm. In this situation, PCm + Kp can be approximated by Kp, and Equation 4.26 simplifies to a linear relationship between the net assimilation An and PCm: An = (gbs + Vpmax / kP) * PCm - RLm. So, to estimate RLm, we make a linear fit of An vs. PCm in the low PCm range (PCm <= pcm_threshold_rlm) where this equation is expected to be valid. Then RLm is given by the negative of the intercept from the fit. In the C4 assimilation model, we assume that RLm = Rm_frac * RL, so we can also estimate RL = RLm / Rm_frac from this value.

  If there are fewer than two points with PCm <= pcm_threshold_rlm, the fit cannot be made, and we use a typical value instead (0.5 micromol m^(-2) s^(-1)). Likewise, if the linear fit predicts a negative or NA value for RLm, we use the same typical value instead.

- **Estimating Vpmax**: An estimate for Vpmax can also be obtained from Equation 4.26. In this case, we simply solve the equation for Vpmax and use it to calculate a value of Vpmax at each

point in the curve from the measured values of An and PCm, the input value of gbs, and the value of RLm estimated above. In the PEP-carboxylation-limited range, the estimated values of Vpmax should be reasonable. In other parts of the curve, the assimilation rate is limited by other factors, so An will be smaller than the PEP-carboxylation-limited values, causing the estimated values of Vpmax to be smaller. So, to make an overall estimate, we choose the largest estimated Vpmax value.

- **Estimating Vcmax**: An estimate for Vcmax can be obtained by solving An = Vcmax - RL for Vcmax, similar to the method used to estimate Vpmax.
- **Estimating Vpr**: An estimate for Vpr can be obtained by solving An = Vpr + gbs * PCm - RLm for Vpr, similar to the method used to estimate Vpmax.
- **Estimating J**: First, an estimate for J can be obtained by solving An = (1 - x_etr) * J / 3 - RL for J. Then, estimates of J can be made from J and Qin. The largest value of J / J_norm is chosen as the best estimate for J_at_25.

Note that a key assumption underlying this approach is that the net assimilation can be reasonably approximated by An = min(Apc, Apr, Ar, Ajm) (Equations 4.19, 4.25, 4.45, and 4.47 combined). While this approximation seems to work well for low values of PCm, it tends to deviate significantly from the more accurate version at higher values of PCm, predicting values that are noticably smaller. Thus, the values of Vcmax and Vpr estimated using this procedure are unlikely to be accurate. This is not a problem; instead it simply highlights the importance of improving this initial guess using an optimizer, which can be accomplished via fit_c4_aci.

Value

A function with one input argument rc_exdf, which should be an exdf object representing one C4 CO2 response curve. The return value of this function will be a numeric vector with eight elements, representing the values of alpha_psii, gbs, J_at_25, RL_at_25, rm_frac, Vcmax_at_25, Vpmax_at_25, and Vpr (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate temperature-dependent values of C4 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c4_temperature_param_vc)
```

```
# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Create the guessing function, using typical values for the alpha_psii, gbs,
# gmc_at_25, and Rm_frac: 0, 0.003, 1, and 0.5
guessing_func <- initial_guess_c4_aci(0, 0.003, 1, 0.5)

# Apply it and see the initial guesses for each curve
print(by(licor_file, licor_file[, 'species_plot'], guessing_func))

# A simple way to visualize the guesses is to "fit" the curves using the null
# optimizer, which simply returns the initial guess
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci,
  optim_fun = optimizer_null()
))

plot_c4_aci_fit(aci_results, 'species_plot', 'Ci', ylim = c(-10, 100))
```

`initial_guess_c4_aci_hyperbola` *Make an initial guess of C4 hyperbola parameter values for one curve*

Description

Creates a function that makes an initial guess of C4 hyperbola model parameter values for one curve. This function is used internally by `fit_c4_aci_hyperbola`.

Values estimated by this guessing function should be considered inaccurate, and should always be improved upon by an optimizer.

Usage

```
initial_guess_c4_aci_hyperbola(
  a_column_name = 'A'
)
```

Arguments

`a_column_name`	The name of the column in `rc_exdf` that contains the net assimilation rate in `micromol m^(-2) s^(-1)`.

Details

Here we estimate values of `c4_curvature`, `c4_slope`, `rL`, and `Vmax` from a measured C4 CO2 response curve. For more information about these parameters, see the documentation for `calculate_c4_assimilation_hyper`

Here we take a very simple approach to forming the initial guess. We always choose `c4_curvature` = `0.5`, `c4_slope` = `1.0`, and `rL` = `0.0`. For `Vmax`, we use `Vmax` = `max{A}` - `rL_guess`, where `max{A}` is the largest observed net CO2 assimilation rate and `rL_guess` is the guess for `rL`.

Value

A function with one input argument `rc_exdf`, which should be an `exdf` object representing one C4 CO2 response curve. The return value of this function will be a numeric vector with four elements, representing the values of `c4_curvature`, `c4_slope`, `rL`, and `Vmax` (in that order).

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Create the guessing function
guessing_func <- initial_guess_c4_aci_hyperbola()

# Apply it and see the initial guesses for each curve
print(by(licor_file, licor_file[, 'species_plot'], guessing_func))

# A simple way to visualize the guesses is to "fit" the curves using the null
# optimizer, which simply returns the initial guess
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci_hyperbola,
  optim_fun = optimizer_null()
))

plot_c4_aci_hyperbola_fit(aci_results, 'species_plot', ylim = c(-10, 100))
```

`is.exdf` *Is an object an exdf?*

Description

Checks whether an object is an exdf object.

Usage

```
is.exdf(x, consistency_check = FALSE)
```

Arguments

x	An R object.
consistency_check	A logical value indicating whether to perform additional consistency checks.

Details

The default version of is.exdf simply checks to see if 'exdf' is in class(x).

If consistency_check is TRUE, then additional checks will be performed to make sure the object has three elements named main_data, units, and categories; that these elements are data frames with the same column names; and that units and categories each have one row. These requirements are all part of the definition of an exdf object, but these checks require additional time so they are not always desired.

Value

A logical (TRUE / FALSE) value indicating whether the object is an exdf object.

See Also

exdf

Examples

```
# Test a simple exdf object
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))
is.exdf(simple_exdf)
is.exdf(simple_exdf, TRUE)

# Test an object that is clearly not an exdf
not_an_exdf <- 2
is.exdf(not_an_exdf)
is.exdf(not_an_exdf, TRUE)

# Test an object that claims to be an exdf but does not meet all of the
# requirements
fake_exdf <- not_an_exdf
class(fake_exdf) <- c('exdf', class(fake_exdf))
is.exdf(fake_exdf)
is.exdf(fake_exdf, TRUE)
```

`jmax_temperature_param_bernacchi` *Jmax-related temperature response parameters from Bernacchi et al.*

Description

Parameters describing the temperature response of Jmax-related photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
jmax_temperature_param_bernacchi
```

Format

List with 2 named elements that each represent a variable whose temperature-dependent value can be calculated using a polynomial equation:

- `alpha_j_norm`: The apparent quantum efficiency of electron transport (`alpha_j`) normalized to its value at 25 degrees C.
- `theta_j_norm`: The empirical curvature parameter normalized to its value at 25 degrees C.

In turn, each of these elements is a list with 3 named elements:

- `type`: the type of temperature response.
- `coef`: the polynomial coefficients.
- `units`: the units of the corresponding variable.

Source

Polynomial coefficients were obtained from Bernacchi et al. (2003). Here, we use the values determined from plants grown at 25 degrees C (Table 2). The coefficients given in the paper are used to calculate the values of `alpha_j` and `theta_j` at leaf temperature. Here we normalize by the values of `alpha_j` and `theta_j` at 25 degrees C, which are 0.6895 and 0.97875, respectively.

References:

- Bernacchi, C. J., Pimentel, C. & Long, S. P. "In vivo temperature response functions of parameters required to model RuBP-limited photosynthesis" Plant, Cell & Environment 26, 1419–1430 (2003) [doi:10.1046/j.00168025.2003.01050.x].

jmax_temperature_param_flat *Jmax-related temperature response parameters from Bernacchi et al.*

Description

Parameters that describe a flat temperature response (in other words, no dependence on temperature) for Jmax-related photosynthetic parameters, intended to be passed to the `calculate_temperature_response` function.

Usage

```
jmax_temperature_param_flat
```

Format

List with 2 named elements that each represent a variable whose temperature-dependent value can be calculated using a polynomial equation:

- `alpha_j_norm`: The apparent quantum efficiency of electron transport (`alpha_j`) normalized to its value at 25 degrees C.
- `theta_j_norm`: The empirical curvature parameter normalized to its value at 25 degrees C.

In turn, each of these elements is a list with 3 named elements:

- `type`: the type of temperature response.
- `coef`: the polynomial coefficients.
- `units`: the units of the corresponding variable.

Source

Here, the polynomial coefficients (`coef`) are all set to 1, speciying a zeroth-order polynomial equal to 1, which means that the values will not depend on temperature.

length.exdf *Length of an exdf object*

Description

Returns the length of an `exdf` object's `main_data`.

Usage

```
  ## S3 method for class 'exdf'
length(x)
```

Arguments

x	An exdf object.

Value

Returns `length(x[['main_data']])`.

See Also

exdf

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))
length(simple_exdf)
length(simple_exdf[['main_data']]) # An equivalent command
```

---

multi_curve_colors *Set of colors for plotting multiple curves*

Description

`multi_curve_colors` returns a vector of color specifications that work reasonably well for plotting multiple curves on the same axes.

`multi_curve_line_colors` returns the same vector, but with the first color set to be transparent. `multi_curve_point_colors` also returns the same vector, but with all colors except the first set to transparent. These color specifications can be helpful when plotting measured data along with fits, allowing the data to be displayed as points and the fits as lines.

Usage

```
multi_curve_colors()

multi_curve_line_colors()

multi_curve_point_colors()
```

Details

The color set was originally formed by calling the following:

`multi_curve_colors <- c( "#000000", RColorBrewer::brewer.pal(8, "Set2"), RColorBrewer::brewer.pal(12, "Paired")[c(1:10,12)], RColorBrewer::brewer.pal(8, "Dark2") )`

Value

A character vector with 28 elements, each of which is a hexadecimal color specification.

Examples

```
multi_curve_colors()

multi_curve_line_colors()

multi_curve_point_colors()
```

---

optimizers *Optimizers*

Description

These functions return optimizers that meet requirements for the optim_fun input argument of fit_c3_aci, fit_c3_variable_j, fit_c4_aci, and fit_c4_aci_hyperbola. Essentially, they are wrappers for optimizers from other libraries that serve to standardize their inputs and outputs.

Usage

```
optimizer_deoptim(itermax, VTR = -Inf)

optimizer_hjkb(tol, maxfeval = Inf, target = Inf)

optimizer_nlminb(rel.tol, eval.max = 200, iter.max = 200, abs.tol = 0)

optimizer_nmkb(tol, maxfeval = 2000, restarts.max = 10)

optimizer_null()
```

Arguments

tol	A convergence tolerance value; to be passed to nmkb or hjkb via their control input arguments. A typical value is 1e-7.
maxfeval	A maximum value for the number of function evaluations to allow during optimization; to be passed to nmkb or hjkb via their control input arguments.
target	A real number restricting the absolute function value; to be passed to hjkb via its control input argument.
rel.tol	A relative convergence tolerance value; to be passed to nlminb via its control input argument. A typical value is 1e-10.
eval.max	A maximum value for the number of function evaluations; to be passed to nlminb via its control input argument.
iter.max	A maximum value for the number of iterations; to be passed to nlminb via its control input argument.
abs.tol	An absolute convergence tolerance value; to be passed to nlminb via its control input argument.

restarts.max	A maximum value for the number of restarts allowed during optimization; to be passed to nmkb via its control input argument.
itermax	The maximum number of generations to be used; to be passed to DEoptim via its control input argument. Note that when VTR is -Inf, the optimizer will always use the maximum number of generations. A typical value is 200.
VTR	The value to be reached; to be passed to DEoptim via its control input argument.

Details

optimizer_deoptim is a wrapper for DEoptim.

optimizer_hjkb is a wrapper for hjkb.

optimizer_nlminb is a wrapper for nlminb.

optimizer_nmkb is a wrapper for nmkb.

optimizer_null simply returns the initial guess without doing any optimization; it can be useful for viewing initial guesses.

See the documentation for those functions for more information about how the optimizers work.

Value

Each of these functions returns an optimizer function optim_fun. The returned optim_fun function has four input arguments: an initial guess (guess), an error function (fun), lower bounds (lower), and upper bounds (upper). It returns a list with four named elements: par, convergence, feval, and convergence_msg.

Examples

```
# Here we just show examples of the optim_fun results. Other examples using the
# optimizers can be found throughout PhotoGEA, such as in the user guides and
# the documentation for fit_c3_aci, fit_c4_aci, etc.

optimizer_deoptim(200)

optimizer_hjkb(1e-7)

optimizer_nlminb(1e-7)

optimizer_nmkb(1e-7)

optimizer_null()
```

organize_response_curve_data *Reorganize response curve data for analysis and plotting*

Description

Prepares a set of response curves for future processing and analysis by numbering and reordering the points, (optionally) removing recovery points, and (optionally) calculating average values of key variables across each curve.

Usage

```
organize_response_curve_data(
  licor_exdf,
  identifier_columns,
  measurement_numbers_to_remove,
  column_for_ordering,
  ordering_column_tolerance = Inf,
  columns_to_average = c(),
  print_information = TRUE
)
```

Arguments

licor_exdf	An exdf object representing response curve data from a Licor gas exchange measurement system.
identifier_columns	A vector or list of strings representing the names of columns in licor_exdf that, taken together, uniquely identify each curve. This often includes names like plot, event, replicate, etc.
measurement_numbers_to_remove	A vector of integers specifying which points to remove from each curve; for example, if each curve has 16 points and the 10^th^ and 11^th^ points along the sequence should not be included in subsequent analysis, measurement_numbers_to_remove could be specified as c(10, 11). If measurement_numbers_to_remove is set to c(), no points will be removed.
column_for_ordering	The name of a column that is systematically varied to produce each curve; for example, in a light response curve, this would typically by Qin.
ordering_column_tolerance	To be passed to check_response_curve_data as the driving_column_tolerance input argument.
columns_to_average	A list of columns whose average values should be calculated; see below for details.
print_information	To be passed to check_response_curve_data.

Details

For an exdf object consisting of multiple response curves that can be identified using the values of its identifier_columns, this function performs the following actions:

- Assigns a sequential number to each measurement in each curve, beginning with 1. In other words, the first point in the curve is given number 1, the second is given number 2, etc. These numbers are stored as a new column called seq_num.
- (Optionally) extracts a subset of the data. If measurement_numbers_to_remove is c(), then this step will be skipped; otherwise, values of seq_num specified by measurement_numbers_to_remove will be removed, and then check_response_curve_data will be called to make sure the remaining points all follow the same sequence of setpoint values (within the tolerance set by ordering_column_tolerance), treating the column_for_ordering as the driving_column.
- Reorders the data according to ascending values of the column_for_ordering.
- (Optionally) calculates average values of important columns. If columns_to_average is c(), then this step will be skipped; otherwise, for each curve, the mean value of each column specified in columns_to_average will be stored in a new column whose name is based on the original column name, but with '_avg' added at the end. For example, the average value of the Qin column would be stored in Qin_avg.

Removing certain points is often helpful for A-Ci curves, where the CO~2~ concentration begins at the ambient value, is decreased to a low value, is reset to atmospheric for several measurements to allow the plant to reacclimate, and then is increased to higher values. In this case, only the first measurement at ambient CO~2~ is used for plotting or additional analysis, and the "recovery" points should be removed.

Reordering the points is often helpful for plotting. For example, the points in an A-Ci curve would not be ordered according to their Ci values in a curve measured using a sequence as described above. This can cause issues when making line plots, so it may be convenient to reorder them according to their Ci values.

Calculating average values of certain columns is especially useful for estimating Jmax values using calculate_jmax, since this operation requires average values of leaf temperature and incident photon flux across each curve.

Value

An exdf object based on licor_exdf but processed as described above.

Examples

```
# Read an example Licor file included in the PhotoGEA package and organize it.
# This file includes several 7-point light-response curves that can be uniquely
# identified by the values of its 'species' and 'plot' columns. Since these are
# light-response curves, each one follows a pre-set sequence of `Qin` values.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Split the data into individual curves, keep all seven measurement points in
# each curve, and order them by their incident light values (since these are
```

```
# light response curves). The curves were measured from high to low values of
# `Qin`, so after organizing the curves, their order will be reversed from the
# original version. Also add the average value of TleafCnd and Qin for each
# curve.
licor_file <- organize_response_curve_data(
  licor_file,
  c('species', 'plot'),
  c(),
  'Qin',
  columns_to_average = c('TleafCnd', 'Qin')
)

# View a subset of the data, including the new `seq_num` column
print(licor_file[, c('species', 'plot', 'seq_num', 'Qin', 'A', 'Qin_avg'), TRUE])
```

`pair_gasex_and_tdl` *Pair gas exchange and TDL data*

Description

Identifies the closest TDL cycle corresponding to each entry in the gas exchange data and adds the TDL data to the gas exchange data.

Usage

```
pair_gasex_and_tdl(
  gasex_exdf,
  tdl_exdf,
  max_allowed_time_difference = 1,
  gasex_timestamp_column_name = 'time',
  tdl_timestamp_column_name = 'TIMESTAMP'
)
```

Arguments

`gasex_exdf`	An `exdf` object representing data from a photosynthetic gas exchange measurement system.
`tdl_exdf`	An `exdf` object representing calibrated data from a tunable diode laser absorption spectroscopy system. Typically this is the output from applying `process_tdl_cycle_erml` or `process_tdl_cycle_polynomial` to a set of uncalibrated TDL data.
`max_allowed_time_difference`	The maximum time difference (in minutes) to allow between gas exchange and TDL timestamp values.
`gasex_timestamp_column_name`	The name of the column in `gasex_exdf` that contains the timestamp values.
`tdl_timestamp_column_name`	The name of the column in `tdl_exdf` that contains the timestamp values.

Details

When making combined gas exchange and isotope discrimination measurements using a portable photosynthetic gas exchange system (such as a Licor LI-6800) coupled with a tunable diode laser (TDL) absorption spectroscopy system, the TDL's gas handling system cycles through several gas lines (or sites) by opening and closing valves. When analyzing such data, a key step is to combine TDL and gas exchange data that were measured at the same times.

The `pair_gasex_and_tdl` function performs this operation by locating the TDL cycle whose timestamp is closest to each Licor file entry. Then, the 12C, 13C, total CO2, and delta_13C values measured by the TDL from the Licor's sample and reference lines during that cycle are added to the gas exchange data as new columns.

Value

An `exdf` object based on `gasex_exdf` that includes TDL values measured at the same times as the original gas exchange logs. Several new columns are added: `'cycle_num'`, `'tdl_time_s'`, `'calibrated_12c_s'`, `'calibrated_13c_s'`, `'total_CO2_s'`, `'delta_C13_s'`, `'tdl_time_r'`, `'calibrated_12c_r'`, `'calibrated_13c_r'`, `'total_CO2_r'`, and `'delta_C13_r'`. Variables with `'_s'` in the name refer to TDL measurements from the Licor sample line, and `'_r'` indicates the reference line. The category of each new column is `pair_gasex_and_tdl` to indicate that it was created using this function.

Examples

```
## In this example we load gas exchange and TDL data files, calibrate the TDL
## data, and pair the data tables together

# Read the TDL data file, making sure to interpret the time zone as US Central
# time
tdl_data <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_for_gm.dat'),
  'TIMESTAMP',
  list(tz = 'America/Chicago')
)

# Identify cycles within the TDL data
tdl_data <- identify_tdl_cycles(
  tdl_data,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Use reference tanks to calibrate the TDL data
processed_tdl <- consolidate(by(
  tdl_data,
  tdl_data[, 'cycle_num'],
  process_tdl_cycle_erml,
  noaa_valve = 2,
```

```
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
  calibration_3_valve = 26,
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
))

# Read the gas exchange data, making sure to interpret the time stamp in the US
# Central time zone
licor_data <- read_gasex_file(
  PhotoGEA_example_file_path('licor_for_gm_site11.xlsx'),
  'time',
  list(tz = 'America/Chicago')
)
# Get TDL valve information from Licor file name; for this TDL system, the
# reference valve is 12 when the sample valve is 11
licor_data <- get_sample_valve_from_filename(licor_data, list('11' = 12))

# Pair the Licor and TDL data by locating the TDL cycle corresponding to each
# Licor measurement
licor_data <- pair_gasex_and_tdl(licor_data, processed_tdl$tdl_data)

# View some of the results
licor_data[, c('A', 'delta_C13_r', 'delta_C13_s', 'total_CO2_r', 'total_CO2_s')]
```

`pdf_print` *Print a plot object or save it to a PDF*

Description

A convenience function that either displays a plot object in an R graphics window or saves it to a PDF.

Usage

```
pdf_print(
  plot_obj,
  width = 7,
  height = 7,
  save_to_pdf = FALSE,
  file = NULL,
  new_window = TRUE,
  ...
)
```

Arguments

plot_obj	A plotting object that can be printed, such as a trellis object returned by a call to xyplot.
width	The width of the figure in inches.
height	The width of the figure in inches.
save_to_pdf	When save_to_pdf is TRUE, plot_obj will be saved as a PDF; otherwise it will be printed to an R graphics window.
file	A file name to use when save_to_pdf is TRUE. If file is NULL, then the default value will be determined by the pdf function.
new_window	When printing plot_obj to an R graphics window, a new window will be created if new_window is TRUE. Otherwise, the plot will replace the currently active plot window (if one exists).
...	Additional arguments to be passed to pdf.

Details

This function is helpful when developing and using analysis scripts. In this context, it is recommended to define a boolean called SAVE_TO_PDF early in the script and to always use pdf_print when creating figures, passing the boolean as the save_to_pdf input argument. Figures can be initially displayed in R (setting SAVE_TO_PDF = FALSE), and then saved as PDFs once graphing parameters have been optimized (setting SAVE_TO_PDF = TRUE).

Note that calling pdf from the command line (as is done internally by pdf_print) is different than exporting an R graphics object as a PDF from RGui or RStudio. For some reason, RGui and RStudio override some of the pdf defaults and set useDingbats to TRUE. This setting almost always causes problems when opening the PDFs in software like Adobe Illustrator or Inkscape.

Value

The pdf_print function does not return anything.

Examples

```
SAVE_TO_PDF = FALSE # change this to TRUE to save to a PDF

pdf_print(
  lattice::xyplot(
    1:4 ~ 11:14,
    xlab = 'X',
    ylab = 'Y',
    type = 'b'
  ),
  save_to_pdf = SAVE_TO_PDF,
  file = 'example.pdf', # this name will only be used when saving to a PDF
  new_window = FALSE    # necessary for rendering the documentation examples
)
```

PhotoGEA *The PhotoGEA R package*

Description

PhotoGEA (short for **photo**synthetic **g**as **e**xchange **a**nalysis) is an R package that provides a suite of tools for loading, processing, and analyzing photosynthetic gas exchange data. See Lochocki, Salesse-Smith, & McGrath (2025) [doi:10.1111/pce.15501] for more information.

The best way to learn about using PhotoGEA is to visit the PhotoGEA website and click the Get Started link in the top menu bar. The website includes documentation for all the functions and data sets included in the package, as well as articles that describe its general features and several important use cases.

PhotoGEA_example_file_path
Locate a PhotoGEA example file on your computer

Description

A convenience function that locates examples files included with the PhotoGEA package (see example_data_files). This function is intended for use in PhotoGEA examples and documentation, and users should not need to use it in their own analysis scripts.

Usage

```
PhotoGEA_example_file_path(example_file_name)
```

Arguments

example_file_name
: The name of an example file included with the PhotoGEA package.

Details

The PhotoGEA package includes several instrument log files to use in examples and other documentation. A full list can be found in the article about example_data_files. When PhotoGEA is installed, these example files will be stored locally in the R package directory (in the PhotoGEA/extdata subdirectory), which will generally have a different path on every computer. The PhotoGEA_example_file_path function simply locates one of these files and returns its full file path.

When loading your own files for analysis, this function should not be used. Instead, either:

1. Directly write absolute file paths
2. Directly write relative file paths

3. Use one of the convenience functions from PhotoGEA to select files via a pop-up window, such as choose_input_licor_files

When directly writing relative file paths, consider using the file.path function from base R, which will ensure that the paths are properly formatted on any operating system. For example, instead of writing 'Documents\file.xlsx', write file.path('Documents', 'file.xlsx'). Doing this will make it easier to share your analysis scripts with other people who may be using different operating systems.

Value

A full path to a PhotoGEA example file.

Examples

```
PhotoGEA_example_file_path('c3_aci_1.xlsx')
```

---

plot_ball_berry_fit *Plot the results of a C3 CO2 response curve fit*

Description

Plots the output from fit_c3_aci or fit_c3_variable_j.

Usage

```
plot_ball_berry_fit(
  fit_results,
  identifier_column_name,
  bb_index_column_name = 'bb_index',
  gsw_column_name = 'gsw',
  ...
)
```

Arguments

fit_results	A list of three exdf objects names fits, parameters, and fits_interpolated, as calculated by fit_c3_aci.
identifier_column_name	The name of a column in each element of fit_results whose value can be used to identify each response curve within the data set; often, this is 'curve_identifier'.
bb_index_column_name	The name of the column in fit_results$fits that contains the Ball-Berry index in mol m^(-2) s^(-1); should be the same value that was passed to fit_ball_berry.
gsw_column_name	The name of the column in fit_results$fits that contains the stomatal conductance to water vapor in mol m^(-2) s^(-1); should be the same value that was passed to fit_ball_berry.
...	Additional arguments to be passed to xyplot.

Details

This is a convenience function for plotting the results of a Ball-Berry curve fit. It is typically used for displaying several fits at once, in which case `fit_results` is actually the output from calling `consolidate` on a list created by calling `by.exdf` with `FUN = fit_ball_berry`.

The resulting plot will show curves for the fitted `gsw`, along with points for the measured values of `gsw`.

Internally, this function uses `xyplot` to perform the plotting. Optionally, additional arguments can be passed to `xyplot`. These should typically be limited to things like `xlim`, `ylim`, `main`, and `grid`, since many other `xyplot` arguments will be set internally (such as `xlab`, `ylab`, `auto`, and others).

See the help file for `fit_ball_berry` for an example using this function.

Value

A `trellis` object created by `lattice::xyplot`.

Examples

```
# Read an example Licor file included in the PhotoGEA package, calculate
# additional gas properties, calculate the Ball-Berry index, define a new column
# that uniquely identifies each curve, and then perform a fit to extract the
# Ball-Berry parameters from each curve.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- calculate_total_pressure(licor_file)

licor_file <- calculate_gas_properties(licor_file)

licor_file[,'species_plot'] <-
  paste(licor_file[,'species'], '-', licor_file[,'plot'])

licor_file <- calculate_ball_berry_index(licor_file)

# Fit all curves in the data set
bb_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_ball_berry
))

# View the fits for each species / plot
plot_ball_berry_fit(bb_results, 'species_plot')
```

plot_c3_aci_fit *Plot the results of a C3 CO2 response curve fit*

Description

Plots the output from fit_c3_aci or fit_c3_variable_j.

Usage

```
plot_c3_aci_fit(
  fit_results,
  identifier_column_name,
  x_name,
  plot_operating_point = TRUE,
  plot_Ad = FALSE,
  a_column_name = 'A',
  cc_column_name = 'Cc',
  ci_column_name = 'Ci',
  ...
)
```

Arguments

fit_results	A list of three exdf objects named fits, parameters, and fits_interpolated, as calculated by fit_c3_aci.
identifier_column_name	The name of a column in each element of fit_results whose value can be used to identify each response curve within the data set; often, this is 'curve_identifier'.
x_name	The name of the column that should be used for the x-axis in the plot. This should refer to either 'Ci' or 'Cc', and it must be the same as ci_column_name or cc_column_name.
plot_operating_point	A logical value indicating whether to plot the operating point.
plot_Ad	A logical value indicating whether to plot the RuBP-depletion-limited net CO2 assimilation rate (Ad).
a_column_name	The name of the columns in the elements of fit_results that contain the net assimilation in micromol m^(-2) s^(-1); should be the same value that was passed to fit_c3_aci or fit_c3_variable_j.
cc_column_name	The name of the columns in the elements of fit_results that contain the chloroplastic CO2 concentration in micromol mol^(-1).
ci_column_name	The name of the columns in the elements of fit_results that contain the intercellular CO2 concentration in micromol mol^(-1); should be the same value that was passed to fit_c3_aci or fit_c3_variable_j.
...	Additional arguments to be passed to xyplot.

Details

This is a convenience function for plotting the results of a C3 A-Ci curve fit. It is typically used for displaying several fits at once, in which case `fit_results` is actually the output from calling `consolidate` on a list created by calling `by.exdf` with `FUN = fit_c3_aci` or `FUN = fit_c3_variable_j`.

The resulting plot will show curves for the fitted rates `An`, `Ac`, `Aj`, and `Ap`, along with points for the measured values of `A`, and (optionally) the estimated operating point. The x-axis can be set to either `Ci` or `Cc`.

Internally, this function uses `xyplot` to perform the plotting. Optionally, additional arguments can be passed to `xyplot`. These should typically be limited to things like `xlim`, `ylim`, `main`, and `grid`, since many other `xyplot` arguments will be set internally (such as `xlab`, `ylab`, `auto`, and others).

See the help file for `fit_c3_aci` for an example using this function.

Value

A `trellis` object created by `lattice::xyplot`.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# Calculate temperature-dependent values of C3 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c3_temperature_param_bernacchi)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)

# Fit all curves in the data set
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c3_aci,
```

```
  Ca_atmospheric = 420,
  optim_fun = optimizer
))

# View the fits for each species / plot
plot_c3_aci_fit(aci_results, 'species_plot', 'Ci')
```

`plot_c4_aci_fit` *Plot the results of a C4 CO2 response curve fit*

Description

Plots the output from `fit_c4_aci`.

Usage

```
plot_c4_aci_fit(
  fit_results,
  identifier_column_name,
  x_name,
  plot_operating_point = TRUE,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  pcm_column_name = 'PCm',
  ...
)
```

Arguments

`fit_results`	A list of three `exdf` objects named `fits`, `parameters`, and `fits_interpolated`, as calculated by `fit_c4_aci`.
`identifier_column_name`	The name of a column in each element of `fit_results` whose value can be used to identify each response curve within the data set; often, this is `'curve_identifier'`.
`x_name`	The name of the column that should be used for the x-axis in the plot. This should refer to either `'Ci'` or `'Cc'`, and it must be the same as `ci_column_name` or `cc_column_name`.
`plot_operating_point`	A logical value indicating whether to plot the operating point.
`a_column_name`	The name of the columns in the elements of `fit_results` that contain the net assimilation in `micromol m^(-2) s^(-1)`; should be the same value that was passed to `fit_c4_aci`.
`ci_column_name`	The name of the columns in the elements of `fit_results` that contain the intercellular CO2 concentration in `micromol mol^(-1)`; should be the same value that was passed to `fit_c4_aci`.

pcm_column_name
: The name of the columns in the elements of exdf_obj that contain the partial pressure of CO2 in the mesophyll, expressed in microbar.

...
: Additional arguments to be passed to xyplot.

Details

This is a convenience function for plotting the results of a C4 A-Ci curve fit. It is typically used for displaying several fits at once, in which case fit_results is actually the output from calling consolidate on a list created by calling by.exdf with FUN = fit_c4_aci.

The resulting plot will show curves for the fitted rates An, Apr, Apc, and Ar, along with points for the measured values of A, and (optionally) the estimated operating point. The x-axis can be set to either Ci or PCm.

Internally, this function uses xyplot to perform the plotting. Optionally, additional arguments can be passed to xyplot. These should typically be limited to things like xlim, ylim, main, and grid, since many other xyplot arguments will be set internally (such as xlab, ylab, auto, and others).

See the help file for fit_c4_aci for an example using this function.

Value

A trellis object created by lattice::xyplot.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Calculate temperature-dependent values of C4 photosynthetic parameters
licor_file <- calculate_temperature_response(licor_file, c4_temperature_param_vc)

# Calculate the total pressure in the Licor chamber
licor_file <- calculate_total_pressure(licor_file)

# For these examples, we will use a faster (but sometimes less reliable)
# optimizer so they run faster
optimizer <- optimizer_nmkb(1e-7)
```

```
# Fit all curves in the data set
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci,
  Ca_atmospheric = 420,
  optim_fun = optimizer
))

# View the fits for each species / plot
plot_c4_aci_fit(aci_results, 'species_plot', 'Ci', ylim = c(0, 100))
```

`plot_c4_aci_hyperbola_fit` *Plot the results of a hyperbolic C4 CO2 response curve fit*

Description

Plots the output from `fit_c4_aci_hyperbola`.

Usage

```
plot_c4_aci_hyperbola_fit(
  fit_results,
  identifier_column_name,
  a_column_name = 'A',
  ci_column_name = 'Ci',
  ...
)
```

Arguments

`fit_results`	A list of three `exdf` objects named `fits`, `parameters`, and `fits_interpolated`, as calculated by `fit_c4_aci_hyperbola`.
`identifier_column_name`	The name of a column in each element of `fit_results` whose value can be used to identify each response curve within the data set; often, this is `'curve_identifier'`.
`a_column_name`	The name of the columns in the elements of `fit_results` that contain the net assimilation in `micromol m^(-2) s^(-1)`; should be the same value that was passed to `fit_c4_aci_hyperbola`.
`ci_column_name`	The name of the columns in the elements of `fit_results` that contain the intercellular CO2 concentration in `micromol mol^(-1)`; should be the same value that was passed to `fit_c4_aci_hyperbola`.
`...`	Additional arguments to be passed to `xyplot`.

Details

This is a convenience function for plotting the results of a C4 A-Ci curve fit. It is typically used for displaying several fits at once, in which case `fit_results` is actually the output from calling `consolidate` on a list created by calling `by.exdf` with `FUN = fit_c4_aci_hyperbola`.

The resulting plot will show curves for the fitted rates `An`, `Ainitial`, and `Amax`, along with points for the measured values of `A`.

Internally, this function uses `xyplot` to perform the plotting. Optionally, additional arguments can be passed to `xyplot`. These should typically be limited to things like `xlim`, `ylim`, `main`, and `grid`, since many other `xyplot` arguments will be set internally (such as `xlab`, `ylab`, `auto`, and others).

See the help file for `fit_c4_aci_hyperbola` for an example using this function.

Value

A `trellis` object created by `lattice::xyplot`.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c4_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Fit all curves in the data set
aci_results <- consolidate(by(
  licor_file,
  licor_file[, 'species_plot'],
  fit_c4_aci_hyperbola
))

# View the fits for each species / plot
plot_c4_aci_hyperbola_fit(aci_results, 'species_plot', ylim = c(0, 100))
```

plot_laisk_fit *Plot the results of a C3 CO2 response curve fit*

Description

Plots the output from fit_laisk.

Usage

```
plot_laisk_fit(
  fit_results,
  identifier_column_name,
  plot_type,
  cols = multi_curve_colors(),
  a_column_name = 'A',
  ci_column_name = 'Ci',
  ppfd_column_name = 'PPFD',
  ...
)
```

Arguments

fit_results	A list of four exdf objects named first_fits, first_fit_parameters, second_fits, and second_fit_parameters, as calculated by fit_laisk.
identifier_column_name	The name of a column in each element of fit_results whose value can be used to identify each replicate within the data set; often, this is 'curve_identifier'.
plot_type	Must be either 'first' or 'second' (case-insensitive); determines which type of plot to create (see below for details).
cols	A vector of color specifications to use for each light level when plotting.
a_column_name	The name of the columns in the elements of fit_results that contain the net assimilation in micromol m^(-2) s^(-1); should be the same value that was passed to fit_laisk.
ci_column_name	The name of the column in the elements of fit_results that contain the intercellular CO2 concentration in micromol mol^(-1); should be the same value that was passed to fit_laisk.
ppfd_column_name	The name of the column in the elements of fit_results that can be used to split the data into individual response curves; should be the same value that was passed to fit_laisk.
...	Additional arguments to be passed to xyplot.

Details

This is a convenience function for plotting the results of a Laisk curve fit. It is typically used for displaying several fits at once, in which case `fit_results` is actually the output from calling `consolidate` on a list created by calling `by.exdf` with `FUN = fit_laisk`.

Because the Laisk fitting process involves two sets of linear fits, there are two possible graphs that can be created. When `plot_type` is `'first'`, this function will plot the individual A-Ci curves at each PPFD, along with the linear fits and the estimated intersection point. When `plot_type` is `'second'`, this function will plot the Laisk intercept vs. Laisk slope from the results of the first fits, along with a linear fit of Laisk intercept vs. Laisk slope. See `fit_laisk` for more details.

Internally, this function uses `xyplot` to perform the plotting. Optionally, additional arguments can be passed to `xyplot`. These should typically be limited to things like `xlim`, `ylim`, `main`, and `grid`, since many other `xyplot` arguments will be set internally (such as `xlab`, `ylab`, `auto`, and others).

See the help file for `fit_laisk` for an example using this function.

Value

A `trellis` object created by `lattice::xyplot`.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('c3_aci_1.xlsx')
)

# Define a new column that uniquely identifies each curve
licor_file[, 'species_plot'] <-
  paste(licor_file[, 'species'], '-', licor_file[, 'plot'] )

# Organize the data
licor_file <- organize_response_curve_data(
    licor_file,
    'species_plot',
    c(9, 10, 16),
    'CO2_r_sp'
)

# Apply the Laisk method. Note: this is a bad example because these curves were
# measured at the same light intensity, but from different species. Because of
# this, the results are not meaningful.
laisk_results <- fit_laisk(
  licor_file, 20, 150,
  ppfd_column_name = 'species_plot'
)

# Plot the linear fits of A vs. Ci
plot_laisk_fit(laisk_results, 'instrument', 'first', ppfd_column_name = 'species_plot')

# Plot the linear fits of Laisk intercept vs. Laisk slope
plot_laisk_fit(laisk_results, 'instrument', 'second', ppfd_column_name = 'species_plot')
```

print.exdf *Print the contents of an exdf object*

Description

Prints the contents of an `exdf` object's `main_data`. Each column is described by its name, unit, and category formatted like `name [category] (units)`.

Usage

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

Arguments

`x`	An `exdf` object.
`...`	Additional arguments to be passed to `print`.

Value

None.

See Also

`exdf`

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))
print(simple_exdf)
```

process_tdl_cycle_erml *Process cycles from the ERML TDL*

Description

Uses the 12C and 13C signal from the calibration lines of a tunable diode laser (TDL) to determine correction factors and apply them to the sample lines. Applicable for a system with a NOAA calibration tank, a nitrogen tank, and three other lines mixing the nitrogen with a CO2 tank in different ratios. This function is designed specifically for the TDL operating in Carl Bernacchi's lab in the Edward R. Madigan Laboratory (ERML) at the University of Illinois, Urbana-Champaign.

Usage

```
process_tdl_cycle_erml(
  tdl_cycle,
  noaa_valve,
  calibration_0_valve,
  calibration_1_valve,
  calibration_2_valve,
  calibration_3_valve,
  noaa_cylinder_co2_concentration,
  noaa_cylinder_isotope_ratio,
  calibration_isotope_ratio,
  valve_column_name = 'valve_number',
  raw_12c_colname = 'Conc12C_Avg',
  raw_13c_colname = 'Conc13C_Avg'
)
```

Arguments

`tdl_cycle`	An `exdf` object representing one cycle of TDL data.
`noaa_valve`	The valve number that corresponds to the NOAA reference cylinder.
`calibration_0_valve`	The valve number that corresponds to the calibration valve 0 (the nitrogen cylinder).
`calibration_1_valve`	The valve number that corresponds to the calibration valve 1 (a mixture of nitrogen gas with a calibrated CO2 source).
`calibration_2_valve`	The valve number that corresponds to the calibration valve 2 (a mixture of nitrogen gas with a calibrated CO2 source).
`calibration_3_valve`	The valve number that corresponds to the calibration valve 3 (a mixture of nitrogen gas with a calibrated CO2 source).
`noaa_cylinder_co2_concentration`	The total CO2 concentration of the NOAA calibration cylinder in ppm; this includes all carbon species, such as 12C18O18O.
`noaa_cylinder_isotope_ratio`	The isotope ratio of the NOAA calibration cylinder in ppt.
`calibration_isotope_ratio`	The isotope ratio of the other CO2 cylinder in ppt.
`valve_column_name`	The name of the column in `tdl_cycle` that contains the valve number; typically, this is `'valve_number'`.
`raw_12c_colname`	The name of the column in `tdl_cycle` that contains the 12C signal; typically, this is `'Conc12C_Avg'`.

raw_13c_colname
: The name of the column in tdl_cycle that contains the 13C signal; typically, this is 'Conc13C_Avg'.

Details

This function applies several corrections to the data in tdl_cycle:

- First, the 12C and 13C signals from the nitrogen line are considered to be additive offsets in the data. These values are subtracted from all measured 12C and 13C signals to produce "zero-corrected" values.
- The zero-corrected 12C signal from the NOAA calibration line is assumed to be related to the true 12C concentration in that line by a multiplicative "gain" factor. This factor is calculated using the known values of the NOAA cylinder's CO2 concentration and isotope ratio, and then applied to all the zero-corrected 12C signals to get "calibrated" 12C concentrations.
- The true 13C concentration in calibration lines 0-3 can be determined from the calibrated 12C concentration measurements and the known isotope ratio of the calibration tank. These true concentrations can be compared to the measured zero-corrected 13C signals to develop a correction function. Here we perform a third-order polynomial fit of expected vs. measured 13C values. (Four data points are used in the fit.) Then the fit result can be used to convert the zero-corrected 13C signals to "calibrated" 13C concentrations.

Should there be any equations here? Are there any references to cite?

This function assumes that tdl_cycle represents a single TDL measurement cycle. To process multiple cycles at once, this function is often used along with by.exdf and consolidate.

Value

A list with five elements:

- tdl_data: An exdf object containing the original content of tdl_cycle and several new columns: 'zero_corrected_12c', 'zero_corrected_13c', 'calibrated_12c', 'calibrated_13c', 'total_CO2', and 'delta_C13'.
- calibration_zero: An exdf object describing the values used to calculate the zero-corrected 12C and 13C signals.
- calibration_12CO2: An exdf object describing the gain factor used to calculate the calibrated 12C signal.
- calibration_13CO2_data: An exdf object describing the data used for the polynomial fit of expected vs. measured 13C signals from calibration valves 0-3.
- calibration_13CO2_fit: An exdf object describing the results of the polynomial fitting procedure.

Examples

```
# Example: reading a TDL file that is included with the PhotoGEA package,
# identifying its measurement cycles, and then processing them.
tdl_file <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_sampling_1.dat'),
  'TIMESTAMP'
```

```
)

# This is a large file; for this example, we will truncate to just the first
# 200 rows so it runs faster
tdl_file <- tdl_file[seq_len(200), , TRUE]

# Identify TDL cycles
tdl_file <- identify_tdl_cycles(
  tdl_file,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Process TDL cycles
processed_tdl <- consolidate(by(
  tdl_file,
  tdl_file[, 'cycle_num'],
  process_tdl_cycle_erml,
  valve_column_name = 'valve_number',
  noaa_valve = 2,
  calibration_0_valve = 20,
  calibration_1_valve = 21,
  calibration_2_valve = 23,
  calibration_3_valve = 26,
  raw_12c_colname = 'Conc12C_Avg',
  raw_13c_colname = 'Conc13C_Avg',
  noaa_cylinder_co2_concentration = 294.996,
  noaa_cylinder_isotope_ratio = -8.40,
  calibration_isotope_ratio = -11.505
))

# The output is a list of five exdf objects; four of them are related to each
# step in the calibration procedure for each TDL cycle
names(processed_tdl)

# The processed TDL data includes new columns for the calibrated CO2
# concentrations
colnames(processed_tdl$tdl_data)

# Make a plot of the raw and calibrated 13C signals across all the TDL cycles.
# Note that the calibrated signal from valve 20 is always exactly zero, since
# this is the line from the nitrogen tank. The calibrated signal from valve 2 is
# also constant since this is the line from the NOAA tank whose concentration is
# known.
lattice::xyplot(
  Conc13C_Avg + calibrated_13c ~ cycle_num | factor(valve_number),
  data = processed_tdl$tdl_data$main_data,
  type = 'l',
  auto = TRUE,
  grid = TRUE,
```

```
  xlab = 'TDL cycle',
  ylab = paste0('13C concentration (', processed_tdl$tdl_data$units$Conc13C_Avg, ')')
)

# Make a plot of 12C gain factor against elapsed time
lattice::xyplot(
  gain_12CO2 ~ elapsed_time,
  data = processed_tdl$calibration_12CO2$main_data,
  type = 'b',
  pch = 16,
  grid = TRUE,
 xlab = paste0('Elapsed time (', processed_tdl$calibration_12CO2$units$elapsed_time, ')'),
 ylab = paste0('12C gain factor (', processed_tdl$calibration_12CO2$units$gain_12CO2, ')')
)
```

`process_tdl_cycle_polynomial` *Process TDL cycles using a polynomial correction method*

Description

Uses the 12C and 13C signal from the calibration lines of a tunable diode laser (TDL) to determine correction factors and apply them to the sample lines. Applicable for a system with two or more reference tanks whose 12C and 13C concentrations are known beforehand.

Usage

```
process_tdl_cycle_polynomial(
  tdl_cycle,
  poly_order,
  reference_tanks,
  reference_tank_time_points = NA,
  valve_column_name = 'valve_number',
  raw_12c_colname = 'Conc12C_Avg',
  raw_13c_colname = 'Conc13C_Avg'
)
```

Arguments

`tdl_cycle`	An `exdf` object representing one cycle of TDL data.
`poly_order`	The order of the polynomial to fit, where 1 indicates a linear fit, 2 indicates a quadratic fit, etc. This argument will be passed to `stats::poly` during the fitting procedure.
`reference_tanks`	A list where each element is a list with three named elements: `valve`, `conc_12C`, and `conc_13C`. `valve` should indicate the valve number for the reference tank, and the other two elements should indicate the known concentrations of 12C and 13C in the tank.

`reference_tank_time_points`
: Either NA or a list where each element is a list with three named elements: `valve`, `start`, and `end`. `valve` should indicate the valve number for a reference tank, and the other two elements should indicate the first and last time points where the measurements from this valve should be averaged. The order of valves must be the same as in the `reference_tanks` input argument.

`valve_column_name`
: The name of the column in `tdl_cycle` that contains the valve number.

`raw_12c_colname`
: The name of the column in `tdl_cycle` that contains the 12C signal.

`raw_13c_colname`
: The name of the column in `tdl_cycle` that contains the 13C signal.

Details

This function applies a simple correction to the measured values of 12C and 13C. This correction is based on the fact that each reference tank has both a true concentration (which is known beforehand) and a measured concentration (from the TDL) of each isotope. Using this information, it is possible to perform a polynomial fit of true vs. measured concentrations; in other words, it is possible to identify a polynomial function that determines true concentrations from measured ones. This function can then be applied to tanks whose concentration is not known beforehand; in this case, it provides an estimate of the true concentration, otherwise referred to as a calibrated value.

When making dynamic TDL measurements, concentrations from some of the reference valves may be logged at multiple time points. In this case, it is typical to take an average value from a subset of them. `process_tdl_cycle_polynomial` can handle this situation when its `reference_tank_time_points` input argument is not NA.

This function assumes that `tdl_cycle` represents a single TDL measurement cycle. To process multiple cycles at once, this function is often used along with `by.exdf` and `consolidate`.

Value

A list with two elements:

- `tdl_data`: An `exdf` object containing the original content of `tdl_cycle` and several new columns: `'calibrated_12c'`, `'calibrated_13c'`, `'total_CO2'`, and `'delta_C13'`.
- `calibration_parameters`: An `exdf` object describing the fitted polynomial coefficients.

Examples

```
# Example 1: An example of a `reference_tank_time_points` list for a situation
# where there are just two reference valves (1 and 3)
reference_tank_time_points <- list(
 list(valve = 1, start = 101, end = 300), # Take an average of time points 101 - 300 for valve 1
 list(valve = 3, start = 201, end = 300)  # Take an average of time points 201 - 300 for valve 3
)

# Example2 : reading a TDL file that is included with the PhotoGEA package,
# identifying its measurement cycles, and then processing them.
tdl_file <- read_gasex_file(
```

```
  PhotoGEA_example_file_path('tdl_sampling_1.dat'),
  'TIMESTAMP'
)

# This is a large file; for this example, we will truncate to just the first
# 200 rows so it runs faster
tdl_file <- tdl_file[seq_len(200), , TRUE]

# Identify TDL cycles
tdl_file <- identify_tdl_cycles(
  tdl_file,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

# Process TDL cycles; note that the reference tank concentrations used in this
# example are not accurate, so the results are not meaningful
processed_tdl <- consolidate(by(
  tdl_file,
  tdl_file[, 'cycle_num'],
  process_tdl_cycle_polynomial,
  poly_order = 1,
  reference_tanks = list(
    list(valve = 23, conc_12C = 70.37507124, conc_13C = 0.754892652),
    list(valve = 26, conc_12C = 491.1854149, conc_13C = 5.269599965)
  )
))

# The output is a list of two exdf objects
names(processed_tdl)

# The calibration parameters include the coefficients of the polynomial fit for
# each cycle
colnames(processed_tdl$calibration_parameters)

# The processed TDL data includes new columns for the calibrated CO2
# concentrations
colnames(processed_tdl$tdl_data)
```

`read_cr3000` *Reading a CR3000 data file*

Description

Tool for reading output files created by Campbell Scientific CR3000 data loggers and storing their contents in `exdf` objects.

Usage

```
read_cr3000(
  file_name,
  rows_to_skip = 1,
  variable_name_row = 2,
  variable_unit_row = 3,
  data_start_row = 5,
  remove_NA_rows = TRUE,
  ...
)
```

Arguments

file_name	A relative or absolute path to a .dat file containing TDL data.
rows_to_skip	The number of rows to skip at the beginning of the file; the first row in a TDL file typically has fewer columns than the others, which causes problems when storing it as a table.
variable_name_row	The row number in the TDL file containing the names of the variables (RECORD, Conc12C_Avg, etc).
variable_unit_row	The row number in the TDL file containing the units of the variables (ppm, V, etc).
data_start_row	The first row number of the table containing the measured data.
remove_NA_rows	A logical value indicating whether to remove any rows whose values are all NA.
...	Additional arguments to be passed to read.csv.

Value

An exdf object that fully includes all the data from the CR3000 output file. In addition to the elements described in the documentation for read_gasex_file, the following "extra" elements are also included:

- rows_to_skip: A copy of the input argument with the same name
- variable_name_row: A copy of the input argument with the same name.
- variable_unit_row: A copy of the input argument with the same name.
- data_start_row: A copy of the input argument with the same name.

See Also

read_gasex_file

Examples

```
# Example: reading a TDL file that is included with the PhotoGEA package.
tdl_file <- read_cr3000(
  PhotoGEA_example_file_path('tdl_sampling_1.dat')
)

tdl_file$file_name # A record of where the data came from
str(tdl_file)      # View the contents of the exdf object's main_data
```

`read_gasex_file` *Reading a gas exchange log file*

Description

Tool for reading log files created by gas exchange measurement instruments and storing their contents in `exdf` objects.

Usage

```
read_gasex_file(
  file_name,
  timestamp_colname = NA,
  posix_options = list(),
  file_type = 'AUTO',
  instrument_type = 'AUTO',
  standardize_columns = TRUE,
  remove_NA_rows = TRUE,
  ...
)
```

Arguments

`file_name`	A relative or absolute path to a log file containing gas exchange data.
`timestamp_colname`	The name of the column that contains the timestamp of each measurement; typically, this is something like `'time'` or `'TIMESTAMP'`.
`posix_options`	Optional arguments to pass to `as.POSIXlt`; must be formatted as a list of named elements. See details below for more information.
`file_type`	The type of file to be loaded. If `file_type` is `'AUTO'`, then the file type will be automatically determined from the extension of `file_name`. The other supported options are `'plaintext'`, `'Excel'`, and `'data'`.
`instrument_type`	The type of measurement instrument that produced the log file. If `instrument_type` is `'AUTO'`, then the instrument type will be determined from the `file_type`. The other supported options are `'Licor LI-6800'` and `'CR3000'`.

`standardize_columns`
: A logical value indicating whether to standardize columns; see details below.

`remove_NA_rows`
: A logical value indicating whether to remove any rows whose values are all `NA`; this argument will be passed to the specialized reading functions; see below for more details.

`...`
: Additional arguments to be passed to specialized reading functions; see below for more details.

Details

Some log files contain Unicode characters in some column names and units, but these characters cannot be represented properly in R. To address this, Unicode characters are replaced with reasonable alternatives; for example, the character for the capital Greek letter delta is replaced with the word `Delta`. The replacement rules are stored in a data frame that can be accessed via `PhotoGEA:::UNICODE_REPLACEMENTS`, and more information can be found in the source code (`R/unicode_replacements.R`).

Sometimes it is useful to "standardize" the names, units, or categories of columns in instrument log files. This can be helpful in several situations:

- An instrument may not be consistent with the name of a column; for example, Licor LI-6800s may may have a `PhiPs2` or `PhiPS2` column depending on the version of the operating system running on the machine.
- An instrument may not specify the units of a column; for example, Licor LI-6800s do not specify that `PhiPS2` has units of `dimensionless`.
- An instrument may use different names or different units than another instrument for the same measured quantity.

To deal with these situations, it is possible to "standardize" the column names, units, and categories when reading an instrument file. A list of definitions for all standardizations can be accessed from an R session by typing `View(PhotoGEA:::gasex_column_conversions)`.

When reading a log file, it can be useful to identify the timestamp column so its values can be properly interpreted as `POSIXlt` objects. If `timestamp_colname` is `NA`, this conversion will be skipped. By default, `read_gasex_file` calls `as.POSIXlt` with `origin = '1970-01-01'` and `tz = ''`. With these options, any numeric timestamps (such as `1692386305.5`) will be interpreted as the number of seconds since January 1, 1970 (the UNIX standard) and the time will be expressed using the local system time. This works well in many situations. However, if a log file was created in a different time zone than the local one, it may be necessary to specify the time zone. This can be done via the `posix_options` argument. For example, to interpret the timestamp as a time in US Central time, set `posix_options = list(tz = 'US/Central')`. This may be necessary when using `pair_gasex_and_tdl` to match timestamps between different log files.

When automatically determining the file type from its extension, the following rules are used:

- A `.xlsx` extension corresponds to `file_type = 'Excel'`.
- A `.dat` extension corresponds to `file_type = 'data'`.
- A `.txt` extension or a file with no extension corresponds to `file_type = 'plaintext'`.

When automatically determining the instrument type from the file type, the following rules are used:

- File types of 'Excel' and 'plaintext' correspond to instrument_type = 'Licor LI-6800'.
- A file type of 'data' corresponds to instrument_type = 'CR3000'.

Internally, this function calls one of several other (non-exported) functions depending on the values of instrument_type and file_type:

- read_licor_6800_plaintext (for instrument_type = 'LI-6800' and file_type = 'plaintext')
- read_licor_6800_Excel (for instrument_type = 'LI-6800' and file_type = 'Excel')
- read_cr3000 (for instrument_type = 'CR3000' and file_type = 'data')

Any additional arguments specified via ... will be passed to these functions, along with the value of remove_NA_rows.

IMPORTANT NOTE ABOUT LICOR EXCEL FILES: by default, Licor Excel files do not "calculate" formula values. This causes a problem when reading them in R, since any data entry determined from a formula will be read as 0. To fix this issue for a Licor Excel file, open it in in Excel, go to the Formulas menu, and choose Calculate Now. (Alternatively, press F9.) Then save the file and close it. See read_licor_6800_Excel for more details.

Value

An exdf object that fully includes all the data from the log file. In addition to the required elements of an exdf object, the following "extra" elements are also included:

- file_name: A copy of the input argument with the same name.
- instrument_type: A copy of the input argument with the same name.
- file_type: A copy of the input argument with the same name, unless it was set to 'AUTO'; in that case, the file type that was determined from the file's extension.
- timestamp_colname: A copy of the input argument with the same name, unless it was set to 'AUTO'; in that case, the instrument type that was determined from the file type.

Examples

```
# Example: Eeading a Licor Excel file that is included with the PhotoGEA
# package. Here we specify 'time' as the name of the timestamp column.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx'),
  'time'
)

licor_file$file_name     # A record of where the data came from
str(licor_file)          # View the contents of the exdf object's main_data
str(licor_file$preamble) # View the Licor file's preamble data
```

`read_licor_6800_Excel` *Reading a Licor LI-6800 Excel log file*

Description

Tool for reading Excel log files created by Licor LI-6800 instruments and storing their contents in `exdf` objects.

Usage

```
read_licor_6800_Excel(
  file_name,
  column_name = 'obs',
  get_oxygen = TRUE,
  check_for_zero = c('A', 'gsw'),
  include_user_remark_column = TRUE,
  remove_NA_rows = TRUE,
  ...
)
```

Arguments

`file_name`	A relative or absolute path to an Excel file containing Licor data.
`column_name`	A column name that should be present in the log file; used to identify the beginning of the data block in the file.
`get_oxygen`	A logical value indicating whether to get the oxygen percentage from the file's preamble using `get_oxygen_from_preamble`.
`check_for_zero`	The names of columns whose values should not all be zero; see below for details.
`include_user_remark_column`	A logical value indicating whether to include the user remarks as a column; see below for details.
`remove_NA_rows`	A logical value indicating whether to remove any rows whose values are all `NA`.
`...`	Additional arguments; currently unused.

Details

Licor LI-6800 instruments create two types of log files: a plain-text file and an Excel file, each containing the same information. In general, the Excel files are much easier to modify, for example, deleting rows or adding new columns. For this reason, it is helpful to be able to read these files in R. Unfortunately, base R does not have any functionality for reading Excel files, so here the `openxlsx` package is used.

Excel log files typically have two sheets called `Measurements` and `Remarks`. The `Measurements` sheet contains the preamble and the main data logs, and if `read_licor_6800_Excel` does not find a sheet called `Measurements`, it will send an error message.

Then, read_licor_6800_Excel looks for a particular data column (column_name) in order to identify the start of the data table within the contents of the Measurements sheet. Rows above the main data table are assumed to be part of the preamble; these are converted to columns and appended to the end of the columns in the main data table.

"Calculating" formula values: By default, Licor Excel files do not "calculate" formula values. This causes a problem when reading them in R, since any data entry determined from a formula will be read as 0. To fix this issue for a Licor Excel file, open it in in Excel, go to the Formulas menu, and choose Calculate Now. (Alternatively, press F9.) Then save the file and close it. See these articles for more information about this issue:

- GitHub issue 261 from the openxlsx package
- GitHub issue 863 from the openxlsx2 package
- GitHub issue 495 from the readxl package

read_licor_6800_Excel attempts to detect this issue by checking the values of key columns (specified by the check_for_zero input argument). If any of these columns are all 0, then an error message will be sent. This feature can be disabled by setting check_for_zero = c() when calling read_licor_6800_Excel or read_gasex_file.

User remarks: When operating a Licor LI-6800, it is possible to make a "remark." Each remark will appear in the Remarks sheet of an Excel log file on its own line, where the entry in the first column is an HH:MM:SS time, and the second column contains the remark text. The read_licor_6800_Excel function identifies these user remarks and includes them in the return as an "extra" element called user_remarks. Note that changing stability criteria will also generate a user remark with a message describing the new stability settings. Also note that the "remarks" tab includes other automatically generated entries, such as the instrument serial number; these entries are included with the "preamble" in the output from read_licor_6800_Excel.

When include_user_remark_column is TRUE, these user remarks will be included in the main data table as a column called user_remark. For each row in the table, the entry in the user_remark column will be set to the most recent user remark.

The user remark system is prone to errors, especially since changes to stability settings are recorded in the log files using the exact same format as true user remarks. In general, it is better to record metadata about measurements via user constants rather than user remarks.

Changing Oxygen Concentration: When operating a Licor LI-6800, it is possible to change the oxygen concentration while a log file is open. In that case, the new value is entered via two rows in the main data table. The read_licor_6800_Excel function is able to handle this, and the new oxygen concentration will be included in the Oxygen column.

User constants as rows: When operating a Licor LI-6800, it is possible to include user constants as either rows or columns. In general, it is better to include them as columns. Although the read_licor_6800_Excel function is currently able to handle oxygen concentrations logged as rows, it is not known whether user constants will be properly read.

Value

An exdf object that fully includes all the data from the Licor Excel file. In addition to the elements described in the documentation for read_gasex_file, the following "extra" elements are also included:

- `preamble`: A data frame containing the "preamble" information from the file; this is also included in the main data.
- `remarks`: A data frame containing the "remarks" from the Remarks sheet, if present. These remarks are automatically generated by the machine, and are separate from the user remarks discussed elsewhere.
- `user_remarks`: A data frame containing any user remarks from the file. The data frame has two columns for the timestamp and the value, called `remark_time` and `remark_value`, respectively.
- `data_row`: The line of the file where the column name was found.

See Also

`read_gasex_file`

Examples

```
# Example 1: Reading a Licor Excel file that is included with the PhotoGEA
# package and viewing some of the "extra" information associated with the file
licor_file <- read_licor_6800_Excel(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

str(licor_file$preamble)

str(licor_file$remarks)

print(licor_file$user_remarks)

# Example 2: Reading a Licor Excel file that is included with the PhotoGEA
# package; here we use a different column name to identify the data block within
# the file's contents.
licor_file <- read_licor_6800_Excel(
  PhotoGEA_example_file_path('ball_berry_1.xlsx'),
  column_name = 'A'
)
```

---

read_licor_6800_plaintext

Reading a Licor LI-6800 plaintext log file

Description

Tool for reading plaintext log files created by Licor LI-6800 instruments and storing their contents in `exdf` objects.

Usage

```
read_licor_6800_plaintext(
  file_name,
  get_oxygen = TRUE,
  include_user_remark_column = TRUE,
  remove_NA_rows = TRUE,
  ...
)
```

Arguments

file_name	A relative or absolute path to a plaintext file containing Licor data.
get_oxygen	A logical value indicating whether to get the oxygen percentage from the file's preamble using get_oxygen_from_preamble.
include_user_remark_column	A logical value indicating whether to include the user remarks as a column; see below for details.
remove_NA_rows	A logical value indicating whether to remove any rows whose values are all NA.
...	Additional arguments; currently unused.

Details

Licor LI-6800 instruments create two types of log files: a plaintext file and an Excel file, each containing the same information. The plaintext files are the only ones guaranteed to be created, since the Excel files require the user to select an option to create them.

read_licor_6800_plaintext looks for two special lines in the Licor log file: the [Head] line indicates the beginning of the header, and the [Data] line indicates the beginning of the data table. If these lines are missing from the file, it will not be loaded properly.

The header information generally consists of two types of lines. One type of line begins with a string formatted like category:name. These lines form the "preamble," and these rows are converted to columns and appended to the end of the main data frame. The remaining lines in the header form the "remarks." These are automatically generated by the machine, and are distinct from the "user remarks" discussed elsewhere.

Closing and reopening a log file: When operating a Licor LI-6800, it is possible to close and then reopen a log file. Doing this causes the plaintext log file to contain multiple [Head] and [Data] sections. This function is able to handle such files.

User remarks: When operating a Licor LI-6800, it is possible to make a "remark." Each remark will appear in the plaintext log file in its own line, which begins with an HH:MM:SS time and then contains the remark text. The read_licor_6800_plaintext function identifies these user remarks and includes them in the return as an "extra" element called user_remarks. Note that changing stability criteria will also generate a user remark with a message describing the new stability settings.

When include_user_remark_column is TRUE, these user remarks will be included in the main data table as a column called user_remark. For each row in the table, the entry in the user_remark column will be set to the most recent user remark.

The user remark system is prone to errors, especially since changes to stability settings are recorded in the log files using the exact same format as true user remarks. In general, it is better to record metadata about measurements via user constants rather than user remarks.

Changing Oxygen Concentration: When operating a Licor LI-6800, it is possible to change the oxygen concentration while a log file is open. In that case, the new value is entered via two rows in the main data table. The `read_licor_6800_plaintext` function is able to handle this, and the new oxygen concentration will be included in the `Oxygen` column.

User constants as rows: When operating a Licor LI-6800, it is possible to include user constants as either rows or columns. In general, it is better to include them as columns. Although the `read_licor_6800_plaintext` function is currently able to handle oxygen concentrations logged as rows, it is not known whether user constants will be properly read.

Value

An `exdf` object that fully includes all the data from the Licor Excel file. In addition to the elements described in the documentation for `read_gasex_file`, the following "extra" elements are also included:

- `preamble`: A data frame containing the "preamble" information from the file.
- `remarks`: A data frame containing the "remarks." These remarks are automatically generated by the machine, and are separate from the user remarks discussed elsewhere.
- `user_remarks`: A data frame containing any user remarks from the file. The data frame has two columns for the timestamp and the value, called `remark_time` and `remark_value`, respectively.

See Also

`read_gasex_file`

Examples

```
# Example: Reading a Licor plaintext file that is included with the PhotoGEA
# package and viewing some of the "extra" information associated with the file
licor_file <- read_licor_6800_plaintext(
  PhotoGEA_example_file_path('plaintext_licor_file')
)

str(licor_file$preamble)

str(licor_file$remarks)

print(licor_file$user_remarks)
```

remove_points *Remove specific points from an exdf object*

Description

Removes all points from an exdf object that satisfy a set of conditions.

Usage

```
remove_points(exdf_obj, ..., method = 'remove')
```

Arguments

exdf_obj	An exdf object.
...	Each optional argument should be a list of named elements that specify points to be removed from exdf_obj. For example, list(species = 'soybean', plot = c('1a', '1b')) specifies the set of points where (1) species is 'soybean' and (2) plot is '1a' or '1b'.
method	Specify whether to remove points ('remove') or designate them as being excluded from subsequent fits ('exclude'); see below for more details.

Value

This function returns an exdf object formed from exdf_obj, where the result depends on the value of method.

When method is 'remove', the returned object is a modified copy of exdf_obj where all rows that meet the conditions specified by the optional arguments have been removed.

When method is 'exclude', the returned object is a modified copy of exdf_obj with a new column called include_when_fitting. The value of this column is FALSE for all rows that meet the conditions specified by the optional arguments, and TRUE otherwise. Points where this column is FALSE will not be used for fitting by fit_c3_aci or other fitting functions.

See Also

exdf

Examples

```
# Create an exdf object by reading a Licor Excel file
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Print the number of points in the data set
nrow(licor_file)

# Remove the following:
```

```
# - All points where `obs` is 28 (1 point)
# - All points where `species` is `soybean` and `plot` is `1a` or `1b` (14 points)
licor_file_2 <- remove_points(
  licor_file,
  list(obs = 28),
  list(species = 'soybean', plot = c('1a', '1b')),
  method = 'remove'
)

# There should now be 15 fewer points remaining in the data set
nrow(licor_file_2)

# We can also specify the same points for exclusion rather than removal:
licor_file_3 <- remove_points(
  licor_file,
  list(obs = 28),
  list(species = 'soybean', plot = c('1a', '1b')),
  method = 'exclude'
)

print(licor_file_3[, c('species', 'plot', 'include_when_fitting')])

# The number of points where `include_when_fitting` is TRUE should be the same
# as the number of remaining rows when using the `remove` method
sum(licor_file_3[, 'include_when_fitting'])
```

`residual_stats` *Calculate statistics that describe the residuals of a fit*

Description

Calculates several key statistics from the residuals of of a fit: the residual sum of squares (`RSS`), the mean squared error (`MSE`), the root mean squared error (`RMSE`), the residual standard error (`RSE`), and the Akaike information criterion (`AIC`). This function is used internally by all fitting functions in the `PhotoGEA` package, such as `fit_ball_berry` and `fit_c3_aci`.

Usage

```
residual_stats(fit_residuals, units, nparam)
```

Arguments

`fit_residuals`	A numeric vector representing the residuals from a fit, i.e., the differences between the measured and fitted values.
`units`	A string expressing the units of the residuals.
`nparam`	The number of free parameters that were varied when performing the fit.

Details

This function calculates several model-independent measures of the quality of a fit. The basis for these statistics are the `residuals` (also known as the `errors`). If the measured values of a quantity `y` are given by `y_measured` and the fitted values are `y_fitted`, then the residuals are defined to be `residual = y_measured - y_fitted`. The key statistics that can be calculated from the residuals are as follows:

- The residual sum of squares (`RSS`) is also known as the sum of squared errors (`SSE`). As its name implies, it is simply the sum of all the squared residuals: `RSS = sum(residuals^2)`.
- The mean squared error (`MSE`) is the mean value of the squared residuals: `MSE = sum(residuals^2) / n = RSS / n`, where `n` is the number of residuals.
- The root mean squared error (`RMSE`) is the square root of the mean squared error: `RMSE = sqrt(MSE) = sqrt(RSS / n)`.
- The residual standard error `RSE` is given by `RSE = sqrt(RSS / dof)`, where `dof = n - nparam` is the number of degrees of freedom involved in the fit.
- The Akaike information criterion `AIC` is given by `AIC = npts * (log(2 * pi) + 1) + npts * log(MSE) + 2 * (nparam + 1)`.

For a given model, the `RMSE` is usually a good way to compare the quality of different fits. When trying to decide which model best fits the measured data, the `AIC` may be a more appropriate metric since it controls for the number of parameters in the model.

The AIC definition used here is appropriate for the results of maximum likelihood fitting with equal variance, or minimum least squares fitting. For more details about the AIC equation above and its relation to the more general definition of AIC, see Section 2 of Banks & Joyner (2017).

References:

Banks, H. T. & Joyner, M. L. "AIC under the framework of least squares estimation." Applied Mathematics Letters 74, 33–45 (2017) [doi:10.1016/j.aml.2017.05.005].

Value

An `exdf` object with one row and the following columns: `npts` (the number of residual values), `nparam`, `dof`, `RSS`, `MSE`, `RMSE`, `RSE`, `AIC`.

Examples

```
# Generate some random residuals
residuals <- runif(10, -1, 1)

# Calculate residual stats as if these values had units of `kg` and were related
# to a model with 3 free parameters
residual_stats(residuals, 'kg', 3)
```

`set_variable` *Set values, units, and categories for a column in a table*

Description

Sets the value, units, and/or category of a new or existing column of a table-like object.

Usage

```
set_variable(
  data_table,
  name,
  units = NULL,
  category = NULL,
  value = NA,
  id_column = NULL,
  value_table = NULL
)
```

Arguments

`data_table`	A table-like R object such as a data frame or an `exdf`.
`name`	The name of the column to be added to `data_table`.
`units`	The units of the column to be added to `data_table`.
`category`	The category of the column to be added to `data_table`.
`value`	The value of the column to be added to `data_table`.
`id_column`	The name of an identifier column in `data_table`.
`value_table`	A list of named elements, where the name of each element is a possible value of the `id_column` and the value of each element is the corresponding value that the `name` column should take.

Details

There are two main "modes" for setting the value of the new column: it can be set to a fixed value (using the `value` input argument), or it can be set according to the values of another column (using the `id_column` and `value_table` input arguments). The latter method is useful when different values must be specified for different treatments within the data set.

In greater detail, this function attempts to set the value of a new or existing column in an `exdf` object according to the following rules:

- The value of the `name` column of `data_table` will be set to `value`; this assignment follows the usual rules; in other words, `value` could be a single value or a vector of length `nrow(data_table)`.
- If `units` and `categories` are both `NULL`, the units and category will not be specified. In this case, if the `name` column already exists, its units and category will remain the same; if the `name` column is new, it will be initialized with `NA` for its units and category.

- If either `units` _or_ `category` is not `NULL`, the units and category for the `name` column _will_ be specified. In this case, if one of `units` or `category` _is_ `NULL`, its value will be set to `NA`.
- If `id_column` is not `NULL`, then the `value_table` will be used to set different values of the `name` column for each specified value of `id_column`. For example, if `id_column` is `species` and `value_table = list(soybean = 1, tobacco = 2)`, then the `name` column will be set to `1` when `species` is `'soybean'` and `2` when `species` is `'tobacco'`. For any other values of species (such as `'maize'`), the value of `name` will still be `value`. **Note**: values of the `id_column` will be converted using `as.character` before making comparisons.

For other table-like objects, such as data frames, only the values will be set, and the units and categories will be ignored.

Value

An object based on `data_table` with new and/or modified columns.

See Also

exdf

Examples

```
# Create a simple exdf object with two columns (`A` and `B`) and default values
# for its units and categories.
simple_exdf <- exdf(data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8)))

print(simple_exdf)

# Add a new column called 'C' with units 'u1' and category 'cat1' whose value is
# 1000.
simple_exdf <- set_variable(simple_exdf, 'C', 'u1', 'cat1', 1000)

# Set the value of the 'B' column to 2000 when 'A' is 3, to 3000 when 'A' is 9,
# and to 4000 for all other values of 'A'. Do not modify its units or category.
simple_exdf <- set_variable(
  simple_exdf,
  'B',
  value = 4000,
  id_column = 'A',
  value_table = list('3' = 2000, '9' = 3000)
)

print(simple_exdf)

# Take the same operations, but using a data frame instead
simple_df <- data.frame(A = c(3, 2, 7, 9), B = c(4, 5, 1, 8))

simple_df <- set_variable(simple_exdf$main_data, 'C', 'u1', 'cat1', 1000)

simple_df <- set_variable(
  simple_df,
  'B',
```

```
  value = 4000,
  id_column = 'A',
  value_table = list('3' = 2000, '9' = 3000)
)

print(simple_df)

# As a more realistic example, load a Licor file and set different values of
# mesophyll conductance for each species in the data set.
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- set_variable(
  licor_file,
  'gmc',
  'mol m^(-2) s^(-1) bar^(-1)',
  '',
  id_column = 'species',
  value_table = list(soybean = 0.9, tobacco = 1.1)
)

print(licor_file[, c('species', 'gmc'), TRUE])
```

`smooth_tdl_data` *Smoothing data from one TDL valve*

Description

Tool for applying a smoothing function to the time series corresponding to measurements from a single valve in a tunable diode laser (TDL) data set.

Usage

```
smooth_tdl_data(
  tdl_exdf,
  column_to_be_smoothed,
  valve_column_name,
  valve_number,
  smoothing_function
)
```

Arguments

`tdl_exdf`	An `exdf` object representing data from a TDL data logger.
`column_to_be_smoothed`	The name of the column in `tdl_exdf` that contains the data to be smoothed; typically, this is `'Conc12C_Avg'` or `'Conc12C_Avg'`.

valve_column_name
: The name of the column in tdl_exdf that contains the valve number; typically, this is 'valve_number'.

valve_number
: The value of the valve_column_name column that indicates the valve to be smoothed.

smoothing_function
: A function that accepts two vectors Y and X (in that order) and returns a smoothed version of Y(X); typically, smoothing_function is based on smooth.spline or a filter from the signal package.

Details

The output from a TDL is highly sensitive to electronic and atmospheric noise, and it is often helpful to smooth the data from one or more valves before attempting to apply calibration corrections or determine the content of an unknown gas mixture. smooth_tdl_data is a convenience function that extracts a time series corresponding to data from one valve, applies a smoothing operation, and replaces the original data in tdl_exdf with the smoothed version. The smoothing function is user-supplied to allow more flexbility.

In addition to the column_to_be_smoothed and valve_column_name columns, the tdl_exdf must also contain an 'elapsed_time' column, which is typically created by a call to identify_tdl_cycles.

Value

An exdf object based on tdl_exdf, where the time series of column_to_be_smoothed vs. 'elapsed_time' has been replaced by a smoothed version obtained by applying the smoothing_function.

Examples

```
# Example: Smoothing the 12C signal from one TDL valve using a spline fit
tdl_file <- read_gasex_file(
  PhotoGEA_example_file_path('tdl_sampling_1.dat'),
  'TIMESTAMP'
)

tdl_file <- identify_tdl_cycles(
  tdl_file,
  valve_column_name = 'valve_number',
  cycle_start_valve = 20,
  expected_cycle_length_minutes = 2.7,
  expected_cycle_num_valves = 9,
  timestamp_colname = 'TIMESTAMP'
)

spline_smoothing_function <- function(Y, X) {
    ss <- smooth.spline(X, Y)
    return(ss$y)
}

spline_smoothed_tdl_file <- smooth_tdl_data(
  tdl_file, 'Conc12C_Avg', 'valve_number', 20, spline_smoothing_function
)
```

split.exdf *Divide an exdf object into groups*

Description

Divides an exdf object into groups defined by one or more factors.

Usage

```
  ## S3 method for class 'exdf'
split(x, f, drop = FALSE, lex.order = FALSE, ...)
```

Arguments

x	An exdf object.
f	A factor or a list of factors.
drop	A logical value indicating whether levels of f that do not occur should be dropped.
lex.order	A logical value passed to interaction.
...	Additional arguments to be passed to the default method of split.

Value

Returns a list of exdf objects created by splitting x along the values of f.

See Also

exdf

Examples

```
# Read a Licor file, select just a few columns, and then split it by the value
# of the `plot` column
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

licor_file <- licor_file[, c('plot', 'species', 'Qin', 'A', 'gsw'), TRUE]

split(
  licor_file,
  list(licor_file[,'species'], licor_file[,'plot']),
  drop = TRUE
)
```

str.exdf *Display the structure of an exdf object*

Description

Displays the structure of an `exdf` object's `main_data`. Each column is described by its name, unit, and category formatted like `name [category] (units)`.

Usage

```
  ## S3 method for class 'exdf'
str(object, ...)
```

Arguments

`object`	An `exdf` object.
`...`	Additional arguments to be passed to `str`.

Value

None.

See Also

`exdf`

Examples

```
simple_exdf <- exdf(data.frame(A = 1), data.frame(A = 'u'), data.frame(A = 'c'))
str(simple_exdf)
```

xyplot_avg_rc *Plot average response curves with error bars*

Description

A wrapper for `lattice::xyplot` that plots average response curves with error bars.

Usage

```
xyplot_avg_rc(
  Y,
  X,
  point_identifier,
  group_identifier,
  y_error_bars = TRUE,
  x_error_bars = FALSE,
  cols = multi_curve_colors(),
  eb_length = 0.05,
  eb_lwd = 1,
  na.rm = TRUE,
  subset = rep_len(TRUE, length(Y)),
  ...
)
```

Arguments

Y	A numeric vector of y-values.
X	A numeric vector of x-values with the same length as Y
point_identifier	A vector with the same length as Y that indicates the location of each (x, y) pair along the response curve; typically this is the seq_num column of an exdf object.
group_identifier	A vector with the same length as Y that indicates the "group" of each response curve.
y_error_bars	A logical value indicating whether to plot y-axis error bars.
x_error_bars	A logical value indicating whether to plot x-axis error bars.
cols	A vector of color specifications.
eb_length	The width of the error bars.
eb_lwd	The line width (thickness) of the error bars.
na.rm	A logical value indicating whether or not to remove NA values before calculating means and standard errors.
subset	A logical vector (of the same length as Y) indicating which points to include in the final plot.
...	Additional arguments to be passed to lattice::xyplot.

Details

This function calculates average values of X and Y at each value of the point_identifier across groups defined by group_identifier, and then uses these values to plot average response curves for each group. Error bars are determined by calculating the standard errors of X and Y at each value of the point_identifier across groups defined by group_identifier.

If points were excluded from the data set using remove_points with method = 'exclude', then the include_when_fitting column should be passed to xyplot_avg_rc as the subset input argument; this will ensure that the excluded points are not used when calculating average response curves.

Value

A trellis object created by lattice::xyplot.

Examples

```
# Read an example Licor file included in the PhotoGEA package
licor_file <- read_gasex_file(
  PhotoGEA_example_file_path('ball_berry_1.xlsx')
)

# Organize the response curve data
licor_file <- organize_response_curve_data(
  licor_file,
  c('species', 'plot'),
  c(),
  'Qin'
)

# Plot the average light response curve for each species (here there is only one
# curve for tobacco, so there are no tobacco error bars)
xyplot_avg_rc(
  licor_file[, 'A'],
  licor_file[, 'Qin'],
  licor_file[, 'seq_num'],
  licor_file[, 'species'],
  ylim = c(0, 50),
  xlab = paste0('Incident PPFD (', licor_file$units$Qin, ')'),
  ylab = paste0('Average net assimilation (', licor_file$units$A, ')'),
  auto = TRUE,
  grid = TRUE
)

# Exclude a few points from the data set and re-plot the average curves
licor_file <- remove_points(
  licor_file,
  list(obs = c(5, 10, 18)),
  method = 'exclude'
)

xyplot_avg_rc(
  licor_file[, 'A'],
  licor_file[, 'Qin'],
  licor_file[, 'seq_num'],
  licor_file[, 'species'],
  subset = licor_file[, 'include_when_fitting'],
  ylim = c(0, 50),
```

```
  xlab = paste0('Incident PPFD (', licor_file$units$Qin, ')'),
  ylab = paste0('Average net assimilation (', licor_file$units$A, ')'),
  auto = TRUE,
  grid = TRUE
)
```

Index