Package ‘photosynthesis’

May 11, 2023

Version 2.1.3
Date 2023-05-11
Title Tools for Plant Ecophysiology & Modeling
Depends R (>= 4.0.0), ggplot2 (>= 3.4.0), minpack.lm (>= 1.2-1), units (>= 0.6.6)
Imports checkmate (>= 2.0.0), crayon (>= 1.3.4), dplyr (>= 0.8.5), furrr (>= 0.1.0), glue (>= 1.4.0), graphics (>= 4.0.0), grDevices (>= 4.0.0), gunit (>= 1.0.2), lifecycle (>= 1.0.0), magrittr (>= 1.5.0), methods (>= 3.5.0), nlme (>= 3.1-147), progress (>= 1.2.0), purrr (>= 0.3.3), readr (>= 2.0.0), rlang (>= 0.4.6), stats (>= 4.0.0), stringr (>= 1.4.0), tealeaves (>= 1.0.5), utils (>= 4.0.0)
Suggests brms, broom, future, knitr, rmarkdown, testthat, tibble, tidyr, tidyselect

Description Contains modeling and analytical tools for plant ecophysiology.
   MODELING: Simulate C3 photosynthesis using the Farquhar, von Caemmerer, Berry (1980) <doi:10.1007/BF00386231> model as described in Buckley and Diaz-Espejo (2015) <doi:10.1111/pce.12459>. It uses units to ensure that parameters are properly specified and transformed before calculations. Temperature response functions get automatically "baked" into all parameters based on leaf temperature following Bernacchi et al. (2002) <doi:10.1104/pp.008250>. The package includes boundary layer, cuticular, stomatal, and mesophyll conductances to CO2, which each can vary on the upper and lower portions of the leaf. Use straightforward functions to simulate photosynthesis over environmental gradients such as Photosynthetic Photon Flux Density (PPFD) and leaf temperature, or over trait gradients such as CO2 conductance or photochemistry.

License  MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.2.3
VignetteBuilder knitr
URL https://github.com/cdmuir/photosynthesis
BugReports https://github.com/cdmuir/photosynthesis/issues
NeedsCompilation no
Author Joseph Stinziano [aut] (<https://orcid.org/0000-0002-7628-4201>), Cassandra Roback [aut], Demi Sargent [aut], Bridget Murphy [aut], Patrick Hudson [aut, dnc], Chris Muir [aut, cre] (<https://orcid.org/0000-0003-2555-3878>)
Maintainer Chris Muir <cdmuir@hawaii.edu>
Repository CRAN
Date/Publication 2023-05-11 21:20:02 UTC

R topics documented:

- photosynthesis-package .................................................. 3
- analyze_sensitivity .......................................................... 4
- aq_response ................................................................. 6
- A_supply ................................................................. 7
- bake ................................................................. 8
- baked-class ............................................................. 10
- bake_par ............................................................. 10
- calculated-parameters .................................................. 11
- calculate_jmax ......................................................... 11
- CO2_conductance ......................................................... 12
- compile_data ............................................................. 15
- compute_sensitivity ..................................................... 16
- constants ............................................................ 18
## Description

Tools for Plant Ecophysiology & Modeling

## Details

See the README on [GitHub](https://github.com/photosynthesis-package/photonsynthesis-package)
analyze_sensitivity  Running 2-parameter sensitivity analyses

Description

Running 2-parameter sensitivity analyses

Usage

analyze_sensitivity(
  data,
  funct,
  test1 = NA,
  values1,
  test2 = NA,
  values2,
  element_out = 1,
  ...
)

Arguments

data        Dataframe
funct       Function to use - do not use parentheses
test1       Input parameter to vary and test
values1     Values of test1 to use
test2       Input parameter to vary and test
values2     Values of test2 to use
element_out List element to compile
...
          Additional arguments required for the function

Value

analyze_sensitivity runs a 2-parameter sensitivity analysis. Note that any parameter value combinations that break the input function WILL break this function. For 1-parameter sensitivity analysis, use test1 only.

Examples

# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata","A_Ci_Q_data_1.csv", package = "photosynthesis")
# Define a grouping factor based on light intensity to split the ACi curves
data$Q_2 <- as.factor(round(data$Qin, digits = 0))

# Convert leaf temperature to K
data$T_leaf <- data$Tleaf + 273.15

# Run a sensitivity analysis on gamma_star and mesophyll conductance at 25 Celsius for one individual curve
pars <- analyze_sensitivity(
  data = data[data$Q_2 == 1500, ],
  funct = fit_aci_response,
  varnames = list(
    A_net = "A",
    T_leaf = "T_leaf",
    C_i = "Ci",
    PPFD = "Qin"
  ),
  useg_mct = TRUE,
  test1 = "gamma_star25",
  element_out = 1,
  test2 = "g_mc25",
  fitTPU = TRUE,
  Ea_gamma_star = 0,
  Ea_g_mc = 0,
  values1 = seq(
    from = 20,
    to = 40,
    by = 2
  ),
  values2 = seq(
    from = 0.5,
    to = 2,
    by = 0.1
  )
)

# Graph V_cmax
ggplot(pars, aes(x = gamma_star25, y = g_mc25, z = V_cmax)) +
  geom_tile(aes(fill = V_cmax)) +
  labs(
    x = expression(Gamma * "*"[25] ~ "(" * mu * mol ~ mol^-
    -1
    ) * ")"),
    y = expression(g[m][25] ~ "(" * mu * mol ~ m^-
    -2
    ) ~ s*
    -1
    ) ~ Pa*
    )
# aq_response

Non-rectangular hyperbolic model of light responses

Description

[Deprecated]

Please use marshall_biscoe_1980().

Usage

```r
aq_response(k_sat, phi_J, Q_abs, theta_J)
```

Arguments

- **k_sat**: Light saturated rate of process k
- **phi_J**: Quantum efficiency of process k
- **Q_abs**: Absorbed light intensity (umol m\(^{-2}\) s\(^{-1}\))
- **theta_J**: Curvature of the light response

Value

`aq_response` is used to describe the response of a process to absorbed light intensity. Assumes that input is absorbed light. Note that if absorbed light is not used, then the meaning of phi_J becomes unclear. This function is designed to be used with fit_aq_response, however it could easily be fed into a different fitting approach (e.g. Bayesian approaches). Originally from Marshall et al. 1980.

References

A_supply

CO2 supply and demand function (mol / m^2 s)

Description
This function is not intended to be called by users directly.

Usage
A_supply(C_chl, pars, unitless = FALSE, use_legacy_version = FALSE)
A_demand(C_chl, pars, unitless = FALSE)

Arguments
C_chl Chloroplastic CO2 concentration in Pa of class units
pars Concatenated parameters (leaf_par, enviro_par, and constants)
unitless Logical. Should units be set? The function is faster when FALSE, but input
must be in correct units or else results will be incorrect without any warning.
use_legacy_version Logical. Should legacy model (<2.1.0) be used? See NEWS for further infor-
mation. Default is FALSE.

Details
Supply function:

\[ A = g_{tc}(C_{air} - C_{chl}) \]

Demand function:

\[ A = (1 - \Gamma^* / C_{chl})\min(W_{carbox}, W_{regen}, W_{tpu}) - R_d \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>photosynthetic rate</td>
<td>( \mu \text{mol CO2} / (\text{m}^2 \text{s}) )</td>
<td>calculated</td>
</tr>
<tr>
<td>g_{tc}</td>
<td>g_ttc</td>
<td>total conductance to CO2</td>
<td>( \mu \text{mol CO2} / (\text{m}^2 \text{s Pa}) )</td>
<td>calculated</td>
</tr>
<tr>
<td>C_{air}</td>
<td>C_air</td>
<td>atmospheric CO2 concentration</td>
<td>Pa</td>
<td>41</td>
</tr>
<tr>
<td>C_{chl}</td>
<td>C_chl</td>
<td>chloroplastic CO2 concentration</td>
<td>Pa</td>
<td>calculated</td>
</tr>
<tr>
<td>R_d</td>
<td>R_d</td>
<td>nonphotorespiratory CO2 release</td>
<td>( \mu \text{mol CO2} / (\text{m}^2 \text{s}) )</td>
<td>2</td>
</tr>
<tr>
<td>\Gamma^*</td>
<td>gamma_star</td>
<td>chloroplastic CO2 compensation point</td>
<td>Pa</td>
<td>3.743</td>
</tr>
</tbody>
</table>

Value
Value in mol / (m^2 s) of class units
Examples

```r
bake_par = make_bakepar()
constants = make_constants(use_tealeaves = FALSE)
enviro_par = make_enviropar(use_tealeaves = FALSE)
leaf_par = bake(leaf_par, enviro_par, bake_par, constants)
# Or bake with piping (need library(magrittr))
# leaf_par %<>% bake(enviro_par, bake_par, constants)
enviro_par$T_air = leaf_par$T_leaf

pars = c(leaf_par, enviro_par, constants)
C_chl = set_units(350, umol/mol)

A_supply(C_chl, pars)
A_demand(C_chl, pars)
```

---

bake  Leaf parameter temperature responses

Description

'bake' leaf parameters using temperature response functions

Usage

```r
bake(leaf_par, enviro_par, bake_par, constants, assert_units = TRUE)

temp_resp1(par25, E_a, R, T_leaf, T_ref, unitless)

temp_resp2(par25, D_s, E_a, E_d, R, T_leaf, T_ref, unitless)
```

Arguments

- `leaf_par`: A list of leaf parameters inheriting class `leaf_par`. This can be generated using the `make_leafpar` function.
- `enviro_par`: A list of environmental parameters inheriting class `enviro_par`. This can be generated using the `make_enviropar` function.
- `bake_par`: A list of temperature response parameters inheriting class `bake_par`. This can be generated using the `make_bakepar` function.
- `constants`: A list of physical constants inheriting class `constants`. This can be generated using the `make_constants` function.
- `assert_units`: Logical. Should parameter units be checked? The function is faster when FALSE, but input must be in correct units or else results will be incorrect without any warning.
- `par25`: Parameter value at 25 °C of class units.
Empirical temperature response value in J/mol of class units.

Ideal gas constant in J/(mol K) of class units. See make_constants().

Leaf temperature in K of class units. Will be converted to °C.

Reference temperature in K of class units.

Logical. Should units be set? The function is faster when FALSE, but input must be in correct units or else results will be incorrect without any warning.

Empirical temperature response value in J/(mol K) of class units.

Empirical temperature response value in J/mol of class units.

Details

Several leaf parameters (leaf_par()) are temperature sensitive. Temperature-sensitive parameters are input at a reference temperature of 25 °C. These parameters are provided as par_name25 and then "baked" using the appropriate temperature response function and parameters in bake_par(). The "baked" parameter will have the name without "25" appended (par_name). E.g. V_cmax25 becomes V_cmax.

Temperature response functions following Buckley and Diaz-Espejo (2015)

Temperature response function 1 (temp_response1):

\[
\text{par}(T_{\text{leaf}}) = \text{par}_25 \exp\left(\frac{E_a}{R T_{\text{ref}}}(T_{\text{leaf}} - 25)/(T_{\text{leaf}} + 273.15)\right)
\]

T_ref is the reference temperature in K
T_leaf is the leaf temperature in °C

Temperature response function 2 (temp_response2) is the above equation multiplied by:

\[
(1 + \exp((D_s/R - E_d/(R T_{\text{ref}})))/(1 + \exp((D_s/R) - (E_d/(R (T_{\text{leaf}} + 273.15))))))
\]

Function 1 increases exponentially with temperature; Function 2 peaks a particular temperature.

Value

Constructor function for baked class. This will also inherit class leaf_par() and list(). This function ensures that temperature is "baked in" to leaf parameter calculations T_leaf using temperature response functions detailed below.

References

Examples

```r
bake_par = make_bakepar()
constants = make_constants(use_tealeaves = FALSE)
enviro_par = make_enviropar(use_tealeaves = FALSE)
leaf_par = make_leafpar(
  replace = list(T_leaf = set_units(293.15, K)),
  use_tealeaves = FALSE
)
baked_leafpar = bake(leaf_par, enviro_par, bake_par, constants)
```

```
baked_leafpar$V_cmax25
baked_leafpar$V_cmax
```

**baked-class**

*S3 class baked*

**Description**

See `bake()`

**bake_par**

*S3 class bake_par*

**Description**

S3 class bake_par

**Usage**

`bake_par(.x)`

**Arguments**

.x A list to be constructed into `bake_par`.

**Value**

Constructor function for `bake_par` class. This function ensures that leaf temperature gets properly "baked" into leaf parameters.
**calculated-parameters**  
*Get default functions for calculated parameters in photosynthesis*

---

**Description**

Get default functions for calculated parameters in photosynthesis

**Usage**

```r
get_f_parameter(.f_name)
```

**Arguments**

- `.f_name` character string of function

---

**calculate_jmax**  
*Inverse non-rectangular hyperbola for J_max calculation*

---

**Description**

Inverse non-rectangular hyperbola for J_max calculation

**Usage**

```r
calculate_jmax(PPFD, alpha, J, theta_J)
calculate_j(PPFD, alpha, J_max, theta_J)
```

**Arguments**

- `PPFD` light intensity in umol m-2 s-1
- `alpha` initial slope of the light response
- `J` electron transport rate in umol m-2 s-1
- `theta_J` curvature of the light response
- `J_max` maximum rate of electron transport in umol m-2 s-1

**Value**

- `calculate_jmax` calculates J_max given PPFD and J. It is necessary for the electron transport component of the fit_aci_response function.
- `calculate_j` provides a model of the light response of J. It is necessary for fitting the electron transport component of the photosynthetic CO2 response curves in fit_aci_response.
Description

Conductance to CO2 (mol / m^2 / s)

- g_tc: total conductance to CO2
- g_uc: cuticular conductance to CO2
- g_bc: boundary layer conductance to CO2
- g_mc: mesophyll conductance to CO2
- g_sc: stomatal conductance to CO2

Usage

- .get_gtc(pars, unitless, use_legacy_version)
- .get_guc(pars, surface, unitless)
- .get_gbc(pars, surface, unitless, use_legacy_version)
- .get_gmc(pars, surface, unitless)
- .get_gsc(pars, surface, unitless)

Arguments

- pars: Concatenated parameters (leaf_par, enviro_par, and constants)
- unitless: Logical. Should units be set? The function is faster when FALSE, but input must be in correct units or else results will be incorrect without any warning.
- use_legacy_version: Logical. Should legacy model (<2.1.0) be used? See NEWS for further information. Default is FALSE.
- surface: Leaf surface (lower or upper)

Details

**Default conductance model**

The conductance model described in this section is used by default unless additional anatomical parameters described in the next section are provided.

Total conductance to CO2 is the sum of parallel conductances on the lower \((g_c,\text{lower})\) and upper \((g_c,\text{upper})\) leaf portions:
Each partial conductance consists of two parallel conductances, the cuticular conductance \( g_{uc} \) and the in-series conductances through mesophyll \( g_{mc} \), stomata \( g_{sc} \), and boundary layer \( g_{bc} \). To simplify the formula, I use substitute resistance where \( r_x = 1/g_x \). For surface \( i \):

\[
g_{c,i} = g_{u,i} + (1/(r_{m,i} + r_{s,i} + r_{b,i}))
\]

The cuticular, stomatal, and mesophyll conductances can be the same or different for upper and lower. The partitioning factors \( k_x \) divide the conductance between surfaces while keeping the total conductance constant:

\[
g_{x,lower} = g_x(1/(1 + k_x))
\]

\[
g_{x,upper} = g_x(k_x/(1 + k_x))
\]

\[
g_x = g_{x,lower} + g_{x,upper}
\]

How the partitioning factors work:

- **0**: all conductance on **lower** surface/portion
- **0.5**: \( 2/3 \) conductance on **lower** surface
- **1**: conductance evenly divided between surfaces/portions
- **2**: \( 2/3 \) conductance on **upper** surface
- **Inf**: all conductance on **upper** surface/portion

The boundary layer conductances for each are calculated on the basis of mass and heat transfer (see `get_gbc()`).

### Symbols and Units

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_{mc} )</td>
<td>mesophyll conductance to CO2 (T_leaf)</td>
<td>mol / m² / s</td>
<td>calculated</td>
</tr>
<tr>
<td>( g_{sc} )</td>
<td>stomatal conductance to CO2</td>
<td>mol / m² / s</td>
<td>0.4</td>
</tr>
<tr>
<td>( g_{uc} )</td>
<td>cuticular conductance to CO2</td>
<td>mol / m² / s</td>
<td>0.01</td>
</tr>
<tr>
<td>( k_{mc} )</td>
<td>partition of ( g_{mc} ) to lower mesophyll</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>( k_{sc} )</td>
<td>partition of ( g_{sc} ) to lower surface</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>( k_{uc} )</td>
<td>partition of ( g_{uc} ) to lower surface</td>
<td>none</td>
<td>1</td>
</tr>
</tbody>
</table>

### New conductance model

The conductance model described in this section is implemented in `photosynthesis` (>= 2.1.0) if parameters to calculate the internal airspace and liquid-phase conductances \( (A_{mes_A}, g_{liqc}) \) are provided. These parameters are 1) the effective path lengths through the lower and upper leaf internal airspaces \( (\delta_{ias_lower}, \delta_{ias_upper}) \) and 2) the mesophyll area per leaf area \( (A_{mes_A}) \) and liquid-phase conductance per mesophyll cell area \( (g_{liqc}) \).

Two parallel diffusion pathways, one from each leaf surface, converge to a single CO2 concentration...
at the mesophyll cell boundary. We use a single liquid-phase resistance to represent the combined cell wall, plasmalemma, and chloroplast resistances. The gas-phase resistance through boundary layer, cuticle/stomata, and internal airspace is \( r_{gas,c,i} \); the liquid-phase intracellular resistance is \( r_{i,c} \).

\[
\begin{align*}
    r_{total,c} &= r_{gas,c} + r_{i,c}.
\end{align*}
\]

The gas-phase resistance occurs through two parallel pathways, which we refer to as the 'lower' and 'upper' pathways because horizontally oriented leaves often have different anatomical properties on each surface. The gas-phase resistance through pathway \( i \in \{ \text{lower, upper} \} \) is:

\[
    r_{gas,c,i} = r_{b,c,i} + r_{u+s,c,i} + r_{ias,c,i}
\]

The subscripts \( b, u+s, \) and \( ias \) denote boundary layer, cuticular + stomatal, and internal airspace, respectively. The subscript \( c \) indicates we are considering the conductance to CO2 rather than another molecular species.

Cuticular and stomatal conductances (1 / resistance) are parallel, so:

\[
    \frac{1}{r_{u+s,c,i}} = g_{u+s,c,i} = g_{u,c,i} + g_{s,c,i}
\]

Substituting the above expression into the equation for \( r_{gas,c,i} \):

\[
    r_{gas,c,i} = r_{b,c,i} + \frac{1}{(g_{u,c,i} = g_{s,c,i})} + r_{ias,c,i}
\]

The total gas-phase resistance is the inverse of the sum of the parallel lower and upper conductances:

\[
    \frac{1}{r_{gas,c}} = g_{gas,c,lower} + g_{gas,c,upper}
\]

The cuticular, stomatal, and mesophyll conductances can be the same or different for upper and lower. The partitioning factors \( k_u \) and \( k_s \) divide the total cuticular and stomatal conductances, respectively, between surfaces while keeping the total conductance constant:

\[
    g_x,lower = g_x(1/(1 + k_x))
\]

\[
    g_x,upper = g_x(k_x/(1 + k_x))
\]

\[
    g_x = g_x,lower + g_x,upper
\]

How the partitioning factors work:

\[
\begin{array}{ll}
    k_x & \text{description} \\
    0 & \text{all conductance on lower surface/portion} \\
    0.5 & 2/3 \text{ conductance on lower surface} \\
    1 & \text{conductance evenly divided between surfaces/portions} \\
    2 & 2/3 \text{ conductance on upper surface} \\
    \text{Inf} & \text{all conductance on upper surface/portion}
\end{array}
\]

The internal airspace conductance is the diffusivity of CO2 at a given temperature and pressure
divided by the effective path length:

\[ g_{\text{ias}, \text{lower}} = \frac{D_c}{\delta_{\text{ias}, \text{lower}}} \]

\[ g_{\text{ias}, \text{upper}} = \frac{D_c}{\delta_{\text{ias}, \text{upper}}} \]

\( g_{\text{iasc}, \text{lower}} \) and \( g_{\text{iasc}, \text{upper}} \) are calculated in the \texttt{bake} function. See \texttt{tealeaves::get_Dx()} for calculating \( D_c \).

The liquid-phase intracellular resistance is given by:

\[ \frac{1}{r_{i,c}} = g_{i,c} = g_{\text{liq},c} A_{\text{mes}} / A \]

\( g_{\text{liq},c} \) is temperature sensitive. See \texttt{bake()}.

The boundary layer conductances for each are calculated on the basis of mass and heat transfer (see \texttt{.get_gbc()}).

---

**compile_data**

### Compiling outputs from lists

**Description**

Compiling outputs from lists

**Usage**

```r
compile_data(data, output_type = "list", list_element)
```

**Arguments**

- `data` List of elements
- `output_type` Type of desired output. For graphs or models, use "list", for parameters, use "dataframe".
- `list_element` Which elements of the sublists do you wish to compile?

**Value**

`compile_data` converts the outputs of \texttt{fit_many} into a form more readily usable for analysis. Can be used to create dataframe of all fitted parameters, a list of model outputs, a list of graphs for plotting. This function is NOT restricted to compiling outputs from \texttt{plantecophysys} tools but could be used to compile elements from ANY list of lists.
Examples

# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_Ci_Q_data_1.csv",
    package = "photosynthesis"
))

# Define a grouping factor based on light intensity to split the ACi
# curves
data$Q_2 <- as.factor(\(\text{round}(\text{data}\$Qin, \text{digits} = 0)\))

# Convert leaf temperature to K
data$T_leaf <- data$Tleaf + 273.15

# Fit many curves
fits <- fit_many(
    data = data,
    varnames = list(
        A_net = "A",
        T_leaf = "T_leaf",
        C_i = "Ci",
        PPFD = "Qin"
    ),
    funct = fit_aci_response,
    group = "Q_2"
)

# Compile graphs into a list for plotting
fits_graphs <- compile_data(fits,
    list_element = 2
)

# Plot one graph from the compiled list
plot(fits_graphs[[1]])

---

**compute_sensitivity**

Computing measures of sensitivity

Description

Computing measures of sensitivity

Usage

compute_sensitivity(
compute_sensitivity

```r
data,
  varnames = list(Par = "Par", test1 = "test1", test2 = "test2"),
  test1_ref,
  test2_ref
)
```

**Arguments**

- `data` Dataframe with output from sensitivity_analysis()
- `varnames` Variable names
- `test1_ref` Reference value for parameter
- `test2_ref` Reference value for parameter

**Value**

compute_sensitivity calculates two sets of sensitivity measures: parameter effect (Bauerle et al., 2014), and control coefficient (Capaldo & Pandis, 1997). This function is useful in determining how much a given input (assumed or otherwise) can affect the model output and conclusions. Particularly useful if a given parameter is unknown during a fitting or modeling process.

**References**


**Examples**

```r
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis")
```

# Define a grouping factor based on light intensity to split the ACi curves
data$Q_2 <- as.factor(round(data$Qin, digits = 0))

# Convert leaf temperature to K
data$T_leaf <- data$Tleaf + 273.15

# Run a sensitivity analysis on gamma_star and mesophyll conductance
# at 25 Celsius for one individual curve
# pars <- analyze_sensitivity(
# data = data[data$Q_2 == 1500, ],
```
# funct = fit_aci_response,
# varnames = list(
#   A_net = "A",
#   T_leaf = "T_leaf",
#   C_i = "Ci",
#   PPFD = "Qin"
# ),
# useg_mct = TRUE,
# test1 = "gamma_star25",
# element_out = 1,
# test2 = "g_mc25",
# fitTPU = TRUE,
# Ea_gamma_star = 0,
# Ea_g_mc = 0,
# values1 = seq(
#   from = 20,
#   to = 60,
#   by = 2
# ),
# values2 = seq(
#   from = 0.2,
#   to = 2,
#   by = 0.1
# )
# )
# Compute measures of sensitivity
# par2 <- compute_sensitivity(
#   data = pars,
#   varnames = list(
#     Par = "V_cmax",
#     test1 = "gamma_star25",
#     test2 = "g_mc25"
#   ),
#   test1_ref = 42,
#   test2_ref = 1
# )
# # Plot control coefficients
# ggplot(par2, aes(y = CE_gamma_star25, x = CE_g_mc25, colour = V_cmax)) +
# geom_point() +
# theme_bw()
# # Note that in this case a missing point appears due to an infinity

---

## constants

### S3 class constants

**Description**

S3 class constants
enviro_par

Usage

constants(.x, use_tealeaves)

Arguments

.x A list to be constructed into constants.
use_tealeaves Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)?
If TRUE, tleaf() calculates T_leaf. If FALSE, user-defined T_leaf is used.
Additional parameters and constants are required, see make_parameters().

Value

Constructor function for constants class. This function ensures that physical constant inputs are properly formatted.

enviro_par S3 class enviro_par

Description

S3 class enviro_par

Usage

enviro_par(.x, use_tealeaves)

Arguments

.x A list to be constructed into enviro_par.
use_tealeaves Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)?
If TRUE, tleaf() calculates T_leaf. If FALSE, user-defined T_leaf is used.
Additional parameters and constants are required, see make_parameters().

Value

Constructor function for enviro_par class. This function ensures that environmental parameter inputs are properly formatted.
fit_aci_response  Fitting ACi curves

Description
Fitting ACi curves

Usage
fit_aci_response(
  data,
  varnames = list(A_net = "A_net", T_leaf = "T_leaf", C_i = "C_i", PPFD = "PPFD", g_mc = "g_mc"),
  P = 100,
  fitTPU = TRUE,
  alpha_g = 0,
  R_d_meas = NULL,
  useR_d = FALSE,
  useg_mc = FALSE,
  useg_mct = FALSE,
  usegamma_star = FALSE,
  useK_M = FALSE,
  useK_C_K_O = FALSE,
  alpha = 0.24,
  theta_J = 0.85,
  gamma_star25 = 42.75,
  Ea_gamma_star = 37830,
  K_M25 = 718.4,
  Ea_K_M = 65508.28,
  g_mc25 = 0.08701,
  Ea_g_mc = 0,
  K_C25 = NULL,
  Ea_K_C = NULL,
  K_O25 = NULL,
  Ea_K_O = NULL,
  Oconc = 21,
  gamma_star_set = NULL,
  K_M_set = NULL,
  ...
)

Arguments
  data  Dataframe for A-Ci curve fitting
  varnames  List of variable names. varnames = list(A_net = "A_net", T_leaf = "T_leaf", C_i = "C_i", PPFD = "PPFD", g_mc = "g_mc"), where A_net is net CO2 assimilation, T_leaf is leaf temperature in Celsius, C_i is intercellular CO2 concentration,
tion in umol/mol. PPFD is incident irradiance in umol m-2 s-1 (note that it is
ASSUMED to be absorbed irradiance, so be sure to adjust according to light ab-
sorbance and PSI/PSII partitioning accordingly OR interpret the resultant values of J and J_max with caution), g_mc is mesophyll conductance to CO2 in mol m-2 s-1 Pa-1.

P  Atmospheric pressure in kPa

fitTPU  Should triose phosphate utilization (V_TPU) be fit?

alpha_g  Fraction of respiratory glycolate carbon that is not returned to the chloroplast (von Caemmerer, 2000). If ACi curves show high-CO2 decline, then this value should be > 0.

R_d_meas  Measured value of respiratory CO2 efflux in umol m-2 s-1. Input value should be positive to work as expected with the equations.

tuseR_d  Use a measured value of R_d? Set to TRUE if using R_d_meas.

useg_mc  Use mesophyll conductance? Set to TRUE if specifying g_mc in varnames above.

useg_mct  Use mesophyll conductance temperature response? Set to TRUE if using a temperature response of mesophyll conductance.

usegamma_star  Specify gamma_star value? If FALSE, uses a temperature response function with Nicotiana tabacum defaults from Bernacchi et al. 2001.


useK_C_K_O  Use individual carboxylation/oxygenation constants for rubisco? If TRUE, need to specify values at 25C and activation energy for the Arrhenius temperature response function.

alpha  Quantum yield of CO2 assimilation

theta_J  Curvature of the photosynthetic light response curve

gamma_star25  gamma_star at 25C in umol mol-1

Ea_gamma_star  Activation energy of gamma_star in J mol-1

K_M25  Michaelis-Menten constant for rubisco at 25C

Ea_K_M  Activation energy for K_M in J mol-1

g_mc25  Mesophyll conductance at 25C in mol m-2 s-1

Ea_g_mc  Activation energy of g_mc in J mol-1

K_C25  Michaelis-Menten constant for rubisco carboxylation at 25C

Ea_K_C  Activation energy for K_C in J mol-1

K_O25  Michaelis-Menten constant for rubisco oxygenation at 25C

Ea_K_O  Activation energy for K_O in J mol-2

Oconc  O2 concentration in %. Used with P to calculate intracellular O2 when using K_C_K_O

gamma_star_set  Value of gamma_star to use (in ppm) if usegamma_star = TRUE

K_M_set  Value of K_M to use if useK_M = TRUE

...  Other arguments to pass on
Value

fit_aci_response fits ACi curves using an approach similar to Gu et al. 2010. Iterates all possible C_i transition points and checks for inadmissible curve fits. If no curves are admissible (either due to poor data or poor assumed parameters), the output will include a dataframe of NA values. Default parameters are all from Bernacchi et al. 2001, 2002.

References


Examples

```r
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", 
  package = "photosynthesis")
)

# Define a grouping factor based on light intensity to split the ACi curves
data$Q_2 <- as.factor(round(data$Qin, digits = 0))

# Convert leaf temperature to K
data$T_leaf <- data$Tleaf + 273.15

# Fit ACi curve. Note that we are subsetting the dataframe here to fit for a single value of Q_2
fit <- fit_aci_response(data[data$Q_2 == 1500, ],
  varnames = list(
    A_net = "A",
    T_leaf = "T_leaf",
    C_i = "Ci",
    PPFD = "Qin"
  )
)

# View fitted parameters
fit[[1]]
```
# View graph
fit[[2]]

# View data with modelled parameters attached
fit[[3]]

# Fit many curves
fits <- fit_many(
  data = data,
  varnames = list(
    A_net = "A",
    T_leaf = "T_leaf",
    C_i = "Ci",
    PPFD = "Qin"
  ),
  funct = fit_aci_response,
  group = "Q_2"
)

# Print the parameters
# First set of double parentheses selects an individual group value
# Second set selects an element of the sublist
fits[[3]][[1]]

# Print the graph
fits[[3]][[2]]

# Compile graphs into a list for plotting
fits_graphs <- compile_data(fits,
  list_element = 2
)

# Compile parameters into dataframe for analysis
fits_pars <- compile_data(fits,
  output_type = "dataframe",
  list_element = 1
)

---

fit_aq_response  Fitting light responses of net CO2 assimilation

**Description**

[Deprecated]

Please use `fit_aq_response2()`.
Usage

`fit_aq_response(`
  data,
  varnames = list(A_net = "A_net", PPFD = "PPFD"),
  usealpha_Q = FALSE,
  alpha_Q = 0.84,
  title = NULL
`
)

Arguments

data Dataframe containing CO2 assimilation light response
varnames Variable names where varnames = list(A_net = "A_net", PPFD = "PPFD"). A_net is net CO2 assimilation in umol m-2 s-1, PPFD is incident irradiance. PPFD can be corrected for light absorbance by using useapha_Q and setting alpha_Q.
usealpha_Q Correct light intensity for absorbance? Default is FALSE.
alpha_Q Absorbance of incident light. Default value is 0.84.
title Title for graph

Value

`fit_aq_response` fits the light response of net CO2 assimilation. Output is a dataframe containing light saturated net CO2 assimilation, quantum yield of CO2 assimilation (phi_J), curvature of the light response (theta_J), respiration (Rd), light compensation point (LCP), and residual sum of squares (resid_SS). Note that Rd fitted in this way is essentially the same as the Kok method, and represents a respiration value in the light that may not be accurate. Rd output should thus be interpreted more as a residual parameter to ensure an accurate fit of the light response parameters. Model originally from Marshall & Biscoe 1980.

References


Examples

```r
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data = read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis")
```

```r
# Fit many AQ curves
# Set your grouping variable
# Here we are grouping by CO2_s and individual
```
data$C_s = (round(data$CO2_s, digits = 0))

# For this example we need to round sequentially due to CO2_s setpoints
data$C_s = as.factor(round(data$C_s, digits = -1))

# To fit one AQ curve
fit = fit_aq_response(data[data$C_s == 600, ],
  varnames = list(
    A_net = "A",
    PPFD = "Qin"
  )
)

# Print model summary
summary(fit[[1]])

# Print fitted parameters
fit[[2]]

# Print graph
fit[[3]]

# Fit many curves
fits = fit_many(
  data = data,
  varnames = list(
    A_net = "A",
    PPFD = "Qin",
    group = "C_s"
  ),
  funct = fit_aq_response,
  group = "C_s"
)

# Look at model summary for a given fit
# First set of double parentheses selects an individual group value
# Second set selects an element of the sublist
summary(fits[[3]][[1]])

# Print the parameters
fits[[3]][[2]]

# Print the graph
fits[[3]][[3]]

# Compile graphs into a list for plotting
fits_graphs = compile_data(fits,
  list_element = 3
)

# Compile parameters into dataframe for analysis
fits_pars = compile_data(fits,
  output_type = "dataframe"
**fit_aq_response2**

Fit photosynthetic light-response curves

**Description**

We recommend using `fit_photosynthesis()` with argument `.photo_fun = "aq_response"` rather than calling this function directly.

**Usage**

```r
fit_aq_response2(
  .data,
  .model = "default",
  .method = "ls",
  usealpha_Q = FALSE,
  alpha_Q = 0.84,
  quiet = FALSE,
  brm_options = NULL
)
```

**Arguments**

- **.data** A data frame containing plant ecophysiological data. See `required_variables()` for the variables required for each model.
- **.model** A character string of model name to use. See `get_all_models()`.
- **.method** A character string of the statistical method to use: 'ls' for least-squares and 'brms' for Bayesian model using `brms::brm()`. Default is 'ls'.
- **usealpha_Q** Flag. Should light intensity be multiplied by `alpha_Q` before fitting? Default is FALSE (i.e. assume that '.Q' is absorbed light).
- **alpha_Q** Number. Absorbance of incident light. Default value is 0.84. Ignored if `usealpha_Q` = FALSE.
- **quiet** Flag. Should messages be suppressed? Default is FALSE.
- **brm_options** A list of options passed to `brms::brm()` if .method = "brms". Default is NULL.

**Value**

- If .method = 'ls': an `stats::nls()` object.
- If .method = 'brms': a `brms::brmsfit()` object.
**Note**

Rd fitted in this way is essentially the same as the Kok (1956) method, and represents a respiration value in the light that may not be accurate. Rd output should thus be interpreted more as a residual parameter to ensure an accurate fit of the light response parameters. Model originally from Marshall & Biscoe (1980).

**References**


**Examples**

```r
library(broom)
library(dplyr)
library(photosynthesis)

# Read in your data
dat = system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis") |> read.csv()
# Set grouping variable
mutate(group = round(CO2_s, digits = 0)) |> # For this example, round sequentially due to CO2_s set points
    mutate(group = as.factor(round(group, digits = -1)))

# Fit one light-response curve
fit = fit_photosynthesis(
    .data = filter(dat, group == 600),
    .photo_fun = "aq_response",
    .vars = list(.A = A, .Q = Qabs),
)

# The 'fit' object inherits class 'nls' and many methods can be used
## Model summary:
summary(fit)
## Estimated parameters:
coef(fit)
## 95% confidence intervals:
confint(fit)
## Tidy summary table using 'broom::tidy()'
tidy(fit, conf.int = TRUE, conf.level = 0.95)
# Fit multiple curves with **photosynthesis** and **purrr**
library(purrr)
```

fits = dat |> 
  split(~ group) |> 
  map(fit_photosynthesis, .photo_fun = "aq_response", .vars = list(A = A, Q = Qabs))

---

**fit_gs_model**

**Fitting stomatal conductance models**

**Description**

Fitting stomatal conductance models

**Usage**

```r
fit_gs_model(
  data,
  varnames = list(A_net = "A_net", C_air = "C_air", g_sw = "g_sw", RH = "RH", VPD = "VPD"),
  model = c("BallBerry", "Leuning", "Medlyn_partial", "Medlyn_full"),
  D0 = 3,
  ...)
```

**Arguments**

- `data`  
  Dataframe

- `varnames`  
  Variable names
  For the Ball-Berry model: `varnames = list(A_net = "A_net", C_air = "C_air", g_sw = "g_sw", RH = "RH")` where `A_net` is net CO2 assimilation, `C_air` is CO2 concentration at the leaf surface in umol mol-1, `g_sw` is stomatal conductance to H2O, and `RH` is relative humidity as a proportion.
  For the Leuning model: `varnames = list(A_net = "A_net", C_air = "C_air", g_sw = "g_sw", VPD = "VPD")` where `A_net` is net CO2 assimilation, `C_air` is CO2 concentration at the leaf surface in umol mol-1, `g_sw` is stomatal conductance to H2O, and `VPD` is leaf to air vapor pressure deficit in kPa.
  For the Medlyn et al. 2011 models: `varnames = list(A_net = "A_net", C_air = "C_air", g_sw = "g_sw", VPD = "VPD")` where `A_net` is net CO2 assimilation, `C_air` is CO2 concentration at the leaf surface in umol mol-1, `g_sw` is stomatal conductance to H2O, and `VPD` is leaf to air vapor pressure deficit in kPa.

- `model`  
  Which model(s) to fit? Defaults to all models. Available options are "Ball-Berry", "Leuning", "Medlyn_partial", and "Medlyn_full", from Ball et al. (1987), Leuning (1995), and Medlyn et al. (2011).

- `D0`  
  Vapor pressure sensitivity of stomata (Leuning 1995)

...  
Arguments to pass on to the nlsLM() function for the Medlyn models.
Value

fit_gs_model fits one or more stomatal conductance models to the data. The top level of the output list is named after the fitted model, while the second level contains the Model, Parameters, and Graph, in that order.

References


Examples

# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_Ci_Q_data_1.csv",
                          package = "photosynthesis"
))

# Convert RH to a proportion
data$RH <- data$RHcham / 100

# Fit stomatal conductance models
# Can specify a single model, or all as below
fits <- fit_gs_model(
  data = data,
  varnames = list(
    A_net = "A",
    C_air = "Ca",
    g_sw = "gsw",
    RH = "RH",
    VPD = "VPDleaf"
  ),
  model = c(
    "BallBerry",
    "Leuning",
    "Medlyn_partial",
    "Medlyn_full"
  ),
  D0 = 3
)

# Look at BallBerry model summary:
summary(fits[["BallBerry"]][["Model"]])

# Look at BallBerry parameters
fits[["BallBerry"]][["Parameters"]]

# Look at BallBerry plot
fits[["BallBerry"]][["Graph"]]

# Fit many g_sw models
# Set your grouping variable
# Here we are grouping by Qin and individual
data$Q_2 <- as.factor(round(data$Qin, digits = 0))

fits <- fit_many(data,
                 varnames = list(
                                  A_net = "A",
                                  C_air = "Ca",
                                  g_sw = "gsw",
                                  RH = "RH",
                                  VPD = "VPDleaf"
                                ),
                 funct = fit_gs_model,
                 group = "Q_2"
)

# Look at the Medlyn_partial outputs at 750 PAR
# Model summary
summary(fits[["750"]][["Medlyn_partial"]][["Model"]])

# Model parameters
fits[["750"]][["Medlyn_partial"]][["Parameters"]]

# Graph
fits[["750"]][["Medlyn_partial"]][["Graph"]]

# Compile parameter outputs for BallBerry model
# Note that it's the first element for each PAR value
# First compile list of BallBerry fits
bbmods <- compile_data(
  data = fits,
  output_type = "list",
  list_element = 1
)

# Now compile the parameters (2nd element) into a dataframe
bbpars <- compile_data(
  data = bbmods,
  output_type = "dataframe",
  list_element = 2
)

# Convert group variable back to numeric
fit_g_mc_variableJ

bbpars$ID <- as.numeric(bbpars$ID)

# Take quick look at light response of intercept parameters
plot(g0 ~ ID, bbpars)

# Compile graphs
graphs <- compile_data(
  data = bbmods,
  output_type = "list",
  list_element = 3
)

# Look at 3rd graph
graphs[[3]]

---

fit_g_mc_variableJ  Fitting mesophyll conductance with the variable J method

Description
Fitting mesophyll conductance with the variable J method

Usage
fit_g_mc_variableJ(
  data,
  varnames = list(A_net = "A_net", J_etr = "J_etr", C_i = "C_i", PPFD = "PPFD", phi_PSII = "phi_PSII"),
  usealpha_Q = FALSE,
  alpha_Q = 0.84,
  beta_Q = 0.5,
  gamma_star,
  R_d,
  P = 100
)

Arguments
data  Dataframe
varnames  Variable names to fit g_mc. varnames = list(A_net = "A_net", J_etr = "J_etr", C_i = "C_i", PPFD = "PPFD", phi_PSII = "phi_PSII"), where A_net is net CO2 assimilation in umol m-2 s-1, J_etr is linear electron transport flux in umol m-2 s-1, C_i is intercellular CO2 concentration in umol mol-1, PPFD is incident irradiance in umol m-2 s-1, phi_PSII is the operating efficiency of photosystem II.
usealpha_Q  Recalculate electron transport with new absorbance value?
**Value**

*fit_g_mc_variableJ* fits mesophyll conductance according to Harley et al. 1992. It also tests the reliability of the calculation and calculates a mean with only reliable values. Note that the output is in units of umol m⁻² s⁻¹ Pa⁻¹.

**References**


**Examples**

```r
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO₂ by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", 
    package = "photosynthesis")
)

# Note: there will be issues here if the alpha value used
# for calculating ETR is off, if gamma_star is incorrect,
# if R_d is incorrect.
data <- fit_g_mc_variableJ(data, 
  varnames = list( 
    A_net = "A",
    J_etr = "ETR",
    C_i = "Ci",
    PPFD = "Qin",
    phi_PSII = "PhiPS2"
  ),
  gamma_star = 46,
  R_d = 0.153,
  usealpha_Q = TRUE,
  alpha_Q = 0.84,
  beta_Q = 0.5,
  P = 84
)

# Note that many g_mc values from this method can be unreliable
ggplot(data, aes(x = CO2_s, y = g_mc, colour = reliable)) +
  labs(
    ...
fit_hydra_vuln_curve

Fitting hydraulic vulnerability curves

Description

Fitting hydraulic vulnerability curves

Usage

```r
fit_hydra_vuln_curve(
  data,
  varnames = list(psi = "psi", PLC = "PLC"),
  start_weibull = list(a = 2, b = 2),
  title = NULL
)```
fit_hydra_vuln_curve

Arguments

data Dataframe

varnames List of variable names. varnames = list(psi = "psi", PLC = "PLC") where psi is water potential in MPa, and PLC is percent loss conductivity.

start_weibull starting values for the nls fitting routine for the Weibull curve

title Title for the output graph

Value

fit_hydra_vuln_curve fits a sigmoidal function (Pammenter & Van der Willigen, 1998) linearized according to Ogle et al. (2009). Output is a list containing the sigmoidal model in element 1 and Weibull model in element 4, the fit parameters with 95% confidence interval for both models are in element 2, and hydraulic parameters in element 3 (including P25, P50, P88, P95, S50, Pe, Pmax, DSI). Px (25 to 95): water potential at which x% of conductivity is lost. S50: slope at 50% loss of conductivity. Pe: air entry point. Pmax: hydraulic failure threshold. DSI: drought stress interval. Element 5 is a graph showing the fit, P50, Pe, and Pmax.

References


Pammenter NW, Van der Willigen CV . 1998. A mathematical and statistical analysis of the curves illustrating vulnerability of xylem to cavitation. Tree Physiology 18:589-593

Examples

# Read in data
data <- read.csv(system.file("extdata", "hydraulic_vulnerability.csv", package = "photosynthesis")

# Fit hydraulic vulnerability curve
fit <- fit_hydra_vuln_curve(data[data$Tree == 4 & data$Plot == "Control", ],
   varnames = list(
      psi = "P",
      PLC = "PLC"
   ),
   title = "Control 4"
 )

# Return Sigmoidal model summary
summary(fit[[1]])

# Return Weibull model summary
summary(fit[[4]])

# Return model parameters with 95% confidence intervals
fit[[2]]
# Return hydraulic parameters
fit[[3]]

# Return graph
fit[[5]]

# Fit many curves
fits <- fit_many(
  data = data,
  varnames = list(
    psi = "P",
    PLC = "PLC"
  ),
  group = "Tree",
  funct = fit_hydra_vuln_curve
)

# To select individuals from the many fits
# Return model summary
summary(fits[[1]][[1]]) # Returns model summary

# Return sigmoidal model output
fits[[1]][[2]]

# Return hydraulic parameters
fits[[1]][[3]]

# Return graph
fits[[1]][[5]]

# Compile parameter outputs
pars <- compile_data(
  data = fits,
  output_type = "dataframe",
  list_element = 3
)

# Compile graphs
graphs <- compile_data(
  data = fits,
  output_type = "list",
  list_element = 5
)
Description

[Deprecated]

We are no longer updating this function. Please use generic methods like \texttt{map} instead. See \texttt{vignette("light-response")} for an example.

Usage

\begin{verbatim}
fit_many(data, funct, group, progress = TRUE, ...)
\end{verbatim}

Arguments

- \texttt{data} \hspace{1cm} Dataframe
- \texttt{funct} \hspace{1cm} Function to fit
- \texttt{group} \hspace{1cm} Grouping variables
- \texttt{progress} \hspace{1cm} Flag. Show progress bar?
- \texttt{...} \hspace{1cm} Arguments for the function to fit. Use \texttt{?functionname} to read the help file on available arguments for a given function.

Value

\texttt{fit_many} fits a function across every instance of a grouping variable.

Examples

\begin{verbatim}
# Read in your data
# Note that this data is coming from data supplied by the package hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data = read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis")

# Define a grouping factor based on light intensity to split the ACi curves
data$Q_2 = as.factor(round(data$Qin, digits = 0))

# Convert leaf temperature to K
data$T_leaf = data$Tleaf + 273.15

# Fit many curves
fits = fit_many(
    data = data,
    varnames = list(
        A_net = "A",
        T_leaf = "T_leaf",
        C_i = "Ci",
        PPFD = "Qin"
    ),

)\end{verbatim}
fit_photosynthesis

Fit photosynthetic models with gas-exchange data

Description

Fit photosynthetic models with gas-exchange data

Usage

```
fit_photosynthesis(
  .data, .photo_fun, .model = "default", .vars = NULL, .method = "ls",
  ..., quiet = FALSE, brm_options = NULL
)
```

Arguments

- **.data**: A data frame containing plant ecophysiological data. See `required_variables()` for the variables required for each model.

fit_PV_curve

Description

Fitting pressure-volume curves

Usage

```r
fit_PV_curve(
  data,
  varnames = list(psi = "psi", mass = "mass", leaf_mass = "leaf_mass", bag_mass = "bag_mass", leaf_area = "leaf_area"),
  title = NULL
)
```
fit_PV_curve

Arguments

data Dataframe

varnames Variable names. varnames = list(psi = "psi", mass = "mass", leaf_mass = "leaf_mass",
bag_mass = "bag_mass", leaf_area = "leaf_area") where psi is leaf water potential in MPa, mass is the weighed mass of the bag and leaf in g, leaf_mass is the mass of the leaf in g, bag_mass is the mass of the bag in g, and leaf_area is the area of the leaf in cm2.

title Graph title

Value

fit_PV_curve fits pressure-volume curve data to determine: SWC: saturated water content per leaf mass (g H2O g leaf dry mass ^ -1), PI_o: osmotic potential at full turgor (MPa), psi_TLP: leaf water potential at turgor loss point (TLP) (MPa), RWC_TLP: relative water content at TLP (%), eps: modulus of elasticity at full turgor (MPa), C_FT: relative capacitance at full turgor (MPa ^ -1), C_TLP: relative capacitance at TLP (MPa ^ -1), and C_FTStar: absolute capacitance per leaf area (g m ^ -2 MPa ^ -1). Element 1 of the output list contains the fitted parameters, element 2 contains the water-psi graph, and element 3 contains the 1/psi-100-RWC graph.

References


Examples

# Read in data
data <- read.csv(system.file("extdata", "PV_curve.csv",
package = "photosynthesis"
))

# Fit one PV curve
fit <- fit_PV_curve(data[data$ID == "L2", ],
varnames = list(
    psi = "psi",
    mass = "mass",
    leaf_mass = "leaf_mass",
    bag_mass = "bag_mass",
    leaf_area = "leaf_area"
)
)

# See fitted parameters
fit[[1]]

# Plot water mass graph
fit[[2]]

# Plot PV Curve
fit[[3]]

# Fit all PV curves in a file
fits <- fit_many(data,
                 group = "ID",
                 funct = fit_PV_curve,
                 varnames = list(
                   psi = "psi",
                   mass = "mass",
                   leaf_mass = "leaf_mass",
                   bag_mass = "bag_mass",
                   leaf_area = "leaf_area"
                 )
)

# See parameters
fits[[1]][[1]]

# See water mass - water potential graph
fits[[1]][[2]]

# See PV curve
fits[[1]][[3]]

# Compile parameter outputs
pars <- compile_data(
  data = fits,
  output_type = "dataframe",
  list_element = 1
)

# Compile the water mass - water potential graphs
graphs1 <- compile_data(
  data = fits,
  output_type = "list",
  list_element = 2
)

# Compile the PV graphs
graphs2 <- compile_data(
  data = fits,
  output_type = "list",
  list_element = 3
)
Fit models to estimate light respiration (R_d)

Description
We recommend using `fit_photosynthesis()` with argument `photo_fun = "r_light"` rather than calling this function directly.

Usage
```r
fit_r_light2(
  .data,
  .model = "default",
  .method = "ls",
  Q_lower = NA,
  Q_upper = NA,
  Q_levels = NULL,
  C_upper = NA,
  quiet = FALSE,
  brm_options = NULL
)
```

Arguments
- `.data` A data frame containing plant ecophysiological data. See `required_variables()` for the variables required for each model.
- `.model` A character string of model name to use. See `get_all_models()`.
- `.method` A character string of the statistical method to use: 'ls' for least-squares and 'brms' for Bayesian model using `brms::brm()`. Default is 'ls'.
- `Q_levels` A numeric vector of light intensity levels (µmol / mol) for estimating R_d from the linear region of the A-C curve using the walker_ort_2015 model.
- `C_upper` Upper C (µmol / mol) limit for estimating R_d from the linear region of the A-C curve using the walker_ort_2015 model.
- `quiet` Flag. Should messages be suppressed? Default is FALSE.
- `brm_options` A list of options passed to `brms::brm()` if `.method = "brms"`. Default is NULL.

Value
- If `.method = 'ls'`: an `stats::nls()` or `stats::lm()` object.
- If `.method = 'brms'`: a `brms::brmsfit()` object.
Note

Confusingly, $R_d$ typically denotes respiration in the light, but you might see $R_{day}$ or $R_{light}$.

Models

*Kok (1956)*

The kok_1956 model estimates light respiration using the Kok method (Kok, 1956). The Kok method involves looking for a breakpoint in the light response of net CO2 assimilation at very low light intensities and extrapolating from data above the breakpoint to estimate light respiration as the y-intercept. $R_d$ value should be negative, denoting an efflux of CO2.

*Yin et al. (2011)*

The yin_etal_2011 model estimates light respiration according to the Yin et al. (2009, 2011) modifications of the Kok method. The modification uses fluorescence data to get a better estimate of light respiration. $R_d$ values should be negative here to denote an efflux of CO2.

*Walker & Ort (2015)*

The walker_ort_2015 model estimates light respiration and $\Gamma^*$ according to Walker & Ort (2015) using a slope-intercept regression method to find the intercept of multiple A-C curves run at multiple light intensities. The method estimates $\Gamma^*$ and $R_d$. If estimated $R_d$ is positive this could indicate issues (i.e. leaks) in the gas exchange measurements. $\Gamma^*$ is in units of umol / mol and $R_d$ is in units of $\mu$mol m$^{-2}$ s$^{-1}$ of respiratory flux. If using $C_i$, the estimated value is technically $C_i^*$. You need to use $C_c$ to get $\Gamma^*$ Also note, however, that the convention in the field is to completely ignore this note.

References


Walker BJ, Ort DR. 2015. Improved method for measuring the apparent CO2 photocompensation point resolves the impact of multiple internal conductances to CO2 to net gas exchange. Plant Cell Environ 38:2462- 2474


Examples

# Walker & Ort (2015) model

library(broom)
library(dplyr)
library(photosynthesis)
acq_data = system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis") |>
   read.csv()

fit = fit_photosynthesis(
   .data = acq_data,
   .photo_fun = "r_light",
   .model = "walker_ort_2015",
   .vars = list(.A = A, .Q = Qin, .C = Ci),
   C_upper = 300,
   # Irradiance levels used in experiment
   Q_levels = c(1500, 750, 375, 125, 100, 75, 50, 25),
)

# The 'fit' object inherits class 'lm' and many methods can be used

## Model summary:
summary(fit)

## Estimated parameters:
coef(fit)

## 95% confidence intervals:
## n.b. these confidence intervals are not correct because the regression is fit
## sequentially. It ignores the underlying data and uncertainty in estimates of
## slopes and intercepts with each A-C curve. Use '.method = "brms"' to properly
## calculate uncertainty.
confint(fit)

## Tidy summary table using 'broom::tidy()'
tidy(fit, conf.int = TRUE, conf.level = 0.95)

## Calculate residual sum-of-squares
sum(resid(fit)^2)

# Yin et al. (2011) model

fit = fit_photosynthesis(
   .data = acq_data,
   .photo_fun = "r_light",
   .model = "yin_etal_2011",
   .vars = list(.A = A, .phiPSII = PhiPS2, .Q = Qin),
   Q_lower = 20,
   Q_upper = 250
)

# The 'fit' object inherits class 'lm' and many methods can be used

## Model summary:
summary(fit)

## Estimated parameters:
coef(fit)
## 95% confidence intervals:
confint(fit)

## Tidy summary table using 'broom::tidy()'
tidy(fit, conf.int = TRUE, conf.level = 0.95)

## Calculate residual sum-of-squares
sum(resid(fit)^2)

# Kok (1956) model

fit = fit_photosynthesis(
  .data = acq_data,
  .photo_fun = "r_light",
  .model = "kok_1956",
  .vars = list(.A = A, .Q = Qin),
  Q_lower = 20,
  Q_upper = 150
)

# The 'fit' object inherits class 'lm' and many methods can be used

## Model summary:
summary(fit)

## Estimated parameters:
coef(fit)

## 95% confidence intervals:
confint(fit)

## Tidy summary table using 'broom::tidy()'
tidy(fit, conf.int = TRUE, conf.level = 0.95)

## Calculate residual sum-of-squares
sum(resid(fit)^2)

---

**fit_r_light_kok**

*Estimating light respiration*

### Description

[Deprecated]

Please use `fit_r_light2()`.

### Usage

```r
fit_r_light_kok()
```
fit_r_light_kok

data,
  varnames = list(A_net = "A_net", PPFD = "PPFD"),
  PPFD_lower = 40,
  PPFD_upper = 100
)

fit_r_light_WalkerOrt(
  data,
  varnames = list(A_net = "A_net", C_i = "C_i", PPFD = "PPFD"),
  P = 100,
  C_i_threshold = 300
)

fit_r_light_yin(
  data,
  varnames = list(A_net = "A_net", PPFD = "PPFD", phi_PSII = "phi_PSII"),
  PPFD_lower = 40,
  PPFD_upper = 100
)

Arguments

data        Dataframe
varnames    List of variable names
PPFD_lower  Lower light intensity limit for estimating Rlight (Kok & Yin)
PPFD_upper  Upper light intensity limit for estimating Rlight (Kok & Yin)
P          Atmospheric pressure in kPa (Walker & Ort, 2015)
C_i_threshold Threshold C_i (in umol / mol) to cut data to linear region for fitting light respiration and gamma_star (Walker & Ort, 2015)

Value

fit_r_light_kok estimates light respiration using the Kok method (Kok, 1956). The Kok method involves looking for a breakpoint in the light response of net CO2 assimilation at very low light intensities and extrapolating from data above the breakpoint to estimate light respiration as the y-intercept. r_light value should be negative, denoting an efflux of CO2.

fit_r_light_WalkerOrt estimates light respiration and GammaStar according to Walk & Ort (2015) using a slope-intercept regression method to find the intercept of multiple ACi curves run at multiple light intensities. Output GammaStar and respiration should be negative. If output respiration is positive this could indicate issues (i.e. leaks) in the gas exchange measurements. GammaStar is output in umol mol-1, and respiration is output in umol m-2 s-1 of respiratory flux. Output is a list containing the slope intercept regression model, a graph of the fit, and estimates of the coefficients. NOTE: if using C_i, the output value is technically C_istar. You need to use Cc to get GammaStar. Also note, however, that the convention in the field is to completely ignore this note.

fit_r_light_yin estimates light respiration according to the Yin et al. (2009, 2011) modifications of the Kok method. The modification uses fluorescence data to get a better estimate of light respiration. Note that respiration output should be negative here to denote an efflux of CO2.
References


Examples

# FITTING KOK METHOD
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data = read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis")

# Fit light respiration with Kok method
r_light = fit_r_light_kok(  
data = data,  
varnames = list(  
  A_net = "A",  
  PPFD = "Qin"  
),  
  PPFD_lower = 20,  
  PPFD_upper = 150  
)

# Return r_light
r_light

# FITTING WALKER-ORT METHOD
# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data = read.csv(system.file("extdata", "A_Ci_Q_data_1.csv", package = "photosynthesis")

# Fit the Walker-Ort method for GammaStar and light respiration
walker_or = fit_r_light_WalkerOrt(data,
Fitting temperature responses

Description

Fitting temperature responses

Usage

```r
call = fit_t_response(data,
varnames = list(Par = "Par", T_leaf = "T_leaf"),
```
model = c("Arrhenius", "Kruse", "Heskel", "Medlyn", "MMRT", "Quadratic", "Topt"),
start = list(a = 1, b = 1, c = 1, dEa = 1, Ea_ref = 1, Par_ref = 1, Ea = 40000, Par25 =
      50, Hd = 2e+05, dS = 650, dCp = 1, dG = 1, dH = 1),
setvar = "none",
hdset = 2e+05,
dSset = 650,
title = NULL,
...)

Arguments

data Dataframe with temperature response variables
varnames Variable names, where Par is the parameter of interest, and T_leaf is the leaf temperature in K.
model Which temperature response model do you want to use? Defaults to all: Arrhenius, Heskel, Kruse, Medlyn, MMRT, Quadratic, and Topt.
start List of starting parameters for the nls model fits. a, b, and c are needed for the Heskel model, dEa, Ea_ref, and Par_ref are needed for the Kruse model, Ea, Par25, and Hd are all needed for the Medlyn and Topt models while the Medlyn model also requires dS, and dCp, dG, and dH are all for the MMRT model.
setvar Which variable to set as constant for the Medlyn model? Defaults to "none", while "Hd" and "dS" options are available.
hdset Which value should Hd be set to when setvar = "Hd"? Specify in J/mol.
dSset Which value should dS be set to when setvar = "dS"? Specify in J/mol/K.
title Title of output graphs
... Further arguments to pass on to the nlsLM() function

Value

fit_t_response fits one or more temperature response models to a dataset, returning a list of lists. The parent list contains the models, while the child list for each model contains the fitted model in element 1, the coefficients in element 2, and a graph in element 3.

References

Heskel MA, O’Sullivan OS, Reich PB, Tjoelker MG, Weerasinghe LK, Penillard A, Egerton JIG,
Creek D, Bloomfield KJ, Xiang J, Sinca F, Stangl ZR, la Torre AM, Griffin KL, Huntingford C,


Examples

# Read in data
data <- read.csv(system.file("extdata", "A_Ci_T_data.csv", 
  package = "photosynthesis" 
),
stringsAsFactors = FALSE)

library(tidyr)

# Round temperatures to group them appropriately
# Use sequential rounding
data$T2 <- round(data$Tleaf, 1)
data$T2 <- round(data$Tleaf, 0)

# Look at unique values to detect rounding issues
unique(data$T2)

# Some still did not round correctly,
# manually correct
for (i in 1:nrow(data)) {
  if (data$T2[i] == 18) {
    data$T2[i] <- 17
  }
  if (data$T2[i] == 23) {
    data$T2[i] <- 22
  }
  if (data$T2[i] == 28) {
    data$T2[i] <- 27
  }
  if (data$T2[i] == 33) {
    data$T2[i] <- 32
  }
  if (data$T2[i] == 38) {
    data$T2[i] <- 37
  }
}

# Make sure it is a character string for grouping
data$T2 <- as.character(data$T2)
# Create grouping variable by ID and measurement temperature
```
data <- unite(data,
  col = "ID2", c("ID", "T2"),
  sep = "_"
)
```

# Split by temperature group
```
data <- split(data, data$ID2)
```

# Obtain mean temperature for group so temperature
# response fitting is acceptable later, round to
# 2 decimal places
```
for (i in 1:length(data)) {
  data[[i]]$Curve_Tleaf <- round(mean(data[[i]]$Tleaf), 2)
}
```

# Convert from list back to dataframe
```
data <- do.call("rbind", data)
```

# Parse grouping variable by ID and measurement temperature
```
data <- separate(data,
  col = "ID2", into = c("ID", "T2"),
  sep = "_"
)
```

# Make sure number of values matches number of measurement
# temperatures. May vary slightly if plants had slightly
# different leaf temperatures during the measurements
```
unique(data$Curve_Tleaf)
```

# Create ID column to curve fit by ID and temperature
```
data <- unite(data,
  col = "ID2", c("ID", "Curve_Tleaf"),
  sep = "_"
)
```

# Convert leaf temperature to K
```
data$T_leaf <- data$Tleaf + 273.15
```

# Fit many CO2 response curves
```
fits2 <- fit_many(
  data = data,
  group = "ID2",
  varnames = list(
    A_net = "A",
    C_i = "Ci",
    T_leaf = "T_leaf",
    PPFD = "Qin",
    g_mc = "g_mc"
  ),
  funct = fit_aci_response,
  alphag = 0
)
```
# Extract ACi parameters
pars <- compile_data(fits2,
  output_type = "dataframe",
  list_element = 1
)

# Extract ACi graphs
graphs <- compile_data(fits2,
  output_type = "list",
  list_element = 2
)

# Parse the ID variable
pars <- separate(pars, col = "ID", into = c("ID", "Curve_Tleaf"), sep = "_")

# Make sure curve leaf temperature is numeric
pars$Curve_Tleaf <- as.numeric(pars$Curve_Tleaf)
pars$T_leaf <- pars$Curve_Tleaf + 273.15

# Fit all models, set Hd to constant in Medlyn model
out <- fit_t_response(
  data = pars[pars$ID == "S2", ],
  varnames = list(
    Par = "V_cmax",
    T_leaf = "T_leaf"
  ),
  setvar = "Hd",
  hdset = 200000
)

out[["Arrhenius"]][["Graph"]]
out[["Heskel"]][["Graph"]]
out[["Kruse"]][["Graph"]]
out[["Medlyn"]][["Graph"]]
out[["MMRT"]][["Graph"]]
out[["Quadratic"]][["Graph"]]
out[["Topt"]][["Graph"]]

---

**FvCB**

**Farquhar-von Caemmerer-Berry (FvCB) C3 photosynthesis model**

**Description**

Farquhar-von Caemmerer-Berry (FvCB) C3 photosynthesis model
Rubisco-limited assimilation rate
RuBP regeneration-limited assimilation rate
TPU-limited assimilation rate
Usage

FvCB(C_chl, pars, unitless = FALSE)

W_carbox(C_chl, pars, unitless = FALSE)

W_regen(C_chl, pars, unitless = FALSE)

W_tpu(C_chl, pars, unitless = FALSE)

Arguments

- **C_chl**: Chloroplastic CO2 concentration in Pa of class units
- **pars**: Concatenated parameters (leaf_par, enviro_par, and constants)
- **unitless**: Logical. Should units be set? The function is faster when FALSE, but input
must be in correct units or else results will be incorrect without any warning.

Details

Equations following Buckley and Diaz-Espejo (2015):

**Rubisco-limited assimilation rate:**

\[
W_{\text{carbox}} = V_{c,\text{max}}C_{\text{chl}}/(C_{\text{chl}} + K_m)
\]

where:

\[
K_m = K_C(1 + O/K_O)
\]

**RuBP regeneration-limited assimilation rate:**

\[
W_{\text{regen}} = J C_{\text{chl}}/(4C_{\text{chl}} + 8\Gamma*)
\]

where \(J\) is a function of PPFD, obtained by solving the equation:

\[
0 = \theta J J^2 - J(J_{\text{max}} + \phi_J PPFD) + J_{\text{max}}\phi_J PPFD
\]

**TPU-limited assimilation rate:**

\[
W_{\text{tpu}} = 3V_{\text{tpu}}C_{\text{chl}}/(C_{\text{chl}} - \Gamma*)
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_chl</td>
<td>C_chl</td>
<td>chloroplastic CO2 concentration</td>
<td>Pa</td>
<td>input</td>
</tr>
</tbody>
</table>
\[ \text{FvCB} \]

### Chloroplastic CO2 Compensation Point (T_leaf)

- \( \Gamma^* \): chloroplastic CO2 compensation point (T_leaf) in Pa, calculated.
- \( J_{\text{max}} \): potential electron transport (T_leaf) in \( \mu\text{mol CO2} / (\text{m}^2 \text{s}) \), calculated.
- \( K_C \): Michaelis constant for carboxylation (T_leaf) in \( \mu\text{mol} / \text{mol} \), calculated.
- \( K_O \): Michaelis constant for oxygenation (T_leaf) in \( \mu\text{mol} / \text{mol} \), calculated.
- \( O \): atmospheric O2 concentration in kPa, 21.27565.
- \( \phi_J \): initial slope of the response of J to PPFD in none, 0.331.
- \( \text{PPFD} \): photosynthetic photon flux density in umol quanta / (m^2 s), 1500.
- \( R_d \): nonphotorespiratory CO2 release (T_leaf) in \( \mu\text{mol CO2} / (\text{m}^2 \text{s}) \), calculated.
- \( \theta_J \): curvature factor for light-response curve in none, 0.825.
- \( V_{c,\text{max}} \): maximum rate of carboxylation (T_leaf) in \( \mu\text{mol CO2} / (\text{m}^2 \text{s}) \), calculated.
- \( V_{tpu} \): rate of triose phosphate utilization (T_leaf) in \( \mu\text{mol CO2} / (\text{m}^2 \text{s}) \), calculated.

### Value

A list of four values with units \( \text{umol CO2} / (\text{m}^2 \text{s}) \) of class units:

- \( W_{\text{carbox}} \): Rubisco-limited assimilation rate
- \( W_{\text{regen}} \): RuBP regeneration-limited assimilation rate
- \( W_{\text{tpu}} \): TPU-limited assimilation rate
- \( A \): minimum of \( W_{\text{carbox}}, W_{\text{regen}}, \text{and } W_{\text{tpu}} \)

### References


### Examples

```r
bake_par = make_bakepar()
constants = make_constants(use_tealeaves = FALSE)
enviro_par = make_enviropar(use_tealeaves = FALSE)
leaf_par = make_leafpar(use_tealeaves = FALSE)
leaf_par = bake(leaf_par, enviro_par, bake_par, constants)

pars = c(leaf_par, enviro_par, constants)
C_chl = set_units(246.0161, umol / mol)
FvCB(C_chl, pars)
```
**get_default_model**  

**Description**  

[Experimental]  
Get the name of the default model used for different plant ecophysiological data analysis methods implemented in `photosynthesis`. Currently only used for `fit_aq_response2()` and `fit_r_light2()`.

**Light response models:**  
- `marshall_biscoe_1980()`: Non-rectangular hyperbolic model of light responses

**Usage**

```r
get_default_model(.photo_fun)
get_all_models(method)
marscshall_biscoe_1980(Q_abs, k_sat, phi_J, theta_J)
```

**Arguments**  

- `.photo_fun`  
  A character string of `photosynthesis` function to call. One of: `aq_response`, `r_light`.
- `method`  
  A character string of the statistical method to use: 'ls' for least-squares and 'brms' for Bayesian model using `brms::brm()`. Default is 'ls'.
- `Q_abs`  
  Absorbed light intensity (µmol m⁻² s⁻¹)
- `k_sat`  
  Light saturated rate of process k
- `phi_J`  
  Quantum efficiency of process k
- `theta_J`  
  Curvature of the light response

**Value**

A character string with name of model.

**Examples**

```r
get_default_model("aq_response")
get_default_model("r_light")
```
Stomatal conductance models

Usage

- `gs_mod_ballberry(A_net, C_air, RH)`
- `gs_mod_leuning(A_net, C_air, D0, VPD)`
- `gs_mod_opti(g0, g1, VPD, A_net, C_air)`
- `gs_mod_optifull(g0, g1, gk, VPD, A_net, C_air)`

Arguments

- `A_net`: Net CO2 assimilation in umol m-2 s-1
- `C_air`: CO2 concentration at the leaf surface in umol mol-1
- `RH`: Relative humidity as a proportion
- `D0`: Vapor pressure sensitivity of stomata (Leuning 1995)
- `VPD`: Vapor pressure deficit (kPa)
- `g0`: Optimization model intercept term (Medlyn et al. 2011)
- `g1`: Optimization model slope term (Medlyn et al. 2011)
- `gk`: Optimization model root term (Medlyn et al. 2011)

Value

- `gs_mod_ballberry` is used for fitting the Ball et al. (1987) model of stomatal conductance
- `gs_mod_leuning` is used for fitting the Leuning (1995) model of stomatal conductance
- `gs_mod_opti` fits the optimal stomatal conductance model according to Medlyn et al. 2011
- `gs_mod_optifull` fits the full optimal stomatal conductance model according to Medlyn et al. 2011

References


$J$

$J$: Rate of electron transport (umol/m^2/s)

**Description**

Calculate the rate of electron transport as a function of photosynthetic photon flux density (PPFD).

**Usage**

$J(pars, \text{unitless} = \text{FALSE})$

**Arguments**

- **pars**: Concatenated parameters (leaf_par, enviro_par, and constants)
- **unitless**: Logical. Should units be set? The function is faster when FALSE, but input must be in correct units or else results will be incorrect without any warning.

**Details**

$J$ as a function of PPFD is the solution to the quadratic expression:

$$0 = \theta_J J^2 - J(J_{\max} + \phi_J PPFD) + J_{\max} \phi_J PPFD$$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>$R$</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_{\max}$</td>
<td>$J_{\max}$</td>
<td>potential electron transport (T_leaf)</td>
<td>$\mu$mol CO2 / (m^2 s)</td>
<td>calculated</td>
</tr>
<tr>
<td>$\phi_J$</td>
<td>phi_J</td>
<td>initial slope of the response of $J$ to PPFD</td>
<td>none</td>
<td>0.331</td>
</tr>
<tr>
<td>PPFD</td>
<td>PPFD</td>
<td>photosynthetic photon flux density</td>
<td>$\mu$mol quanta / (m^2 s)</td>
<td>1500</td>
</tr>
<tr>
<td>$\theta_J$</td>
<td>theta_J</td>
<td>curvature factor for light-response curve</td>
<td>none</td>
<td>0.825</td>
</tr>
</tbody>
</table>

**Value**

Value in $\mu$mol/ (m^2 s) of class units

**Examples**

```r
library(magrittr)
library(photosynthesis)

bake_par = make_bakepar()
constants = make_constants(use_tealeaves = FALSE)
enviro_par = make_enviropar(use_tealeaves = FALSE)
leaf_par = make_leafpar(use_tealeaves = FALSE)
```
```r
enviro_par$T_air = leaf_par$T_leaf
leaf_par %<>% bake(enviro_par, bake_par, constants)

pars = c(leaf_par, enviro_par, constants)
J(pars, FALSE)
```

---

### leaf_par

**S3 class leaf_par**

**Description**

S3 class leaf_par

**Usage**

```r
leaf_par(.x, use_tealeaves)
```

**Arguments**

- `.x` A list to be constructed into `leaf_par`.
- `use_tealeaves` Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)? If TRUE, `tleaf()` calculates T_leaf. If FALSE, user-defined T_leaf is used. Additional parameters and constants are required, see `make_parameters()`.

**Value**

Constructor function for leaf_par class. This function ensures that leaf parameter inputs are properly formatted.

---

### make_parameters

**Make lists of parameters for photosynthesis**

**Description**

Make lists of parameters for photosynthesis

- `make_leafpar`
- `make_enviropar`
- `make_bakepar`
- `make_constants`
make_parameters

Usage

make_leafpar(replace = NULL, use_tealeaves)

make_enviropar(replace = NULL, use_tealeaves)

make_bakepar(replace = NULL)

make_constants(replace = NULL, use_tealeaves)

Arguments

replace A named list of parameters to replace defaults. If NULL, defaults will be used.

use_tealeaves Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)? If TRUE, tleaf() calculates T_leaf. If FALSE, user-defined T_leaf is used. Additional parameters and constants are required, see make_parameters().

Details

Constants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dc,0</td>
<td>D_c0</td>
<td>diffusion coefficient for CO2 in air at 0 °C</td>
<td>m²/s</td>
<td>1.29 × 10⁻⁵</td>
</tr>
<tr>
<td>Dh,0</td>
<td>D_h0</td>
<td>diffusion coefficient for heat in air at 0 °C</td>
<td>m²/s</td>
<td>1.90 × 10⁻⁵</td>
</tr>
<tr>
<td>Dm,0</td>
<td>D_m0</td>
<td>diffusion coefficient for momentum in air at 0 °C</td>
<td>m²/s</td>
<td>1.33 × 10⁻⁵</td>
</tr>
<tr>
<td>Dw,0</td>
<td>D_w0</td>
<td>diffusion coefficient for water vapor in air at 0 °C</td>
<td>m²/s</td>
<td>2.12 × 10⁻⁵</td>
</tr>
<tr>
<td>ϵ</td>
<td>epsilon</td>
<td>ratio of water to air molar masses</td>
<td>none</td>
<td>0.622</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>gravitational acceleration</td>
<td>m / s²</td>
<td>9.8</td>
</tr>
<tr>
<td>eT</td>
<td>eT</td>
<td>exponent for temperature dependence of diffusion</td>
<td>none</td>
<td>1.75</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>ideal gas constant</td>
<td>J / mol / K</td>
<td>8.31</td>
</tr>
<tr>
<td>σ</td>
<td>sigma</td>
<td>Stephan-Boltzmann constant</td>
<td>W / m² / K¹</td>
<td>5.67 × 10⁻⁸</td>
</tr>
<tr>
<td>f_sh</td>
<td>f_sh</td>
<td>function to calculate constant(s) for Sherwood number</td>
<td>none</td>
<td>NA</td>
</tr>
<tr>
<td>f_Nu</td>
<td>f_Nu</td>
<td>function to calculate constant(s) for Nusselt number</td>
<td>none</td>
<td>NA</td>
</tr>
</tbody>
</table>

Baking (i.e. temperature response) parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ds,gmc</td>
<td>Ds_gmc</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>487</td>
</tr>
<tr>
<td>Ds,Imax</td>
<td>Ds_Imax</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>388</td>
</tr>
<tr>
<td>Ea,gamma*</td>
<td>Ea_gammastar</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>24500</td>
</tr>
<tr>
<td>Ea,gmc</td>
<td>Ea_gmc</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>68900</td>
</tr>
<tr>
<td>Ea,Imax</td>
<td>Ea_Imax</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>56100</td>
</tr>
<tr>
<td>Ea,KC</td>
<td>Ea_KC</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>81000</td>
</tr>
<tr>
<td>Ea,KO</td>
<td>Ea_KO</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>23700</td>
</tr>
<tr>
<td>Ea,Rd</td>
<td>Ea_Rd</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>40400</td>
</tr>
<tr>
<td>Ea,Vcmax</td>
<td>Ea_Vcmax</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>52200</td>
</tr>
<tr>
<td>Ea,Vtpu</td>
<td>Ea_Vtpu</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>52200</td>
</tr>
<tr>
<td>Ed,gmc</td>
<td>Ed_gmc</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
<td>149000</td>
</tr>
</tbody>
</table>
**Environment parameters:**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{air}}$</td>
<td>C_air</td>
<td>atmospheric CO2 concentration</td>
<td>umol/mol</td>
<td>420</td>
</tr>
<tr>
<td>$O$</td>
<td>O</td>
<td>atmospheric O2 concentration</td>
<td>mol/mol</td>
<td>0.21</td>
</tr>
<tr>
<td>$P$</td>
<td>P</td>
<td>atmospheric pressure</td>
<td>kPa</td>
<td>101</td>
</tr>
<tr>
<td>PPFD</td>
<td>PPFD</td>
<td>photosynthetic photon flux density</td>
<td>umol / m$^2$ / s</td>
<td>1500</td>
</tr>
<tr>
<td>RH</td>
<td>RH</td>
<td>relative humidity</td>
<td>none</td>
<td>0.5</td>
</tr>
<tr>
<td>$u$</td>
<td>wind</td>
<td>windspeed</td>
<td>m / s</td>
<td>2</td>
</tr>
</tbody>
</table>

**Leaf parameters:**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>leafsize</td>
<td>leaf characteristic dimension</td>
<td>m</td>
<td>0.1</td>
</tr>
<tr>
<td>$\Gamma*$</td>
<td>gamma_star</td>
<td>chloroplastic CO2 compensation point (T_leaf)</td>
<td>umol/mol</td>
<td>NA</td>
</tr>
<tr>
<td>$\Gamma_{25}$</td>
<td>gamma_star25</td>
<td>chloroplastic CO2 compensation point (25 °C)</td>
<td>umol/mol</td>
<td>37.9</td>
</tr>
<tr>
<td>$g_{\text{mc}}$</td>
<td>g_mc</td>
<td>mesophyll conductance to CO2 (T_leaf)</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{mc},25}$</td>
<td>g_mc25</td>
<td>mesophyll conductance to CO2 (25 °C)</td>
<td>mol / m$^2$ / s</td>
<td>0.4</td>
</tr>
<tr>
<td>$y_{\text{sc}}$</td>
<td>g_sc</td>
<td>stomatal conductance to CO2</td>
<td>mol / m$^2$ / s</td>
<td>0.4</td>
</tr>
<tr>
<td>$y_{\text{uc}}$</td>
<td>g_uc</td>
<td>cuticular conductance to CO2</td>
<td>mol / m$^2$ / s</td>
<td>0.01</td>
</tr>
<tr>
<td>$J_{\text{max},25}$</td>
<td>J_max25</td>
<td>potential electron transport (25 °C)</td>
<td>umol / m$^2$ / s</td>
<td>200</td>
</tr>
<tr>
<td>$J_{\text{max}}$</td>
<td>J_max</td>
<td>potential electron transport (T_leaf)</td>
<td>umol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$k_{\text{mc}}$</td>
<td>k_mc</td>
<td>partition of $g_{\text{mc}}$ to lower mesophyll</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>$k_{\text{sc}}$</td>
<td>k_sc</td>
<td>partition of $g_{\text{sc}}$ to lower surface</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>$k_{\text{uc}}$</td>
<td>k_uc</td>
<td>partition of $g_{\text{uc}}$ to lower surface</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>$K_{\text{C},25}$</td>
<td>K_C25</td>
<td>Michaelis constant for carboxylation (25 °C)</td>
<td>umol / mol</td>
<td>268</td>
</tr>
<tr>
<td>$K_{\text{C}}$</td>
<td>K_C</td>
<td>Michaelis constant for carboxylation (T_leaf)</td>
<td>umol / mol</td>
<td>NA</td>
</tr>
<tr>
<td>$K_{\text{O},25}$</td>
<td>K_O25</td>
<td>Michaelis constant for oxygenation (25 °C)</td>
<td>umol / mol</td>
<td>165000</td>
</tr>
<tr>
<td>$K_{\text{O}}$</td>
<td>K_O</td>
<td>Michaelis constant for oxygenation (T_leaf)</td>
<td>umol / mol</td>
<td>NA</td>
</tr>
<tr>
<td>$\phi_J$</td>
<td>phi_J</td>
<td>initial slope of the response of $J$ to PPFD</td>
<td>none</td>
<td>0.331</td>
</tr>
<tr>
<td>$R_{\text{d},25}$</td>
<td>R_d25</td>
<td>nonphotorespiratory CO2 release (25 °C)</td>
<td>umol / m$^2$ / s</td>
<td>2</td>
</tr>
<tr>
<td>$R_d$</td>
<td>R_d</td>
<td>nonphotorespiratory CO2 release (T_leaf)</td>
<td>umol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$\theta_J$</td>
<td>theta_J</td>
<td>curvature factor for light-response curve</td>
<td>none</td>
<td>0.825</td>
</tr>
<tr>
<td>$T_{\text{leaf}}$</td>
<td>T_leaf</td>
<td>leaf temperature</td>
<td>K</td>
<td>298</td>
</tr>
<tr>
<td>$V_{\text{c,max},25}$</td>
<td>V_cmax25</td>
<td>maximum rate of carboxylation (25 °C)</td>
<td>umol / m$^2$ / s</td>
<td>150</td>
</tr>
<tr>
<td>$V_{\text{c,max}}$</td>
<td>V_cmax</td>
<td>maximum rate of carboxylation (T_leaf)</td>
<td>umol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$V_{\text{tpu},25}$</td>
<td>V_tpu25</td>
<td>rate of triose phosphate utilization (25 °C)</td>
<td>umol / m$^2$ / s</td>
<td>200</td>
</tr>
<tr>
<td>$V_{\text{tpu}}$</td>
<td>V_tpu</td>
<td>rate of triose phosphate utilisation (T_leaf)</td>
<td>umol / m$^2$ / s</td>
<td>NA</td>
</tr>
</tbody>
</table>

If `use_tealeaves = TRUE`, additional parameters are:

**Constants:**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
</table>
### make_parameters

\[
\begin{align*}
& c_p \quad c_p \quad \text{heat capacity of air} \quad J / g / K \quad 1.01 \\
& R_{\text{air}} \quad R_{\text{air}} \quad \text{specific gas constant for dry air} \quad J / kg / K \quad 287
\end{align*}
\]

### Baking (i.e. temperature response) parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_q$</td>
<td>$E_q$ energy per mole quanta</td>
<td>$kJ / mol$</td>
<td>220</td>
</tr>
<tr>
<td>$f_{\text{PAR}}$</td>
<td>$f_{\text{par}}$ fraction of incoming shortwave radiation that is photosynthetically active radiation (PAR)</td>
<td>none</td>
<td>0.5</td>
</tr>
<tr>
<td>$r$</td>
<td>$r$ reflectance for shortwave irradiance (albedo)</td>
<td>none</td>
<td>0.2</td>
</tr>
<tr>
<td>$T_{\text{air}}$</td>
<td>$T_{\text{air}}$ air temperature</td>
<td>K</td>
<td>298</td>
</tr>
<tr>
<td>$T_{\text{sky}}$</td>
<td>$T_{\text{sky}}$ sky temperature</td>
<td>K</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Environment parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_q$</td>
<td>$E_q$ energy per mole quanta</td>
<td>$kJ / mol$</td>
<td>220</td>
</tr>
<tr>
<td>$f_{\text{PAR}}$</td>
<td>$f_{\text{par}}$ fraction of incoming shortwave radiation that is photosynthetically active radiation (PAR)</td>
<td>none</td>
<td>0.5</td>
</tr>
<tr>
<td>$r$</td>
<td>$r$ reflectance for shortwave irradiance (albedo)</td>
<td>none</td>
<td>0.2</td>
</tr>
<tr>
<td>$T_{\text{air}}$</td>
<td>$T_{\text{air}}$ air temperature</td>
<td>K</td>
<td>298</td>
</tr>
<tr>
<td>$T_{\text{sky}}$</td>
<td>$T_{\text{sky}}$ sky temperature</td>
<td>K</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Leaf parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_l$</td>
<td>$\alpha_l$ absorbity of longwave radiation (4 - 80 um)</td>
<td>none</td>
<td>0.97</td>
</tr>
<tr>
<td>$\alpha_s$</td>
<td>$\alpha_s$ absorbity of shortwave radiation (0.3 - 4 um)</td>
<td>none</td>
<td>0.5</td>
</tr>
<tr>
<td>$g_{\text{sw}}$</td>
<td>$g_{\text{sw}}$ stomatal conductance to H2O</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{uw}}$</td>
<td>$g_{\text{uw}}$ cuticular conductance to H2O</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>logit($sr$)</td>
<td>logit($sr$) stomatal ratio (logit transformed)</td>
<td>none</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Optional leaf parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R Description</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_{\text{ias,lower}}$</td>
<td>$\delta_{\text{ias,lower}}$ effective distance through lower internal airspace</td>
<td>um</td>
<td>NA</td>
</tr>
<tr>
<td>$\delta_{\text{ias,upper}}$</td>
<td>$\delta_{\text{ias,upper}}$ effective distance through upper internal airspace</td>
<td>um</td>
<td>NA</td>
</tr>
<tr>
<td>$A_{\text{mes}} / A$</td>
<td>$A_{\text{mes}} / A$ mesophyll surface area per unit leaf area</td>
<td>none</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{liqc,25}}$</td>
<td>$g_{\text{liqc,25}}$ liquid-phase conductance to CO2 (25 °C)</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{liqc}}$</td>
<td>$g_{\text{liqc}}$ liquid-phase conductance to CO2 (T_leaf)</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{iasc,lower}}$</td>
<td>$g_{\text{iasc,lower}}$ internal airspace conductance to CO2 in lower part of leaf (T_leaf)</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
<tr>
<td>$g_{\text{iasc,upper}}$</td>
<td>$g_{\text{iasc,upper}}$ internal airspace conductance to CO2 in upper part of leaf (T_leaf)</td>
<td>mol / m$^2$ / s</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Value

make_leafpar: An object inheriting from class leaf_par()
make_enviropar: An object inheriting from class enviro_par()
make_bakepar: An object inheriting from class bake_par()
make_constants: An object inheriting from class constants()
References


Examples

```r
bake_par = make_bakepar()
comasto = make_comastart(use_tealeaves = FALSE)
dat_par = make_datapar(use_tealeaves = FALSE)
leaf_par = make_leafpar(use_tealeaves = FALSE)

leaf_par = make_leafpar(
  replace = list(
    g_sc = set_units(0.3, mol / m^2 / s),
    V_cmax25 = set_units(100, umol / m^2 / s)
  ), use_tealeaves = FALSE
)
```

---

**parameter_names**

Get vector of parameter names

**Description**

Get vector of parameter names

**Usage**

```r
parameter_names(which, use_tealeaves)
```

**Arguments**

- `which`: A character string indicating which parameter names to retrieve: "leaf", "enviro", "bake", or "constants". Partial matching allowed.
- `use_tealeaves`: Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)? If TRUE, `tleaf()` calculates T_leaf. If FALSE, user-defined T_leaf is used. Additional parameters and constants are required, see `make_parameters()`.

**Value**

A character vector with parameter names associated with each type, "leaf", "enviro", "bake", or "constants".

**Examples**

```r
parameter_names("leaf", use_tealeaves = FALSE)
```
photosynthesis

Simulate C3 photosynthesis

Description

photosynthesis: simulate C3 photosynthesis over multiple parameter sets

photo: simulate C3 photosynthesis over a single parameter set

Usage

photosynthesis(
  leaf_par,
  enviro_par,
  bake_par,
  constants,
  use_tealeaves,
  progress = TRUE,
  quiet = FALSE,
  assert_units = TRUE,
  check = TRUE,
  parallel = FALSE,
  use_legacy_version = FALSE
)

photo(
  leaf_par,
  enviro_par,
  bake_par,
  constants,
  use_tealeaves,
  quiet = FALSE,
  assert_units = TRUE,
  check = TRUE,
  prepare_for_tleaf = use_tealeaves,
  use_legacy_version = FALSE
)

Arguments

leaf_par        A list of leaf parameters inheriting class leaf_par. This can be generated using the make_leafpar function.

enviro_par      A list of environmental parameters inheriting class enviro_par. This can be generated using the make_enviropar function.

bake_par        A list of temperature response parameters inheriting class bake_par. This can be generated using the make_bakepar function.
constants  A list of physical constants inheriting class constants. This can be generated using the make_constants function.

use_tealeaves  Logical. Should leaf energy balance be used to calculate leaf temperature (T_leaf)? If TRUE, tleaf() calculates T_leaf. If FALSE, user-defined T_leaf is used. Additional parameters and constants are required, see make_parameters().

progress  Logical. Should a progress bar be displayed?

quiet  Logical. Should messages be displayed?

assert_units  Logical. Should parameter units be checked? The function is faster when FALSE, but input must be in correct units or else results will be incorrect without any warning.

check  Logical. Should arguments checks be done? This is intended to be disabled when photo() is called from photosynthesis() Default is TRUE.

parallel  Logical. Should parallel processing be used via furrr::future_map()?

use_legacy_version  Logical. Should legacy model (<2.1.0) be used? See NEWS for further information. Default is FALSE.

prepare_for_tleaf  Logical. Should arguments additional calculations for tleaf()? This is intended to be disabled when photo() is called from photosynthesis(). Default is use_tealeaves.

Details

photo: This function takes simulates photosynthetic rate using the Farquhar-von Caemmerer-Berry (FvCB()) model of C3 photosynthesis for single combined set of leaf parameters (leaf_par()), environmental parameters (enviro_par()), and physical constants (constants()). Leaf parameters are provided at reference temperature (25 °C) and then “baked” to the appropriate leaf temperature using temperature response functions (see bake()).

photosynthesis: This function uses photo to simulate photosynthesis over multiple parameter sets that are generated using cross_df().

Value

A data.frame with the following units columns

Inputs:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dc,0</td>
<td>D_c0</td>
<td>diffusion coefficient for CO2 in air at 0 °C</td>
<td>m²/s</td>
</tr>
<tr>
<td>Dh,0</td>
<td>D_h0</td>
<td>diffusion coefficient for heat in air at 0 °C</td>
<td>m²/s</td>
</tr>
<tr>
<td>Dm,0</td>
<td>D_m0</td>
<td>diffusion coefficient for momentum in air at 0 °C</td>
<td>m²/s</td>
</tr>
<tr>
<td>Dw,0</td>
<td>D_w0</td>
<td>diffusion coefficient for water vapor in air at 0 °C</td>
<td>m²/s</td>
</tr>
<tr>
<td>ε</td>
<td>epsilon</td>
<td>ratio of water to air molar masses</td>
<td>none</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>gravitational acceleration</td>
<td>m / s²</td>
</tr>
<tr>
<td>eT</td>
<td>eT</td>
<td>exponent for temperature dependence of diffusion</td>
<td>none</td>
</tr>
</tbody>
</table>
photosynthesis

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>ideal gas constant</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Stephan-Boltzmann constant</td>
<td>W / m² / K</td>
</tr>
<tr>
<td>$\Gamma_{sh}$</td>
<td>function to calculate constant(s) for Sherwood number</td>
<td>none</td>
</tr>
<tr>
<td>$\Gamma_{nu}$</td>
<td>function to calculate constant(s) for Nusselt number</td>
<td>none</td>
</tr>
<tr>
<td>$D_{s,gmc}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$D_{s,\text{Jmax}}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,1,*}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,gmc}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,\text{Jmax}}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,KC}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,KO}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,Rd}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{a,\text{Vcmax}}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{d,gmc}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$E_{d,\text{Jmax}}$</td>
<td>empirical temperature response parameter</td>
<td>J / mol / K</td>
</tr>
<tr>
<td>$C_{\text{air}}$</td>
<td>atmospheric CO2 concentration</td>
<td>umol/mol</td>
</tr>
<tr>
<td>$O_{2}$</td>
<td>atmospheric O2 concentration</td>
<td>mol/mol</td>
</tr>
<tr>
<td>$P$</td>
<td>atmospheric pressure</td>
<td>kPa</td>
</tr>
<tr>
<td>$\text{PPFD}$</td>
<td>photosynthetic photon flux density</td>
<td>umol / m² / s</td>
</tr>
<tr>
<td>$\text{RH}$</td>
<td>relative humidity</td>
<td>none</td>
</tr>
<tr>
<td>$u$</td>
<td>wind speed</td>
<td>m / s</td>
</tr>
<tr>
<td>$d$</td>
<td>leaf characteristic dimension</td>
<td>m</td>
</tr>
<tr>
<td>$\Gamma_{25}$</td>
<td>chloroplastic CO2 compensation point (25 °C)</td>
<td>umol/mol</td>
</tr>
<tr>
<td>$g_{mc,25}$</td>
<td>mesophyll conductance to CO2 (25 °C)</td>
<td>mol / m² / s</td>
</tr>
<tr>
<td>$g_{sc}$</td>
<td>stomatal conductance to CO2</td>
<td>mol / m² / s</td>
</tr>
<tr>
<td>$g_{uc}$</td>
<td>cuticular conductance to CO2</td>
<td>mol / m² / s</td>
</tr>
<tr>
<td>$J_{\text{max}25}$</td>
<td>potential electron transport (25 °C)</td>
<td>umol / m² / s</td>
</tr>
<tr>
<td>$k_{mc}$</td>
<td>partition of $g_{mc}$ to lower mesophyll</td>
<td>none</td>
</tr>
<tr>
<td>$k_{sc}$</td>
<td>partition of $g_{sc}$ to lower surface</td>
<td>none</td>
</tr>
<tr>
<td>$k_{uc}$</td>
<td>partition of $g_{uc}$ to lower surface</td>
<td>none</td>
</tr>
<tr>
<td>$K_{C,25}$</td>
<td>Michaelis constant for carboxylation (25 °C)</td>
<td>umol / mol</td>
</tr>
<tr>
<td>$K_{O,25}$</td>
<td>Michaelis constant for oxygenation (25 °C)</td>
<td>umol / mol</td>
</tr>
<tr>
<td>$\phi_J$</td>
<td>initial slope of the response of $J$ to PPFD</td>
<td>none</td>
</tr>
<tr>
<td>$R_d25$</td>
<td>nonphotorespiratory CO2 release (25 °C)</td>
<td>umol / m² / s</td>
</tr>
<tr>
<td>$\theta_J$</td>
<td>curvature factor for light-response curve</td>
<td>none</td>
</tr>
<tr>
<td>$T_{\text{leaf}}$</td>
<td>leaf temperature</td>
<td>K</td>
</tr>
<tr>
<td>$V_{c,max,25}$</td>
<td>maximum rate of carboxylation (25 °C)</td>
<td>umol / m² / s</td>
</tr>
<tr>
<td>$V_{\text{tpu}25}$</td>
<td>rate of triose phosphate utilization (25 °C)</td>
<td>umol / m² / s</td>
</tr>
<tr>
<td>$E_{\text{ias,lower}}$</td>
<td>effective distance through lower internal airspace</td>
<td>um</td>
</tr>
<tr>
<td>$E_{\text{ias,upper}}$</td>
<td>effective distance through upper internal airspace</td>
<td>um</td>
</tr>
<tr>
<td>$A_{\text{mes}}$</td>
<td>mesophyll surface area per unit leaf area</td>
<td>m²</td>
</tr>
<tr>
<td>$g_{liqc25}$</td>
<td>liquid-phase conductance to CO2 (25 °C)</td>
<td>mol / m² / s</td>
</tr>
</tbody>
</table>

Baked Inputs:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma^*$</td>
<td>chloroplastic CO2 compensation point (T_leaf)</td>
</tr>
<tr>
<td>$g_{mc}$</td>
<td>mesophyll conductance to CO2 (T_leaf)</td>
</tr>
</tbody>
</table>
**photo_parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>potential electron transport</td>
<td>$J_{\text{max}}$</td>
<td>$J_{\text{max}}$</td>
<td>umol/m^2/s</td>
</tr>
<tr>
<td>Michaelis constant for carboxylation</td>
<td>$K_{\text{C}}$</td>
<td>$K_{\text{C}}$</td>
<td>umol/mol</td>
</tr>
<tr>
<td>Michaelis constant for oxygenation</td>
<td>$K_{\text{O}}$</td>
<td>$K_{\text{O}}$</td>
<td>umol/mol</td>
</tr>
<tr>
<td>nonphotorespiratory CO2 release</td>
<td>$R_{\text{d}}$</td>
<td>$R_{\text{d}}$</td>
<td>umol/m^2/s</td>
</tr>
<tr>
<td>maximum rate of carboxylation</td>
<td>$V_{\text{c, max}}$</td>
<td>$V_{\text{c, max}}$</td>
<td>umol/m^2/s</td>
</tr>
<tr>
<td>rate of triose phosphate utilisation</td>
<td>$V_{\text{tpu}}$</td>
<td>$V_{\text{tpu}}$</td>
<td>umol/m^2/s</td>
</tr>
<tr>
<td>liquid-phase conductance to CO2</td>
<td>$g_{\text{liq}}$</td>
<td>$g_{\text{liq}}$</td>
<td>mol/m^2/s</td>
</tr>
<tr>
<td>internal airspace conductance to CO2 in lower part of leaf</td>
<td>$g_{\text{ias, lower}}$</td>
<td>$g_{\text{ias, lower}}$</td>
<td>mol/m^2/s</td>
</tr>
<tr>
<td>internal airspace conductance to CO2 in upper part of leaf</td>
<td>$g_{\text{ias, upper}}$</td>
<td>$g_{\text{ias, upper}}$</td>
<td>mol/m^2/s</td>
</tr>
</tbody>
</table>

**Output:**

- Photosynthetic rate at $C_{\text{chl}}$ ($\mu$mol CO2/m^2/s)
- Chloroplastic CO2 concentration where $A_{\text{supply}}$ intersects $A_{\text{demand}}$ (mmol/mol)
- Intercellular CO2 concentration where $A_{\text{supply}}$ intersects $A_{\text{demand}}$ (mmol/mol)
- Total conductance to CO2 at $T_{\text{leaf}}$ (mol/m^2/s)
- Value of $A_{\text{supply}}$ - $A_{\text{demand}}$ ($\mu$mol/(m^2)s) at $C_{\text{chl}}$
- Convergence code (0 = converged)

**Examples**

```r
# Single parameter set with 'photo'

bake_par = make_bakepar()
constants = make_constants(use_tealeaves = FALSE)
enviro_par = make_enviropar(use_tealeaves = FALSE)
leaf_par = make_leafpar(use_tealeaves = FALSE)
photo(leaf_par, enviro_par, bake_par, constants, use_tealeaves = FALSE)

# Multiple parameter sets with 'photosynthesis'

leaf_par = make_leafpar(
  replace = list(
    T_leaf = set_units(c(293.14, 298.15), "K")
  ), use_tealeaves = FALSE
)
photosynthesis(leaf_par, enviro_par, bake_par, constants, use_tealeaves = FALSE)
```

---

photo_parameters

**Input parameters to simulate C3 photosynthesis using**

photosynthesis()
Description
A table of input parameters used in `photosynthesis()`

Usage
`photo_parameters`

Format
`photo_parameters`: A data frame with 73 rows and 11 columns:
- `country` Country name
- `iso2, iso3` 2 & 3 letter ISO country codes
- `year` Year ...

Source
https://www.who.int/teams/global-tuberculosis-programme/data

---

**ppm2pa**

*Convert pressure from PPM to Pascals*

Description
Convert pressure from PPM to Pascals

Usage
`ppm2pa(ppm, P)`

Arguments
- `ppm` Pressure value in umol/mol of class `units`
- `P` Atmospheric pressure value in kPa of class `units`

Details

\[
\text{Press}(kPa) = \text{Press}(ppm)P(kPa) \\
\text{Press}(Pa) = 1000\text{Press}(kPa)
\]

Value
Value in Pa of class `units`
print_graphs

Examples

ppm = set_units(400, "umol/mol")
P = set_units(101.325, "kPa")
ppm2pa(ppm, P)

print_graphs

Printing graphs to system

Description

Printing graphs to system

Usage

print_graphs(
    data,
    path,
    output_type = "jpeg",
    height = 5,
    width = 5,
    res = 600,
    units = "in",
    pdf_filename,
    ...
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>List of graphs</td>
</tr>
<tr>
<td>path</td>
<td>File path for printing our graphs. Use &quot;./&quot; to set to current working directory</td>
</tr>
<tr>
<td>output_type</td>
<td>Type of output file, jpeg or pdf</td>
</tr>
<tr>
<td>height</td>
<td>Height of jpegs</td>
</tr>
<tr>
<td>width</td>
<td>Width of jpegs</td>
</tr>
<tr>
<td>res</td>
<td>Resolution of jpegs</td>
</tr>
<tr>
<td>units</td>
<td>Units of height and width</td>
</tr>
<tr>
<td>pdf_filename</td>
<td>Filename for pdf option</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments for jpeg() and pdf()</td>
</tr>
</tbody>
</table>

Value

print_graphs creates graph files in current working directory from a list of graphs
Examples

# Read in your data
# Note that this data is coming from data supplied by the package
# hence the complicated argument in read.csv()
# This dataset is a CO2 by light response curve for a single sunflower
data <- read.csv(system.file("extdata", "A_CI_Q_data_1.csv", 
    package = "photosynthesis"
))

# Fit many AQ curves
# Set your grouping variable
# Here we are grouping by CO2_s and individual
data$C_s <- (round(data$CO2_s, digits = 0))

# For this example we need to round sequentially due to CO2_s setpoints
data$C_s <- as.factor(round(data$C_s, digits = -1))

# To fit one AQ curve
fit <- fit_aq_response(data[data$C_s == 600, ],
    varnames = list(  
    A_net = "A",
    PPFD = "Qin"
    )
)

# Print model summary
summary(fit[[1]])

# Print fitted parameters
fit[[2]]

# Print graph
fit[[3]]

# Fit many curves
fits <- fit_many(
    data = data,
    varnames = list(  
    A_net = "A",
    PPFD = "Qin",
    group = "C_s"
    ),
    funct = fit_aq_response,
    group = "C_s"
)

# Look at model summary for a given fit
# First set of double parentheses selects an individual group value
# Second set selects an element of the sublist
summary(fits[[3]][[1]])
# Print the parameters
fits[[3]][[2]]

# Print the graph
fits[[3]][[3]]

# Compile graphs into a list for plotting
fits_graphs <- compile_data(fits,
  list_element = 3
)

# Print graphs to pdf
# Uncomment to run
# print_graphs(data = fits_graphs,
# output_type = "pdf",
# path = tempdir(),
# pdf_filename = "mygraphs.pdf")

---

**read_li6800**  
*Read a LI-COR file*

**Description**

[Deprecated]

We are no longer updating this function. Please use `read_licor` instead.

**Usage**

`read_li6800(x)`

**Arguments**

- `x`  
  File name

**Value**

Returns a data.frame from raw LI-COR files. Current support for LI-COR LI-6800 files only.
### read_licor

**Description**

*Experimental*

Reads a raw LI-COR LI6800 file, including remarks. This function was developed using output from Bluestem v.2.0.04 to v.2.1.08. We cannot guarantee backward compatibility with earlier versions of Bluestem. We will try to update code when new versions are released, but there may be a time-lag, so inspect results carefully.

**Usage**

```r
read_licor(
  file,
  bluestem_version = get_bluestem_version(file, n_max = 10L),
  ...
)
```

**Arguments**

- `file`: Path to a raw LI6800 file
- `bluestem_version`: Character string of Bluestem software version number. By default, the function will try to pull the version number from file.
- `...`: Argument passed to `read_lines`

**Value**

Returns a *tibble* from raw LI-COR LI6800 files.

---

### required_variables

**Description**

Variables required for *photosynthesis* models

**Usage**

```r
required_variables(.model, quiet)
```

**Arguments**

- `.model`: A character string of model name to use. See `get_all_models()`.
- `quiet`: Flag. Should messages be suppressed? Default is FALSE.
Simulate gas exchange data with measurement error

Description

[Experimental]

Usage

simulate_error(
  ph_out,
  chamber_pars,
  n = 1L,
  use_tealeaves = ("T_air" %in% colnames(ph_out))
)

Arguments

- **ph_out**: A data frame of output from photo() or photosynthesis() with units.
- **chamber_pars**: A data frame with a single row of chamber parameters. See Note below for table of required parameters.
- **n**: Integer. Number of replicated simulations per row of ph_out.
- **use_tealeaves**: Flag. The tealeaves package uses a slightly different equation to calculate the saturating water content of air as a function temperature and pressure than LI-COR. If FALSE, the function uses LI-COR's equation in the LI6800 manual. If TRUE, it uses the tealeaves function for internal consistency. The function attempts to guess whether ph_out was run with tealeaves, but this can be manually overridden by providing a value for the argument.

Value

A data frame with \( n \times nrow(ph\_out) \) rows. It contains all the original output in ph_out as well as a column .rep indicating replicate number from 1 to n. Other new columns are assumed or measured chamber parameters and 'measured' values estimated from synthetic data with measurement error:

<table>
<thead>
<tr>
<th>column name</th>
<th>assumed or derived?</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>assumed</td>
<td>chamber flow rate</td>
</tr>
<tr>
<td>leaf_area</td>
<td>assumed</td>
<td>leaf area in chamber</td>
</tr>
<tr>
<td>sigma_CO2_r</td>
<td>assumed</td>
<td>standard deviation of measurement error in CO2_r</td>
</tr>
<tr>
<td>sigma_CO2_s</td>
<td>assumed</td>
<td>standard deviation of measurement error in CO2_s</td>
</tr>
<tr>
<td>sigma_H2O_r</td>
<td>assumed</td>
<td>standard deviation of measurement error in H2O_r</td>
</tr>
<tr>
<td>sigma_H2O_s</td>
<td>assumed</td>
<td>standard deviation of measurement error in H2O_s</td>
</tr>
<tr>
<td>c_0</td>
<td>derived</td>
<td>CO2 concentration before entering chamber [( \mu \text{mol} / \text{mol} )]</td>
</tr>
<tr>
<td>w_i</td>
<td>derived</td>
<td>Water vapor concentration within leaf [mmol / mol]</td>
</tr>
<tr>
<td>w_a</td>
<td>derived</td>
<td>Water vapor concentration in chamber [mmol / mol]</td>
</tr>
<tr>
<td>w_0</td>
<td>derived</td>
<td>Water vapor concentration before entering chamber [mmol / mol]</td>
</tr>
</tbody>
</table>
### Note

The required parameters for the `chamber_pars` argument are:

- `flow` [µmol/s]: chamber flow rate
- `leaf_area` [cm²]: leaf area in chamber
- `sigma_CO2_s` [µmol/mol]: standard deviation of sample [CO₂] measurement error
- `sigma_CO2_r` [µmol/mol]: standard deviation of reference [CO₂]
- `sigma_H2O_s` [mmol/mol]: standard deviation of sample [H₂O] measurement error
- `sigma_H2O_r` [mmol/mol]: standard deviation of sample [H₂O] measurement error

Units for `flow` and `leaf_area` should be provided; units are implied for sigma’s but not necessary to specify because `rnorm()` drop units.

To evaluate the accuracy and precision of parameter estimation methods, it may be useful to simulate data with realistic measurement error. This function takes output from from `photo()` or `photosynthesis()` models, adds measurement error in CO₂ and H₂O concentrations, and calculates parameter estimates with synthetic data. Currently, the function assumes a simplified 1-dimensional CO₂ and H₂O conductance model: zero cuticular conductance, infinite boundary layer conductance, and infinite airspace conductance. Other assumptions include:

- chamber flow rate, leaf area, leaf temperature, and air pressure are known without error
- measurement error is normally distributed mean 0 and standard deviation specified in `chamber_pars`

This function was designed with the LI-COR LI6800 instrument in mind, but in principle applies to any open path gas exchange system.

### Examples

```r
library(photosynthesis)

# Use photosynthesis() to simulate 'real' values
# 'replace = ...' sets parameters to meet assumptions of 'simulate_error()'
```
t_response_arrhenius

lp = make_leafpar(replace = list(
    g_sc = set_units(0.1, mol/m^2/s),
    g_uc = set_units(0, mol/m^2/s),
    k_mc = set_units(0, 1),
    k_sc = set_units(0, 1),
    k_uc = set_units(0, 1)
),
    use_tealeaves = FALSE)

ep = make_enviropar(replace = list(
    wind = set_units(Inf, m/s)
), use_tealeaves = FALSE)
bp = make_bakepar()
cs = make_constants(use_tealeaves = FALSE)

chamber_pars = data.frame(
    flow = set_units(600, umol / s),
    leaf_area = set_units(6, cm ^ 2),
    sigma_CO2_s = 0.1,
    sigma_CO2_r = 0.1,
    sigma_H2O_s = 0.1,
    sigma_H2O_r = 0.1
)

ph = photosynthesis(lp, ep, bp, cs, use_tealeaves = FALSE, quiet = TRUE) |> 
    simulate_error(chamber_pars, n = 1L)

t_response_arrhenius **Temperature response functions**

**Description**

Temperature response functions

**Usage**

t_response_arrhenius(T_leaf, Ea)

t_response_arrhenius_kruse(dEa, Ea_ref, Par_ref, T2)

t_response_arrhenius_medlyn(T_leaf, Ea, Hd, dS)

t_response_arrhenius_topt(T_leaf, Ea, Hd, Topt)

t_response_calc_dS(Ea, Hd, Topt)

t_response_calc_topt(Hd, dS, Ea)
t_response_heskel(T_leaf, a, b, c)

t_response_mmrt(dCp, dG, dH, T_leaf)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_leaf</td>
<td>Leaf temperature in K</td>
</tr>
<tr>
<td>Ea</td>
<td>Activation energy in J mol⁻¹ (Medlyn et al. 2002)</td>
</tr>
<tr>
<td>dEa</td>
<td>Temperature-dependent change in Ea in K² (Kruse et al. 2008)</td>
</tr>
<tr>
<td>Ea_ref</td>
<td>Activation energy in J mol⁻¹ (Kruse et al. 2008)</td>
</tr>
<tr>
<td>Par_ref</td>
<td>Parameter at reference temperature of 25 Celsius (Kruse et al. 2008)</td>
</tr>
<tr>
<td>T2</td>
<td>Leaf temperature term (Kruse et al. 2008)</td>
</tr>
<tr>
<td>Hd</td>
<td>Deactivation energy in J mol⁻¹ (Medlyn et al. 2002)</td>
</tr>
<tr>
<td>dS</td>
<td>Entropy parameter in J mol⁻¹ (Medlyn et al. 2002)</td>
</tr>
<tr>
<td>Topt</td>
<td>Optimum temperature of the process in K (Medlyn et al. 2002)</td>
</tr>
<tr>
<td>a</td>
<td>Constant to minimize residuals (Heskel et al. 2016)</td>
</tr>
<tr>
<td>b</td>
<td>Linear coefficient to minimize residuals (Heskel et al. 2016)</td>
</tr>
<tr>
<td>c</td>
<td>Quadratic coefficient to minimize residuals (Heskel et al. 2016)</td>
</tr>
<tr>
<td>dCp</td>
<td>Change in heat capacity of the enzyme between the enzyme-substrate #’ and</td>
</tr>
<tr>
<td></td>
<td>enzyme-transition states in J mol⁻¹ K⁻¹ (Hobbs et al. 2013)</td>
</tr>
<tr>
<td>dG</td>
<td>Change in Gibbs free energy of the reaction at 25 C in J mol⁻¹ (Hobbs et al.</td>
</tr>
<tr>
<td></td>
<td>2013)</td>
</tr>
<tr>
<td>dH</td>
<td>Change in enthalpy of the reaction at 25 C in J mol⁻¹ (Hobbs et al. 2013)</td>
</tr>
</tbody>
</table>

**Value**

- `t_response_arrhenius` calculates the rate of a process based on an Arrhenius-type curve
- `t_response_arrhenius_kruse` fits a peaked Arrhenius response according to Kruse et al. 2008.
- `t_response_arrhenius_medlyn` is a peaked Arrhenius response as found in Medlyn et al. 2002.
- `t_response_arrhenius_topt` is a peaked Arrhenius temperature response function.
- `t_response_calc_dS` calculates dS from the fitted Topt model.
- `t_response_calc_topt` calculates Topt for a process from Arrhenius parameters.
- `t_response_heskel` is a quadratic temperature response according to Heskel et al. 2016.
- `t_response_mmrt` is a macromolecular rate theory temperature response according to Hobbs et al. 2013.

**References**


Hobbs et al. 2013. Change in heat capacity for enzyme catalysis determines temperature dependence of enzyme catalyzed rates. ACS Chemical Biology 8:2388-2393


Index

* datasets
  - photo_parameters, 65
    .get_gbc (CO2_conductance), 12
    .get_gbc(), 13, 15
    .get_gmc (CO2_conductance), 12
    .get_gsc (CO2_conductance), 12
    .get_gtc (CO2_conductance), 12
    .get_guc (CO2_conductance), 12
  - A_demand (A_supply), 7
  - A_supply, 7
  - analyze_sensitivity, 4
  - aq_response, 6
  - bake, 8, 15
    bake(), 10, 15, 63
    bake_par, 10
    bake_par(), 9, 60
    baked-class, 10
    brms::brm(), 26, 38, 41, 54
    brms::brmsfit(), 26, 41
  - calculate_j (calculate_jmax), 11
    calculate_jmax, 11
    calculated, 7, 13, 53, 56
    calculated-parameters, 11
    CO2_conductance, 12
    compile_data, 15
    compute_sensitivity, 16
    constants, 18
    constants(), 60, 63
    cross_df(), 63
  - enviro_par, 19
    enviro_par(), 60, 63
  - fit_aci_response, 20
  - fit_aq_response, 23
    fit_aq_response2, 26
    fit_aq_response2(), 38, 54
  - fit_g_mc_variableJ, 31
  - fit_gs_model, 28
    fit_hydra_vuln_curve, 33
  - fit_many, 35
    fit_photosynthesis, 37
    fit_photosynthesis(), 26, 41
    fit_pV_curve, 38
  - fit_r_light2, 41
  - fit_r_light2(), 38, 54
    fit_r_light_kok, 44
    fit_r_light_WalkerOrt
      (fit_r_light_kok), 44
    fit_r_light_yin (fit_r_light_kok), 44
    fit_t_response, 47
  - furrr::future_map(), 63
  - FvCB, 51
  - FvCB(), 63
  - get_all_models (get_default_model), 54
    get_all_models(), 26, 38, 41, 70
    get_default_model, 54
  - get_f_parameter
    (calculated-parameters), 11
    gs_mod_ballberry, 55
    gs_mod_leuning (gs_mod_ballberry), 55
    gs_mod_opti (gs_mod_ballberry), 55
    gs_mod_optifull (gs_mod_ballberry), 55
    J, 56
  - leaf_par, 57
    leaf_par(), 9, 60, 63
    list(), 9
  - make_bakepar (make_parameters), 57
  - make_constants (make_parameters), 57
    make_constants(), 9
  - make_enviropar (make_parameters), 57
  - make_leafpar (make_parameters), 57
  - make_parameters, 57
    make_parameters(), 19, 57, 58, 61, 63
INDEX

map, 36
marshall_biscoe_1980
   (get_default_model), 54

parameter_names, 61
photo (photosynthesis), 62
photo(), 63
photo_parameters, 65
photosynthesis, 11, 62
photosynthesis(), 63, 65, 66
photosynthesis-package, 3
ppm2pa, 66
print_graphs, 67

read_li6800, 69
read_licor, 69, 70
read_lines, 70
required_variables, 70
required_variables(), 26, 37, 38, 41

simulate_error, 71
stats::lm(), 41
stats::nls(), 26, 41

t_response_arrhenius, 73
t_response_arrhenius_kruse
   (t_response_arrhenius), 73
t_response_arrhenius_medlyn
   (t_response_arrhenius), 73
t_response_arrhenius_topt
   (t_response_arrhenius), 73
t_response_calc_dS
   (t_response_arrhenius), 73
t_response_calc_topt
   (t_response_arrhenius), 73
t_response_heskel
   (t_response_arrhenius), 73
t_response_mmrt (t_response_arrhenius), 73
tleaves:::get_Dx(), 15
temp Resp1 (bake), 8
temp Resp2 (bake), 8
tibble, 70
tleaf(), 19, 57, 58, 61, 63

W_carbox (FvCB), 51
W_regen (FvCB), 51
W_tpu (FvCB), 51