AoB PLANTS 2021, Vol. 13, No. 5

https://doi.org/10.1093/aobpla/plab059
Advance Access Publication September 19, 2021
Tools

TOOLS

Principles of resilient coding for plant ecophysiologists

Jospeh R. Stinziano[1], Cassaundra Roback[1], Demi Sargent[2], Bridget K. Murphy[3], Patrick J. Hudson[1] and Christopher D. Muir[4,*]

[1]Department of Biology, University of New Mexico, Albuquerque, NM 87131, USA, [2]Hawkesbury Institute for the Environment, Western Sydney University, Sydney 2753, Australia, [3]Department of Biology, University of Toronto, Mississauga L5L 1C6, Canada, [4]School of Life Sciences, University of Hawai'i, Honolulu, HI 96822, USA

*Corresponding author's e-mail address: cdmuir@hawaii.edu

Form & Function. Chief Editor: Kate McCulloh

Associate Editor: Markus Hauck

Abstract

Plant ecophysiology is founded on a rich body of physical and chemical theory, but it is challenging to connect theory with data in unambiguous, analytically rigorous and reproducible ways. Custom scripts written in computer programming languages (coding) enable plant ecophysiologists to model plant processes and fit models to data reproducibly using advanced statistical techniques. Since many ecophysiologists lack formal programming education, we have yet to adopt a unified set of coding principles and standards that could make coding easier to learn, use and modify. We identify eight principles to help in plant ecophysiologists without much programming experience to write resilient code: (i) standardized nomenclature, (ii) consistency in style, (iii) increased modularity/extensibility for easier editing and understanding, (iv) code scalability for application to large data sets, (v) documented contingencies for code maintenance, (vi) documentation to facilitate user understanding; (vii) extensive tutorials and (viii) unit testing and benchmarking. We illustrate these principles using a new R package, {photosynthesis}, which provides a set of analytical and simulation tools for plant ecophysiology. Our goal with these principles is to advance scientific discovery in plant ecophysiology by making it easier to use code for simulation and data analysis, reproduce results and rapidly incorporate new biological understanding and analytical tools.

Keywords: Curve fitting; gas exchange; hydraulics; modelling; photosynthesis; R; software; stomatal conductance.

Background

Computer coding is becoming an increasingly important skill in biological research (Sayres *et al.* 2018), especially within plant ecophysiology. A disconnect in coding skill and a lack of formal computer science training can make it difficult for biologists to create or modify programs to incorporate new understanding of biological processes. In other words, sophisticated code (by trained programmers) is efficient, but difficult to modify by biologists for new uses. So why code at all? Coding allows for consistent, reproducible, transparent and scalable analyses of scientific data, while at the same time minimizing human work hours compared to using pre-packaged software. However, most published ecophysiological analyses use spreadsheet-based methods rather than computer code, which comes with some limitations. For example, Sharkey *et al.* (2007) have an Excel spreadsheet-based method for fitting photosynthetic CO_2 response (A–C_i) curves (also see Bellasio *et al.* 2016). A spreadsheet-based method can take several minutes per curve and involves a substantial amount of subjective decision-making (e.g. 'eye-balling' where transitions between CO_2- and RuBP-limited photosynthesis occur). Likewise, analysis of pressure–volume curves for hydraulic parameters is usually done via an Excel spreadsheet-based method (Sack *et al.* 2003),

Received: 24 September 2020; **Editorial decision:** 9 September 2021; **Accepted:** 15 September 2021

which can be time-consuming, requires subjective decisions, and spreadsheets are usually not published with manuscripts, obscuring methodology. The total workload is time per spreadsheet multiplied by the number of curves, which can be inefficient in large studies. Cryptic changes in the spreadsheets can occur without a record of the change, potentially leading to compounding errors. Furthermore, spreadsheet tools often break, requiring a fresh, unaltered spreadsheet to be used for each CO_2 response curve. Another option, provided by Gu *et al.* (2010) (leafweb.org) provides an online service that analyses $A–C_i$ curves; however, in this case, the analysis is a black-box and could be misused by users lacking an understanding of the fitting process, and the data are stored on a government server which may cause some users discomfort.

Meanwhile, Duursma (2015) developed an R package, {plantecophys}, that can obtain similar outputs to the Sharkey *et al.* (2007) fitting tools in seconds, with far fewer subjective decisions that can easily be outlined in the code used in the fitting process, while providing a similar, but transparent approach as in Gu *et al.* (2010). Like the {plantecophys} package, analytical methods should be fully transparent and reproducible. As such, authors should publish their code, which is still not the norm in plant ecophysiology (but see Kumarathunge *et al.* 2019 for an example of published code). As a community, increased adoption and dissemination of code will help the field perform more sophisticated analyses and model comparison (e.g. Walker *et al.* 2021). Coding may also streamline integration between theory and data analysis, especially for complex mathematical formulations that require computationally intensive numerical methods, a common situation in plant ecophysiology. Ideally, we would like a workflow in which we state our assumptions mathematically, derive empirical predictions, and test those predictions or estimate parameters with data. The process of translating a mathematical model of biology into code can also help novice and advanced coders deepen their understanding of models and their assumptions before confronting them with data. Open-source, research-grade computer algebra systems like SymPy (Meurer *et al.* 2017) and numerical solvers aid mathematical derivation and are part of or can be readily integrated with programming languages that are widely used for data manipulation and analysis, such as R (R Core Team 2021), Python (Python Software Foundation) or Julia (Bezanson *et al.* 2017).

Although coding can speed up large analyses, reduce errors, make analyses reproducible and integrate theory with data, writing robust code that can be understood and reused by other scientists is not easy. First, one must learn one or more programming languages (e.g. R, Python, Matlab, Julia), which can involve steep learning curves. Second, even though coding one's own analysis can make it easier to catch errors associated with inappropriate use of black-box proprietary software, one must still understand the assumptions and limitations of statistical techniques and conceptual tools. Finally, code can be as unique as someone's handwriting, which can make it difficult even for an experienced programmer to make sense of a 'transparent' analysis unless there is sufficient annotation within the code.

In this perspective, we propose eight principles of coding tailored to the specific needs of the plant ecophysiology research community. For example, guidance in other scientific fields often emphasizes computational speed. However, given the typical scale of ecophysiological data sets (~MB, i.e. small-batch, artisanal data sets) and the computer power of personal computers (~GB of RAM, ~GHz of processing power), computational speed is usually not a major limitation. Instead, ecophysiologists often need to estimate parameters derived from complex biophysical/chemical models. Coding is important as the complex models required to fit different response curves involve many interacting equations, numerical solvers and parameters that either need to be set or estimated. For example, there are seven different models that can be used to fit temperature responses which ultimately require different equations and fixed parameters (Arrhenius 1915; Johnson *et al.* 1942; Medlyn *et al.* 2002; Kruse *et al.* 2006; Heskel *et al.* 2016; Liang *et al.* 2018). In this domain, code flexibility and modularity are usually more important than computational speed. Furthermore, flexibility and modularity in code would enhance the sustainability of software after publication, which can be an issue (Prlić and Procter 2012). Here we demonstrate coding principles designed for plant ecophysiology using a new R package called {photosynthesis}. We caution that this software is a work-in-progress that does not yet completely adhere to all of the coding principles to which we aspire, but will be refined in future releases. This perspective, written by trained biologists not programmers, is intended to convey some of the lessons we have learned so far to provide guidance for plant ecophysiologists who are thinking about or starting to code their workflows, especially using R. We recognize that many other scientists in this field are adept coders who have already honed their practices through experience. Hence, this perspective is intended to guide for less experienced coders rather than a mandate for the entire field. We hope our perspective spurs experienced coders to share 'best practices' with less experienced peers and expand the principles below to other languages besides R. As computational plant ecophysiology matures, we hope that this perspective will help move the field toward more standardized and sustainable software practices like those in more computationally intensive subfields of biology like population genetics (Adrion *et al.* 2020).

Description

Principles of coding

The overarching concept we propose is making code resilient by making it easier to use, reproduce and modify. Obviously not every possible discovery and need within a scientific field can be predicted, but the code can be written to allow easy modification and accommodation of the source code as the science progresses. Functional programming in R and other languages provides a powerful tool for writing functions that take functions as arguments and easily process newly written code into a standardized output without the need for ever modifying the original function itself (Wickham 2019). Such an approach helps to write modular code that is easy to modify and understand, while minimizing interdependencies between functions.

Freely available resources already exist for good coding practices in R packages and can be applied to R scripts as well, primarily from the efforts of Hadley Wickham (Wickham 2014, 2015, 2016b, 2017, 2019; Wickham and Grolemund 2016). As well, guides to best practices for scientific computing exist (see Wilson *et al.* 2014 for a list of best practices). Here we propose principles of coding for plant ecophysiology that, if implemented, could circumvent some of the common coding issues encountered when modifying the code of others, reduce the learning difficulty for nascent coders and make software maintenance much easier:

1. Standardized nomenclature for variables and functions
2. Consistent style

3. Modularity and extensibility
4. Scalability
5. Documented contingencies
6. Documentation
7. Extensive tutorials
8. Unit testing and benchmarking

We think that adopting some or all of these principles will improve code reproducibility and help advance scientific discovery, but our goal is not to rigidly prescribe how plant ecophysiologists should do their work. First, we recognize that others will have different, well-reasoned preferences and/or apply principles we have not covered here. Second, those who find these principles useful may find implementing all of them time-consuming at first. We strongly encourage incremental progress and not making perfection the enemy of the good. Indeed, the {photosynthesis} package described below only partially implements our principles, with much left to do in future development.

Principle 1: standardized nomenclature. Names vary wildly between functions with published code and data and even amongst instruments within the same company (e.g. for net CO_2 assimilation, 'A' is used in the Li-Cor 6800 and 'PHOTO' is used in the Li-Cor 6400). Ideally, we need both standardized nomenclature in the field (e.g. Reid *et al.* 2005) and standardized construction of variable and function names to enhance readability and reduce the burden for learning how to use new packages and functions or testing published code. For example, *g* is always in reference to conductance, where a subscript term would then describe the physical pathway (e.g. s for stomata, c for cuticle or m for mesophyll) as well as the gas (e.g. c for CO_2, w for water vapour). For example, g_{sw} would mean stomatal conductance to water vapour. Standardizing nomenclature across both mathematical models and data files can also streamline theory–data integration, but this also requires standard translation between mathematical and computer notation, which is beyond our scope here.

For example, in {photosynthesis}, every function is named in a descriptive manner: e.g. `fit_t_response` fits specified temperature responses model to data, while `fit_gs_model` fits specified models of stomatal conductance. Variable names are also standardized: e.g. 'T_leaf' always means leaf temperature in Kelvin (K), 'A_net' always means net CO_2 assimilation in μmol m^{-2} s^{-1}. In this regard, standard units should also be imposed in the analysis (e.g. in R via the {units} package; Pebesma *et al.* 2016), to remove any ambiguities when interpreting the output. To allow for differences in variable names from the raw data (e.g. from using different machines), the 'varnames' list is used to translate input names (note that this convention is adopted from {plantecophys}; Duursma 2015). We propose adopting Wickham's (Wickham 2019) style in that functions that *do* something have a verb name, e.g. `fit_aci_response`, while functions that act as objects within other functions (e.g. stomatal conductance models) should have a noun name, e.g. `gs_model`.

Principle 2: consistent style. Consistent coding style makes reading code easier—certain conventions, e.g. commenting what the *next* line of code does, can make it easier to understand code documentation. Our preference is for the 'tidy style', which applies to both data and code structure, and much else (see the *The tidyverse Style Guide*: https://style.tidyverse.org/). For data, tidy style advocates that each column is a variable, and each row is an observation, since R is particularly suited for this style of data structure. Popular R packages like {dplyr} (Wickham *et al.* 2020) and {tidyr} (Wickham and Henry 2020) facilitate tidy data and many other packages, like {photosynthesis}, use them for consistent style (Notes S3 contains an example of tidy data organization). For code, computers do not care about style, as long as it is correctly formatted, but for humans reading code, adherence to well-designed style can be helpful, especially for beginners trying to learn from others. A benefit of tidy style in particular is that R packages {styler} (Müller and Walthert 2020), {lintr} (Hester *et al.* 2020) and {formatR} (Xie 2019) can automate conformity to style. Ideally, a consistent style would be adopted across the field; however, this may be too rigid. Style can be highly personal, and many experienced coders likely have developed their own style, formal or informal, that works for them. Our proposal is geared for beginning coders who are looking for guidance on an established and easy-to-implement style. At the very least, a consistent style *within* a project will make it easier to read, understand and modify the code.

Principle 3: modularity and extensibility. Arguably, code written for plant ecophysiologists, whether formally trained in coding or not, should be written in a modular manner, much like Lego bricks, where one component (e.g. Arrhenius function) can be easily swapped with another (e.g. peaked Arrhenius function), or extended (e.g. hypothetical mechanistic temperature response model). Note that this may increase apparent complexity of software packages by creating more functions and make it more difficult to work with at first. However, it will make adding, subtracting or modifying code modules easier for researchers who need to make on-the-fly changes to code as new biological processes are discovered or old ones re-evaluated. To achieve modularity in the structure of photosynthesis, we used principles of functional programming to develop a set of key functions for processing data and running quality control checks: `fit_many`, `analyze_sensitivity`, `compile_data` and `print_graphs`. Both `fit_many` and `analyze_sensitivity` can be run with any function within and outside of {photosynthesis} to run multiple curve fits or sensitivity analyses on assumed input parameters. Meanwhile, `compile_data` is used for processing the list outputs from `fit_many` into a form usable for further analyses and export from R, and `print_graphs` is used to export all graphs from a list as either .jpeg or compiled as a .pdf.

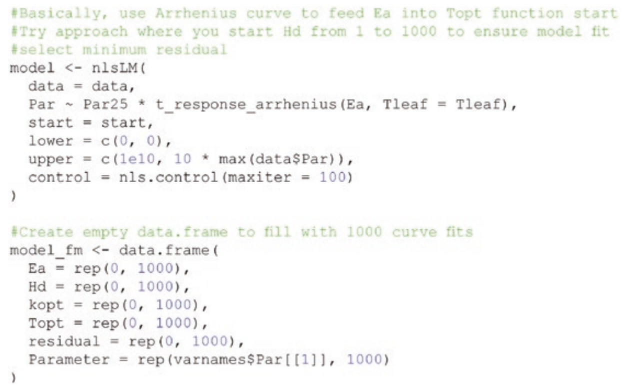
For curve fitting functions with multiple models (e.g. temperature responses, g_s models), we use a basic function (e.g. `fit_t_response`), which contains fitting procedures for each of the seven temperature response models in the package. Meanwhile, a `t_functions` file contains all the temperature response functions. To extend the capabilities and add in a new temperature response model, we simply need to add the new model to `t_functions`, and the fitting procedure to `fit_t_response`. Currently, adding new functions requires modifying the source code, but future versions should increase extensibility by allowing users to supply any temperature response function. This principle of function building increases the extensibility of the code, while consistent style and standardized nomenclature provide the rules for writing the extended components.

Modularity also applies to modelling. The {photosynthesis} functions `photo` and `photosynthesis` model C3 photosynthesis using the Farquhar–von Caemmerer–Berry biochemical model (Farquhar *et al.* 1980). To account for temperature dependence, a user can specify leaf temperature, or they can provide additional inputs (e.g. air temperature, leaf size, wind speed, etc.) to model leaf temperature using energy balance in the R package {tealeaves} (Muir 2019). Both {photosynthesis} and {tealeaves} packages are modular in that they can work independently or be readily integrated **[see Supporting Information—**Methods S1]. Ideally, future modelling packages would add modules to model environmental and plant parameters either on their own or integrated with these tools.

Principle 4: scalability. A major advantage in using code to analyse data is the ability to scale up an analysis to reduce time spent on repetitive tasks common in spreadsheet-based methods such as copy-and-paste, selecting data, choosing menu options, etc. Functions allow the same model to be fit across groups within a data set using a consistent method. For this, our `fit_many` function and the principles of functional programming are how we achieve scalability within the package. Rather than writing functions for each type of model or curve, we have a single multiple fitting function, sensitivity analysis function and printing function. R even has generic functions for scaling such as `apply` (base R language) and `map` ({purrr} package; Henry and Wickham 2020) which can be easily parallelized for speed (e.g. {parallel} and {furrr}; Vaughan and Dancho 2018 packages). This makes it easy to scale a new function within the software to a large data set.

Principle 5: documented contingencies. By documenting which functions are dependent on one another, it becomes easier to troubleshoot issues when modifying code and to pre-empt issues when adding or replacing a component. For example, `fit_aq_response` depends on `aq_response`—if we want to change from the non-rectangular hyperbola model to a rectangular hyperbolic model, then `fit_aq_response` needs to be modified in addition to `aq_response`. To document contingencies, we created a function, `check_dependencies`, which uses {pkgnet} (Burns *et al.* 2020) to generate an html report that automatically documents R package interdependencies and function interdependencies. This is particularly useful when adding, subtracting or modifying functions in the package, as it allows planning to minimize issues that could break code.

Principle 6: documentation. Code annotations allow a new user to readily understand what a line of code is doing, how it is doing it and why the code is written in a particular way. By providing exhaustive line-by-line annotation of a function, a new user can more rapidly understand the blueprint of the function. This is especially useful for R scripts and code hosted



```
#Basically, use Arrhenius curve to feed Ea into Topt function start
#Try approach where you start Hd from 1 to 1000 to ensure model fit
#select minimum residual
model <- nlsLM(
  data = data,
  Par ~ Par25 * t_response_arrhenius(Ea, Tleaf = Tleaf),
  start = start,
  lower = c(0, 0),
  upper = c(1e10, 10 * max(data$Par)),
  control = nls.control(maxiter = 100)
)

#Create empty data.frame to fill with 1000 curve fits
model_fm <- data.frame(
  Ea = rep(0, 1000),
  Hd = rep(0, 1000),
  kopt = rep(0, 1000),
  Topt = rep(0, 1000),
  residual = rep(0, 1000),
  Parameter = rep(varnames$Par[[1]], 1000)
)
```

Figure 1. Example of coding annotations to explain the given analytical approach.

on GitHub (unfortunately, comments are erased from code upon submission to CRAN). For example, in `fit_t_response`, we outline the need for running looped iterations for the starting values of non-linear least squares curve fitting (Fig. 1). In the case of R packages hosted on CRAN, R documentation files provide information on how to use a function, though as a terser set of instructions as per CRAN policies (https://cran.r-project.org/doc/manuals/r-devel/R-exts.html).

Enough metadata and commenting should be provided for a new user to understand how to use the written code (which can be an issue that affects widespread use of a program; Mangul *et al.* 2019).

Principle 7: extensive tutorials. As with any tool, software will only be used if potential users can understand how it works. Extensive tutorials, while providing function-by-function examples of how to use the software, should also incorporate basic data-wrangling examples and explanations of why a given approach to data analysis is used in the field. The benefits of this approach include: making the code easier to adopt into your own analysis, making it easier for new coders to learn enough of the language to use the package effectively, and help trainees learn the appropriate theory behind the measurements and analytical approach. The net effect should be to increase the inclusivity of the field by reducing barriers to success since not all individuals will have equal access to workshops or experienced colleagues.

Principle 8: unit testing and benchmarking. For reproducibility, code should yield the same results when it is run by other users months or years into the future. Unit testing, a common practice in software development that is still rare in scientific code, evaluates whether various components, such as custom functions, perform as expected. If all the components work as expected, it provides confidence that the whole body of code does what it is supposed to. Most scientists informally test their functions as they develop them, but formal unit testing involves writing scripts to test code and can be rerun to periodically check whether code still works as expected. More dedicated efforts automate testing and quantify code coverage, the fraction of code that is evaluated during automated tests. There are many ways to implement unit testing, but the {testthat} package is one option for R packages (Wickham 2011) that {photosynthesis} uses for some (but not yet all) of its source code. A related concept is benchmarking, by which we mean comparing parameter estimates from the same data set using different software or later versions of the same software. Benchmarking can help determine if parameter estimation is consistent between software packages. For example, parameter estimates of photosynthetic CO_2 response parameters (Farquhar *et al.* 1980) are very similar using comparable settings in {photosynthesis} and {plantecophys} **[see Supporting Information—Notes S1]**.

Examples of resilient coding in the {photosynthesis} package for R

We built a package containing analytical tools for plant ecophysiology (Stinziano *et al.* 2020), embedding our coding principles into the package itself. The R package contains functions for fitting photosynthetic CO_2 (Farquhar *et al.* 1980; von Caemmerer 2000; Gu *et al.* 2010; Duursma 2015) and light response curves (Marshall and Biscoe 1980), temperature responses of biological processes (Arrhenius 1915; Medlyn *et al.* 2002; Kruse *et al.* 2006; Heskel *et al.* 2016; Liang *et al.* 2018), light respiration (Kok 1956; Laisk 1977; Yin *et al.* 2009, 2011; Walker and Ort 2015), mesophyll conductance (Harley *et al.* 1992), stomatal conductance models (Ball *et al.* 1987; Leuning 1995; Medlyn *et al.* 2011), pressure–volume curves (Tyree and Hammel 1972; Koide *et al.* 2000; Sack *et al.* 2003), hydraulic vulnerability curves (Pammenter and van der Willigen 1998; Ogle *et al.* 2009) and sensitivity analyses (Table 1; **see Supporting Information—Table S1**). It also contains functions for modelling C_3 photosynthesis using the Farquhar–von Caemmerer–Berry biochemical model (Farquhar *et al.* 1980). The default kinetic parameters for gas exchange fitting procedures are taken from *Nicotiana tabacum* (Bernacchi *et al.* 2001, 2002). The {photosynthesis} package is currently limited to C_3 photosynthesis, but future releases should expand its functionality to other photosynthetic pathways. A comprehensive illustration of how to use the package can be found in the vignette of the package (**see Supporting Information—Notes S2**, 'photosynthesis-curve-fitting-sensitivity-analyses.rmd'). There are currently two vignettes available for the package that function as tutorials on CRAN (https://CRAN.R-project.org/package=photosynthesis).

Table 1. List of {photosynthesis} functions with applications and descriptions. The documentation for each function describes the estimated or simulated parameters, constants and other calculated values. Documentation is updated to describe new functionalities as they are added.

Base functions		
Applications	Function	Description
Gas Exchange	`fit_aci_response`	Fits A–C_i curves, provides parameters/graphs
Gas Exchange	`fit_aq_response`	Fits A–Q curves, provides parameters/graphs
Gas Exchange	`fit_g_mc_variableJ`	Fits g_{mc}, adds g_{mc} and $dC_c dA$ to data frame for reliability checking
Gas Exchange	`fit_gs_model`	Fits the Ball *et al.* (1987), Leuning (1995) and Medlyn *et al.* (2011) models of stomatal conductance, provides parameters/graphs
Hydraulics	`fit_hydra_vuln_curve`	Fits the sigmoidal and Weibull models to hydraulic vulnerability data, provides parameters/graphs
Hydraulics	`fit_PV_curve`	Fits pressure–volume curves, provides parameters/graphs
Gas Exchange	`fit_r_light`	Fits r_light according to the Kok (1956) method, Yin method (Yin *et al.* 2009, 2011) or Walker and Ort (2015) method.
Gas Exchange, Biochemistry	`fit_t_response`	Fits an Arrhenius (Arrhenius 1915), Heskel (Heskel *et al.* 2016), Kruse (Kruse *et al.* 2006), Medlyn (Medlyn *et al.* 2002), Macromolecular rate theory (Hobbs *et al.* 2013) and quadratic temperature response models, provides parameters/graphs
Modelling	`photo`	Simulates C_3 photosynthesis over a parameter set
Modelling	`make_parameters`	A set of functions (e.g. `make_enviropar`, `make_leafpar`) that generates the required inputs for photo
Meta-functions and utilities		
Application	Function	Description
Software modification	`check_dependencies`	Generates HTML with package and function dependencies
All components	`compile_data`	Compiles the output from the `fit_many` function
All components	`fit_many`	Fits a function many times through a grouping variable
All components	`print_graphs`	Prints graphs from a list of graphs
All components	`sensitivity_analysis`	Allows up to two-factor sensitivity analysis of any function

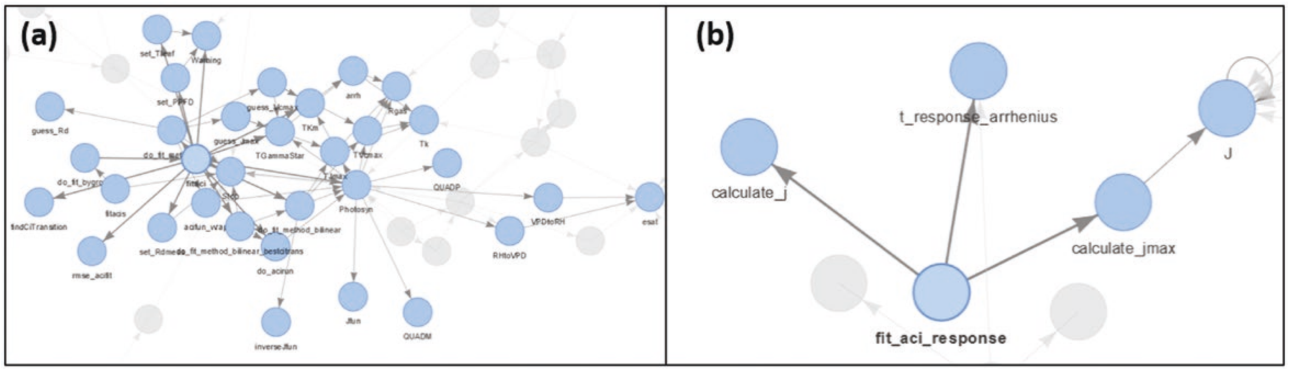


Figure 2. Dependencies of the A–C_i fitting functions in (A) {plantecophys} and (B) {photosynthesis}.

The first vignette (titled 'photosynthesis-curve-fitting-sensitivity-analyses') demonstrates how to use curve fitting and sensitivity tools and the second (titled 'introduction to the photosynthesis package') demonstrates how to simulate photosynthetic rate using the Farquhar–von Caemmerer–Berry C_3 biochemical model, define leaf and environmental parameters, replace default parameters and solve for chloroplastic CO_2 concentrations.

The package is specifically designed to accommodate new analytical tools and discoveries and be easily maintained by new users. Non-linear curve fitting procedures use the nlsLM function from {minpack.lm} (Elzhov *et al.* 2016), which provides a more robust fitting procedure for non-linear functions than the base R `nls` function. Graphical outputs are provided using {ggplot2} (Wickham 2016a). Meta-functions were constructed with the tools provided for generalizing functions and arguments in {rlang} (Henry and Wickham 2019).

The principles of modularity and functional programming have been used to substantially reduce code interdependencies within the software. For example, the `fitaci` function from {plantecophys} has over 30 function dependencies (Fig. 2A). By applying our principles, we were able to reduce this to just four function dependencies (Fig. 2B), by re-engineering the fitting procedure and eliminating redundant functions and code. Arguably, fewer dependencies could indicate less modularity, even though each of the components is modular, but fewer dependencies may reduce the number of bugs introduced by revisions in other components.

Example data set. To demonstrate the fitting functions of the package, we use a combination of data collected for the package and previously published data. A CO_2 by light response curve and CO_2 by temperature response curve were collected in sunflower (*Helianthus annuum*) grown in a rooftop greenhouse at the University of New Mexico (35.0843°N, 106.6198°W, 1587 m a.s.l., 18.3 to 21.1/15.6 to 21.1 °C day/night temperature with daily irradiances of 600 to 1200 μmol m^{-2} s^{-1}). CO_2 response curves were measured at irradiances of 1500, 150, 375, 125, 100, 75, 50 and 25 μmol m^{-2} s^{-1} at a T_{leaf} of 25 °C. CO_2 response curves were also measured at T_{leaf} of 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5 and

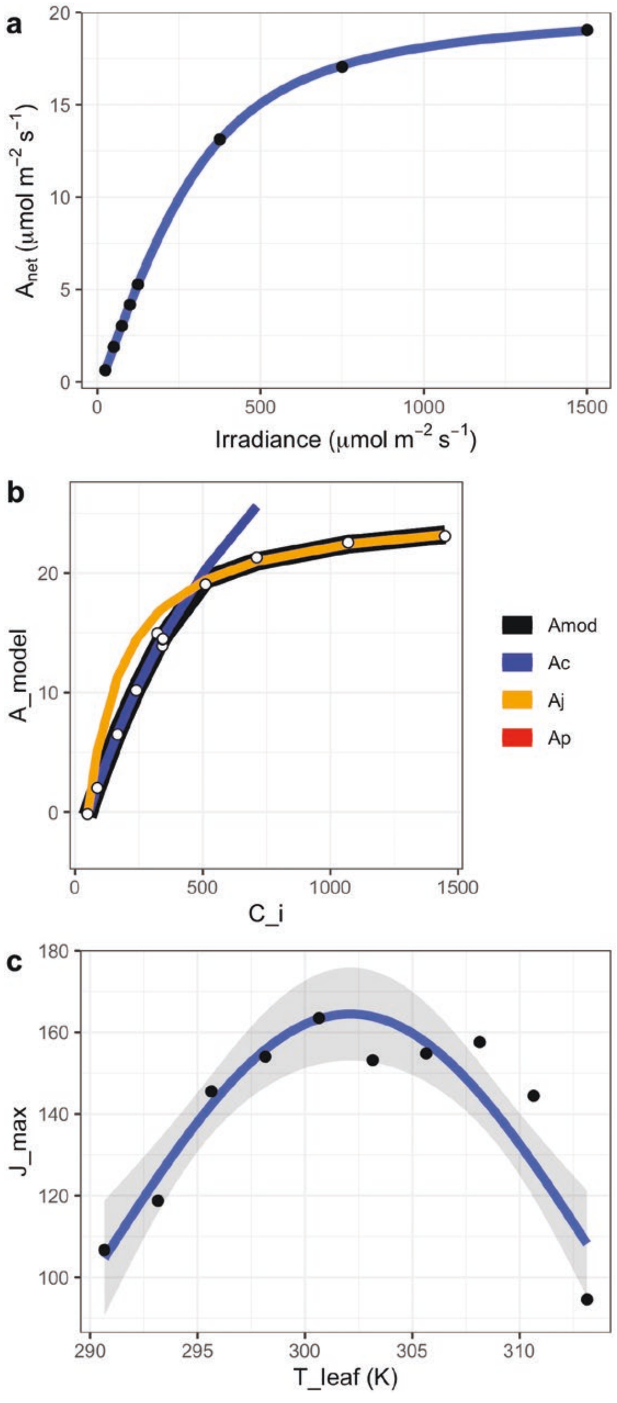


Figure 3. Gas exchange curve fitting outputs. (A) Output from `fit_aq_response` showing the data (black points), the model fit (blue line) and the standard error on the model fit (grey region). The light response at a $[CO_2]$ of 100 μmol mol^{-1} is shown. A_{net}: net CO_2 assimilation. (B) Graph from `fit_aci_response` showing modelled A_{net} (A_{mod}, black line), CO_2-limited A_{net} (A_c, blue), RuBP regeneration-limited A_{net} (A_j, orange), triose phosphate utilization-limited A_{net} (A_p) and the data (white dots). A_{net}: net CO_2 assimilation; C_i: intercellular CO_2 concentration. (C) Output from `fit_t_response` showing the Heskel temperature response of J_{max}. Data are black dots, model fit is the blue line and the grey shaded region is the standard error on the model fit. J_{max}: maximum rate of electron transport; T_{leaf}: leaf temperature..

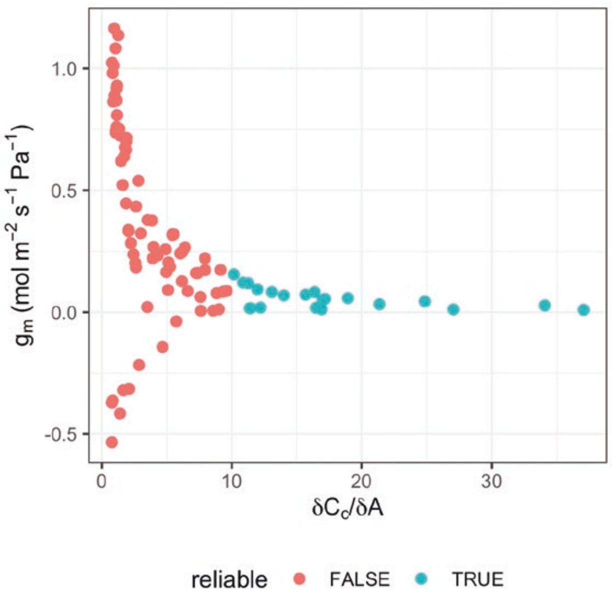


Figure 4. Relationship between g_{mc} estimated through the variable *J* method and $\delta C_c/\delta A$ to test for reliability. The `fit_g_mc_variableJ` function was used on the CO_2 by light response data in sunflower. g_m: mesophyll conductance; C_c: chloroplastic CO_2 concentration; A: net CO_2 assimilation.

40 °C at an irradiance of 1500 µmol m^{-2} s^{-1}. Data to demonstrate hydraulic vulnerability curve fitting methods were drawn from Hudson *et al.* (2018), while data for leaf pressure/volume analysis come from an unpublished data set collected at the University of New Mexico. Below we illustrate some of the functionality of the package. These data are freely available in the package, so potential users can test out the functions and different analyses in the code. We refer potential users to the package vignette for more worked examples (**see Supporting Information—Notes S2**, 'photosynthesis-curve-fitting-sensitivity-analyses.rmd').

Photosynthetic light response curve fitting. The `fit_aq_response` function returns a list containing the fitted light response model, model parameters and a graph showing the model fit to the data (Fig. 3A). This function estimates the light-saturated net CO$_2$ assimilation rate, quantum yield of CO$_2$ assimilation, an empirical curvature factor and respiration (Marshall and Biscoe 1980).

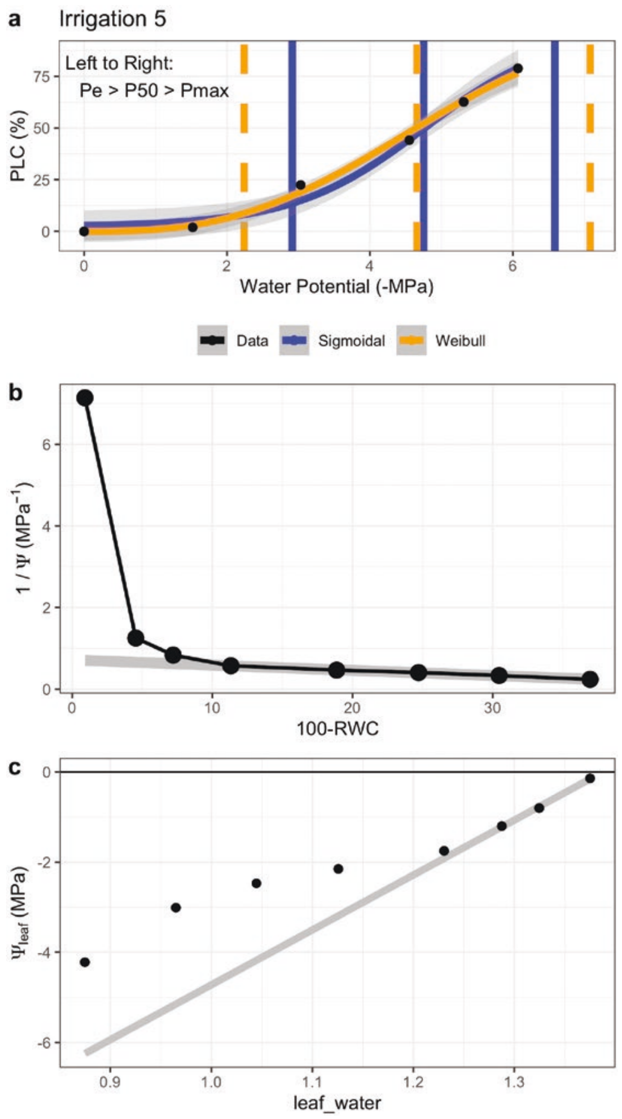


Figure 5. (A) Example output from fit_hydra_vuln_curve showing both model fits overlaid on the data (black dots). PLC: percent loss of conductivity; P$_e$: air entry point; P$_{50}$: water potential at 50 % PLC; P$_{max}$: hydraulic failure threshold. (B, C) Example output from `fit_pv_curve` showing the (B) water mass graph and (C) the pressure–volume curve. Grey lines are fit to the linear regions of the data. Ψ: water potential; RWC: relative water content.

Photosynthetic CO$_2$ response curve fitting. The `fit_aci_response` function returns a list containing the fitted parameters, a data frame with the modelled data output and a graph showing the model fit to the data (Fig. 3B). It estimates the standard parameters of the Farquhar–von Caemmerer–Berry C$_3$ biochemical model (Farquhar *et al.* 1980) and parameter standard errors to help evaluate results. As with any non-linear regression, failure of the solver to converge on a solution or very large standard errors usually indicates problems fitting the model to the data and unreliable parameter estimates.

Photosynthetic temperature response curve fitting. A series of temperature response functions can be fit using the package, with the outputs including the fitted model, model parameters and a graph (Fig. 3C). As with other functions, details about parameters are given in the package documentation.

Fitting g_m using the variable J method. The `fit_g_mc_variableJ` function implements the method of Harley *et al.* (1992) using chlorophyll fluorescence and gas exchange data to estimate g_{mc}. Both g_{mc} and $\delta C_c/\delta A$ are calculated, where $\delta C_c/\delta A$ between 10 and 50 are deemed to be 'reliable' (Harley *et al.* 1992), and an average g_{mc} value is estimated based on the reliable values. This makes it relatively easy to assess the reliability of g_{mc} estimates (Fig. 4).

Hydraulic vulnerability curve fitting. The `fit_hydra_vuln_curve` fits hydraulic vulnerability data using both a sigmoidal and Weibull function. Outputs include model fits, parameters and a graph (Fig. 5A).

Pressure–volume curves. The `fit_pv_curve` fits pressure–volume curves, returning parameters such as relative water content and water potential at turgor loss points, relative capacitance at full turgor and others. Outputs include parameters and graphs (Fig. 5B and C).

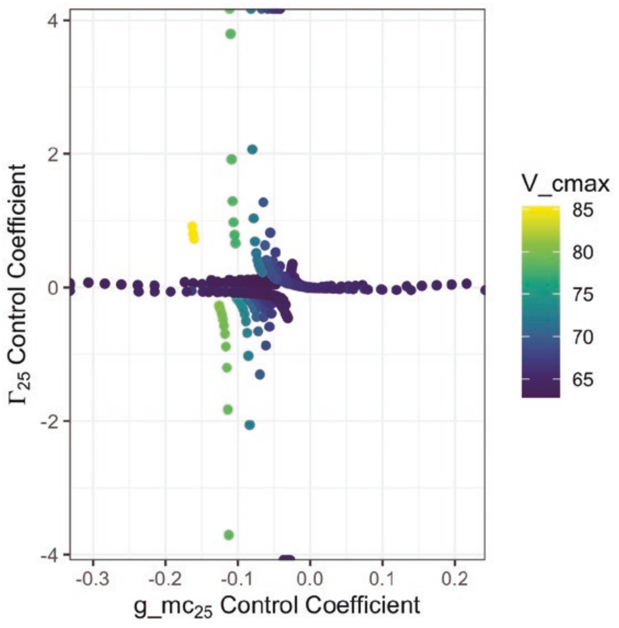


Figure 6. Control coefficients of g_m and Γ* at 25 °C calculated from `analyze_sensitivity` and `compute_sensitivity`.

Sensitivity analyses. Both `analyze_sensitivity` and `compute_sensitivity` are used in combination for sensitivity analyses. `analyze_sensitivity` allows up to two assumed parameters to be varied in a fitting function, while `compute_sensitivity` runs two types of local sensitivity calculations based on a user-defined reference value: parameter effect (Bauerle *et al.* 2014) and control coefficient (Capaldo and Pandis 1997). We can look at the impact of varying g_m and Γ^* at 25 °C on fitted V_{cmax} (Fig. 6). We can see that g_m and Γ^* at 25 °C have an orthogonal impact on V_{cmax}, with Γ^* having a stronger control than g_m on V_{cmax}.

Moving forward—standardized practices and code editors

It is not easy to rewrite software, and we are not arguing as such. Rather, going forward as a community, we argue that we should adopt a set of coding principles and guidelines to create code as flexible as the biology we study. We present the R package, {photosynthesis}, as an example of these principles and guidelines. The consequences of this are not to be understated: it will be easier for new trainees and beginner coders to learn, understand and write code for the community; and it will be easier to tailor existing code to our projects.

The drawback is that code may run more slowly, which may be a worthwhile trade-off for some but not others. For example, computational speed may take precedence over flexibility for eddy flux covariance, genomics and other ‘big data’ applications. In ecophysiology, many data sets are often small enough that even complex analyses may only take 1 h on one computer core of a multi-core system—as a community we can often afford slower-running code for greater flexibility and ease-of-understanding, especially as this could save days or weeks of coding to write a desired analysis. Our code should be as flexible as, and easier to understand, than the biology it describes.

However, providing code according to these standards is not sufficient—we also need code-competent editorial staff for journals who can properly review and test submitted code to ensure that it runs as intended. In some cases, code for a published data set does not work even after comprehensive modification (J. R. Stinziano, pers. comm.). Standardized coding practices will help to reduce the burden on code editors by making it easier to read and understand code submissions.

Supporting Information

The following additional information is available in the online version of this article—

Methods S1. Description of variables used in {photosynthesis}.

Table S1. Table of other utility functions in {photosynthesis}.

Notes S1. Benchmark comparison of `plantecophys::fitaci` and `photosynthesis::fit_aci_response`.

Notes S2. The {photosynthesis} R package tar.gz file.

Notes S3. Examples tidy data file (hydraulic_vulnerability.csv).

Conflict of Interest

None declared.

Funding

CDM was supported by grant 1929167 from the National Science Foundation.

Acknowledgements

We thank Tom Buckley, Markus Hauck and three anonymous reviewers for feedback on earlier versions of the manuscript. This is publication #145 from the School of Life Sciences, University of Hawai’i at Mānoa.

Data Availability

All data and code used in the manuscript are available at https://github.com/cdmuir/photosynthesis.

Literature Cited

Adrion JR, Cole CB, Dukler N, Galloway JG, Gladstein AL, Gower G, Kyriazis CC, Ragsdale AP, Tsambos G, Baumdicker F, Carlson J, RA Cartwright, A Durvasula, I Gronau, BY Kim, P McKenzie, Messer PW, Noskova E, Vecchyo DO-D, Racimo F, Struck TJ, Gravel S, Gutenkunst RN, Lohmueller KE, Ralph PL, Schrider DR, Siepel A, Kelleher J, Kern AD. 2020. A community-maintained standard library of population genetic models. *eLife* **9**:e54967.

Arrhenius S. 1915. *Quantitative laws in biological chemistry*. London: Bell.

Ball JT, Woodrow IE, Berry JA. 1987. A model predicting stomatal conductance and its contribution to the control of photosynthesis under different environmental conditions. In: Biggins I, ed. Progress in Photosynthesis Research, Proceedings of the VII International Congress on Photosynthesis, vol. **4**. Dordrecht, The Netherlands: Martinus Nijhoff, 221–224.

Bauerle WL, Daniels AB, Barnard DM. 2014. Carbon and water flux responses to physiology by environment interactions: a sensitivity analysis of variation in climate on photosynthetic and stomatal parameters. *Climate Dynamics* **42**:2539–2554.

Bellasio C, Beerling DJ, Griffiths H. 2016. An Excel tool for deriving key photosynthetic parameters from combined gas exchange and chlorophyll fluorescence: theory and practice. *Plant, Cell & Environment* **39**:1180–1197.

Bernacchi CJ, Portis AR, Nakano H, von Caemmerer S, Long SP. 2002. Temperature response of mesophyll conductance. Implications for the determination of Rubisco enzyme kinetics and for limitations to photosynthesis in vivo. *Plant Physiology* **130**:1992–1998.

Bernacchi CJ, Singsaas EL, Pimentel C, Portis AR, Long SP. 2001. Improved temperature response functions for models of Rubisco-limited photosynthesis. *Plant Cell & Environment* **24**:253–259.

Bezanson J, Edelman A, Karpinski S, Shah VB. 2017. Julia: a fresh approach to numerical computing. *SIAM Review* **59**(1): 65–98.

Burns B, Lamb J, Qi J. 2020. pkgnet: get network representation of an R package. R package version 0.4.1. https://CRAN.R-project.org/package=pkgnet (7 September 2021).

Capaldo KP, Pandis SN. 1997. Dimethylsulfide chemistry in the remote marine atmosphere: evaluation and sensitivity analysis of available mechanisms. *Journal of Geophysical Research* **102**:23251–23267.

Duursma RA. 2015. Plantecophys—an R package for analysing and modelling leaf gas exchange data. *PLoS One* **10**:e0143346.

Elzhov TV, Mull KM, Spiess A-N, Bolker B. 2016. minpack.lm: R interface to the Levenberg–Marquardt nonlinear least-squares algorithm found in MINPACK, plus support for bounds. R package version 1.2-1. https://CRAN.R-project.org/package=minpack.lm (7 September 2021).

Farquhar GD, von Caemmerer S, Berry JA. 1980. A biochemical model of photosynthetic CO_2 assimilation in leaves of C_3 species. *Planta* **149**:78–90.

Gu L, Pallardy SG, Tu K, Law BE, Wullschleger SD. 2010. Reliable estimation of biochemical parameters from C_3 leaf photosynthesis-intercellular carbon dioxide response curves. *Plant Cell & Environment* **33**:1852–1874.

Harley PC, Loreto F, Di Marco G, Sharkey TD. 1992. Theoretical considerations when estimating mesophyll conductance to CO_2 flux by analysis of the response of photosynthesis to CO_2. *Plant Physiology* **98**:1429–1436.

Henry L, Wickham H. 2019. Rlang: functions for base types and core R and ‘Tidyverse’ features. R package version 0.4.2. https://CRAN.R-project.org/package=rlang (7 September 2021).

Henry L, Wickham H. 2020. purrr: functional programming tools. R package version 0.3.4. https://CRAN.R-project.org/package=purrr (7 September 2021).

Heskel MA, O'Sullivan OS, Reich PB, Tjoelker MG, Weerasinghe LK, Penillard A, Egerton JJ, Creek D, Bloomfield KJ, Xiang J, Sinca F, Stangl ZR, Martinez-de la Torre A, Griffin KL, Huntingford C, Hurry V, Meir P, Turnbull MH, Atkin OK. 2016. Convergence in the temperature response of leaf respiration across biomes and plant functional types. *Proceedings of the National Academy of Sciences of the United States of America* **113**:3832–3837.

Hester J, Angly F, Hyde R. 2020. lintr: a 'Linter' for R code. R package version 2.0.1. https://CRAN.R-project.org/package=lintr (7 September 2021).

Hobbs JK, Jiao W, Easter AD, Parker EJ, Schipper LA, Arcus VL. 2013. Change in heat capacity for enzyme catalysis determines temperature dependence of enzyme catalyzed rates, *ACS Chemical Biology* **8**:2388–2393.

Hudson PJ, Limousin JM, Krofcheck DJ, Boutz AL, Pangle RE, Gehres N, McDowell NG, Pockman WT. 2018. Impacts of long-term precipitation manipulation on hydraulic architecture and xylem anatomy of piñon and juniper in Southwest USA. *Plant, Cell & Environment* **41**:421–435.

Johnson F, Eyring H, Williams R. 1942. The nature of enzyme inhibitions in bacterial luminescence: sulphanilamide, urethane, temperature, pressure. *Journal of Cell Comparative Physiology* **20**:247–268.

Koide RT, Robichaux RH, Morse SR, Smith CM. 2000. Plant water status, hydraulic resistance and capacitance. In: Pearcy RW, Ehleringer JR, Mooney HA, Rundel PW, eds. *Plant physiological ecology: field methods and instrumentation*. Dordrecht, The Netherlands: Kluwer, 161–183.

Kok B. 1956. On the inhibition of photosynthesis by intense light. *Biochimica et Biophysica Acta* **21**:234–244.

Kruse J, Rennenberg H, Adams MA. 2006. Steps towards a mechanistic understanding of respiratory temperature responses. *New Phytologist* **189**:659–677.

Kumarathunge DP, Medlyn BE, Drake JE, Tjoelker MG, Aspinwall MJ, Battaglia M, Cano FJ, Carter KR, Cavaleri MA, Cernusak LA, Chambers JQ, Crous KY, De Kauwe MG, Dillaway DN, Dreyer E, Ellsworth DS, Ghannoum O, Han Q, Hikosaka K, Jensen AM, Kelly JWG, EL Kruger, LM Mercado, Y Onoda, PB Reich, Rogers A, Slot M, Smith NG, Tarvainen L, Tissue DT, Togashi HF, Tribuzy ES, Uddling J, Vårhammar A, Wallin G, Warren JM, Way DA. 2019. Acclimation and adaptation components of the temperature dependence of plant photosynthesis at the global scale. *New Phytologist* **222**:768–784.

Laisk A. 1977. *Kinetics of photosynthesis and photorespiration in C_3 plants*. Moscow: Nauka.

Leuning R. 1995. A critical appraisal of a coupled stomatal-photosynthesis model for C_3 plants. *Plant Cell & Environment* **18**:339–357.

Liang LL, Arcus VL, Heskel MA, O'Sullivan OS, Weerasinghe LK, Creek D, Egerton JJG, Tjoelker MG, Atkin OK, Schipper LA. 2018. Macromolecular rate theory (MMRT) provides a thermodynamics rationale to underpin the convergent temperature response in plant leaf respiration. *Global Change Biology* **24**:1538–1547.

Mangul S, Martin LS, Eskin E, Blekhman R. 2019. Improving the usability and archival stability of bioinformatics software. *Genome Biology* **20**:47.

Marshall B, Biscoe P. 1980. A model for C_3 leaves describing the dependence of net photosynthesis on irradiance. *Journal of Experimental Botany* **31**:29–39.

Medlyn BE, Dreyer E, Ellsworth D, Forstreuter M, Harley PC, Kirschbaum MUF, Le Roux X, Montpied P, Strassemeyer J, Walcroft A, Wang K, Loutstau D. 2002. Temperature response of parameters of a biochemically based model of photosynthesis. II. A review of experimental data. *Plant Cell & Environment* **25**:1167–1179.

Medlyn BE, Duursma RA, Eamus D, Ellsworth DS, Prentice IC, Barton CVM, Crous KY, Angelis PD, Freeman M, Wingate L. 2011. Reconciling the optimal and empirical approaches to modelling stomatal conductance. *Global Change Biology* **17**:2134–2144.

Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. 2016. SymPy: symbolic computing in Python. *PeerJ Computer Science* **3**:e103.

Muir CD. 2019. tealeaves: an R package for modelling leaf temperature using energy budgets. *AoB Plants* **11**:plz054; doi:10.1093/aobpla/plz054 (7 September 2021).

Müller K, Walthert L. 2020. styler: non-invasive pretty printing of R code. R package version 1.3.2. https://CRAN.R-project.org/package=styler (7 September 2021).

Ogle K, Barber JJ, Willson C, Thompson B. 2009. Hierarchical statistical modeling of xylem vulnerability to cavitation. *The New Phytologist* **182**:541–554.

Pammenter NW, van der Willigen C. 1998. A mathematical and statistical analysis of the curves illustrating vulnerability of xylem to cavitation. *Tree Physiology* **18**:589–593.

Pebesma R, Mailund T, Hiebert J. 2016. Measurement units in R. *R Journal* **8**:486–494.

Prlić A, Proctor PB. 2012. Ten simple rules for the open development of scientific software. *PLoS Computational Biology* **8**:e1002802.

R Core Team. 2021. R: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/ (7 September 2021).

Reid DE, Silins U, Mendoza C, Lieffers VJ. 2005. A unified nomenclature for quantification and description of water conducting properties of sapwood xylem based on Darcy's law. *Tree Physiology* **25**:993–1000.

Sack L, Cowan PD, Jaikumar N, Holbrook NM. 2003. The 'hydrology' of leaves: co-ordination of structure and function in temperate woody species. *Plant, Cell & Environment* **26**:1343–1356.

Sayres MAW, Hauser C, Sierk M, Robic S, Rosenwald AG, Smith TM, Triplett EW, Williams JJ, Dinsdale E, Morgan WR, Burnette III JM, Donovan SS, Drew JC, Elgin SCR, Fowlks ER, Galindo-Gonzalez S, Goodman AL, Grandgenett NF, Goller CC, Jungck JR, Newman JD, Pearson W, Ryder EF, Tosado-Acevedo R, Tapprich W, Tobin TC, Toro-Martínez A, Welch LR, Wright R, Barone L, Ebenbach D, McWilliams M, Olney KC, Pauley MA. 2018. Bioinformatics core competencies for undergraduate life sciences education. *PLoS One* **13**:e0196878.

Sharkey TD, Bernacchi CJ, Farquhar GD, Singaas EL. 2007. Fitting photosynthetic carbon dioxide response curves for C3 leaves. *Plant, Cell and Environment* **30**:1035–1040.

Stinziano JR, C Roback, D Sargent, B Murphy, P Hudson, CD Muir. 2020. photosynthesis: tools for plant ecophysiology & modeling. https://CRAN.R-project.org/package=photosynthesis (7 September 2021).

Tyree MT, Hammel HT. 1972. Measurement of turgor pressure and water relations of plants by pressure bomb technique. *Journal of Experimental Botany* **23**:267.

Vaughan D, Dancho M. 2018. furrr: apply mapping functions in parallel using futures. R package version 0.1.0. https://CRAN.R-project.org/package=furrr (7 September 2021).

von Caemmerer S. 2000. *Biochemical models of leaf photosynthesis*. Collingwood: CSIRO Publishing.

Walker AP, Johnson AL, Rogers A, Anderson J, Bridges RA, Fisher RA, Lu D, Ricciuto DM, Serbin SP, Ye M. 2021. Multi-hypothesis comparison of Farquhar and Collatz photosynthesis models reveals the unexpected influence of empirical assumptions at leaf and global scales. *Global Change Biology* **27**:804–822.

Walker BJ, Ort DR. 2015. Improved method for measuring the apparent CO_2 photocompensation point resolves the impact of multiple internal conductances to CO_2 to net gas exchange. *Plant, Cell & Environment* **38**:2462–2474.

Wickham H. 2011. testthat: get started with testing. *The R Journal* **3**:5–10.

Wickham H. 2014. Tidy data. *Journal of Statistical Software* **59**:1–23.

Wickham H. 2015. *R packages: organize, test, document, and share your code*. Sebastopol, CA: O'Reilly Media Inc.

Wickham H. 2016a. *ggplot2: elegant graphics for data analysis*. New York: Springer-Verlag.

Wickham H. 2016b. Tidyverse: easily install and load 'tidyverse' packages. https://CRAN.R-project.org/package=tidyverse (7 September 2021).

Wickham H. 2017. Tidyverse Tidyweb. http://tidyverse.org/.project.org/package=tidyverse (7 September 2021).

Wickham H. 2019. *Advanced R*, 2nd edn. Boca Raton: Chapman & Hall.

Wickham H, François R, Henry L, Müller K. 2020. dplyr: a grammar of data manipulation. R package version 1.0.0. https://CRAN.R-project.org/package=dplyr (7 September 2021).

Wickham H, Grolemund G. 2016. *R for data science*. Sebastopol, CA: O'Reilly Media Inc.

Wickham H, Henry L. 2020. tidyr: tidy messy data. R package version 1.1.0. https://CRAN.R-project.org/package=tidyr.

Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, Haddock SH, Huff KD, Mitchell IM, Plumbley MD, Waugh B, White EP, Wilson P. 2014. Best practices for scientific computing. *PLoS Biology* **12**:e1001745.

Xie Y. 2019. formatR: format R code automatically. R package version 1.7. https://CRAN.R-project.org/package=formatR (7 September 2021).

Yin X, Struik PC, Romero P, Harbinson J, Evers JB, van der Putten PEL, Vos J. 2009. Using combined measurements of gas exchange and chlorophyll fluorescence to estimate parameters of a biochemical C_3 photosynthesis model: a critical appraisal and a new integrated approach applied to leaves in a wheat (*Triticum aestivum*) canopy. *Plant Cell & Environment* **32**:448–464.

Yin X, Sun Z, Struik PC, Gu J. 2011. Evaluating a new method to estimate the rate of leaf respiration in the light by analysis of combined gas exchange and chlorophyll fluorescence measurements. *Journal of Experimental Botany* **62**:3489–3499.