Tutorial on bigleaf

Juergen Knauer

2019-06-08

This vignette is a short introduction to the functionalities of the `bigleaf` R package (version 0.7.0). It is directed to first-time package users who are familiar with the basic concepts of R. After presenting the use of several key functions of the package, some useful hints and guidelines are given at the end of the vignette.

Package scope and important conceptual considerations

`bigleaf` calculates physical and physiological ecosystem properties from eddy covariance data. Examples for such properties are aerodynamic and surface conductance, surface conditions(e.g. temperature, VPD), wind profile, roughness parameters, vegetation-atmosphere decoupling, potential evapotranspiration, (intrinsic) water-use efficiency, stomatal sensitivity to VPD, or intercellular CO2 concentration. All calculations in the `bigleaf` package assume that the ecosystem behaves like a "big-leaf", i.e. a single, homogenous plane which acts as the only source and sink of the measured fluxes. This assumption comes with the advantages that calculations are simplified considerably and that (in most cases) little ancillary information on the EC sites is required. It is important to keep in mind that these simplifications go hand in hand with critical limitations. All derived variables are bulk ecosystem characteristics and have to be interpreted as such. It is for example not possible to infer within-canopy variations of a certain property.

Please also keep in mind that the `bigleaf` package does NOT provide formulations for bottom-up modelling. The principle applied here is to use an inversion approach in which ecosystem properties are inferred top-down from the measured fluxes. Such an inversion can, in principle, be also be conducted with more complex models (e.g. sun/shade or canopy/soil models), but keep in mind that these approaches also require that the additional, site-specific parameters are adequately well known.

The use of more detailed models is not within the scope of the `bigleaf` R package, but it is preferable to use such approaches when important assumptions of the "big-leaf" approach are not met. This is the case in particular when the ecosystem is sparsely covered with vegetation (low LAI, e.g. sparse crops, some savanna systems).

Installation and Loading

The `bigleaf` R package is on CRAN and can be installed with the usual command:

```
install.packages("bigleaf")
```

Alternatively, the development version can be installed from the online repository using the `devtools` package. The development version includes the most recent changes to the package that have not yet been submitted to CRAN:

```
library(devtools)
install_bitbucket("juergenknauer/bigleaf")
```

Note that for the `install_bitbucket` command to work on Windows, Rtools must be installed. After installation, the package can be loaded:

```
library(bigleaf)
```

Preparing the data

In this tutorial, we will work with a dataset from the eddy covariance site Tharandt (DE-Tha), a spruce forest in Eastern Germany. The data.frame `DE_Tha_Jun_2014` is automatically loaded when the `bigleaf` package is loaded and contains half-hourly data of meteorological and flux measurements made in June 2014:

```
head(DE_Tha_Jun_2014)
#>   year month doy hour  Tair Tair_qc PPFD PPFD_qc    VPD VPD_qc pressure
#> 1 2014     6 152  0.0 11.88       0    0       0 0.5746      0    97.64
#> 2 2014     6 152  0.5 11.67       0    0       0 0.5634      0    97.63
#> 3 2014     6 152  1.0 11.19       0    0       0 0.5137      0    97.61
#> 4 2014     6 152  1.5 10.80       0    0       0 0.4561      0    97.61
#> 5 2014     6 152  2.0 10.67       0    0       0 0.4184      0    97.61
#> 6 2014     6 152  2.5 10.13       0    0       0 0.3609      0    97.61
#>   precip precip_qc ustar wind wind_qc     Ca Ca_qc  LW_up LW_down     Rn
#> 1      0         0  0.54 4.21       0 402.19     0 369.43  282.93 -86.49
#> 2      0         0  0.49 4.46       0 403.83     0 368.67  284.46 -84.20
#> 3      0         0  0.48 4.54       0 406.00     0 366.48  284.67 -81.81
#> 4      0         0  0.45 4.08       0 407.71     0 364.57  286.68 -77.90
#> 5      0         0  0.51 3.95       0 407.69     0 363.74  284.65 -79.09
#> 6      0         0  0.46 4.02       0 410.11     0 361.47  282.38 -79.09
#>      LE LE_qc      H H_qc      G G_qc  NEE NEE_qc      GPP GPP_qc    Reco
#> 1  9.94     0 -68.18    0 -4.935    0 9.94      0 -4.02527      0 5.91473
#> 2  5.27     0 -48.54    0 -5.085    0 7.59      0 -1.72228      0 5.86772
#> 3  3.98     0 -59.10    0 -5.135    0 9.51      0 -3.74479      0 5.76521
#> 4 -6.72     0 -60.11    0 -5.210    0 8.89      0 -3.20864      0 5.68137
#> 5 -3.08     0 -59.40    0 -5.280    0 9.03      0 -3.37897      0 5.65103
#> 6 -2.16     0 -48.92    0 -5.345    0 9.12      0 -3.58337      0 5.53664
tha <- DE_Tha_Jun_2014
```

We give the data.frame a shorter name here. More information on the data (e.g. meaning of column names and units) can be found when typing `?DE_Tha_Jun_2014`. For more information on the site see e.g. Grünwald & Bernhofer 2007. In addition, we will need some ancillary data for this site throughout this tutorial. To ensure consistency, we define them here at the beginning:

```
LAI <- 7.6   # leaf area index
zh  <- 26.5  # average vegetation height (m)
zr  <- 42    # sensor height (m)
Dl  <- 0.01  # leaf characteristic dimension (m)
```

General guidelines on package usage

There are a few general guidelines that are important to consider when using the `bigleaf` package.

Units

It is imperative that variables are provided in the right units, as the plausibility of the input units is not checked in most cases. The required units of the input arguments can be found in the respective help file of the function. The good news is that units do not change across functions. For example, pressure is always required in kPa, and temperature always in °c.

Function arguments

Most functions of the `bigleaf` package require an input matrix or (more common) a data.frame, from which the required variables are extracted. This is usually the first argument of a function. Most functions further provide default values for their arguments, such that in many cases it is not necessary to provide them explicitly.

Another convenient feature of the `bigleaf` package is that arguments can be provided as both character and numeric vectors. If the argument is of type character, it is interpreted as the column name of the input data.frame. If it is of type numeric, it is used directly for the calculations.

We can demonstrate the usage with a simple example:

```
potential.ET(tha,Tair="Tair",pressure="pressure",Rn="Rn",VPD="VPD",approach="Priestley-Taylor")
potential.ET(tha)
potential.ET(tha,Tair=tha$Tair)
potential.ET(tha,Tair=25)
potential.ET(Tair=25,pressure=100,Rn=200)
```

In the first line above, the input arguments are provided as the names of the data.frame. In this case we do not need to provide them explicitly because the column names correspond to the default names of the function (i.e. the command can be written as in line 2). In the third example, we replace one variable name with a numeric vector. In the fourth row, we calculate PET for a constant temperature of 25°C, but take all other variables from the data.frame. For some applications, in particular for exploratory or sensitivity analyses, application nr. 5 can be useful. In this case, we did not provide a data.frame, but only numeric vectors of length one (or of any other length). This can be useful to see e.g. how sensitive the results of a given functions are with respect to one of the input variables. We could, for instance, investigate how potential ET as calculated with the Priestley-Taylor formulation changes when R_n increases from 200 W m^{-2} to 400 W m^{-2} when all other variables are held constant:

```
potential.ET(Tair=25,pressure=100,Rn=200)
#> Ground heat flux G is not provided and set to 0.
#> Energy storage fluxes S are not provided and set to 0.
#>         ET_pot   LE_pot
#> 1 7.637154e-05 186.4802
potential.ET(Tair=25,pressure=100,Rn=400)
#> Ground heat flux G is not provided and set to 0.
#> Energy storage fluxes S are not provided and set to 0.
#>         ET_pot   LE_pot
#> 1 0.0001527431 372.9604
```

When using your own data, it is not mandatory to use exactly the same variable names as here, but working with `bigleaf` is easier if you do so because then the variable names do not have to be specified when calling a function.

Ground heat flux and storage fluxes

Many functions require the available energy (A), which is defined as ($A = R_n - G - S$, all in W m^{-2}), where R_n is the net radiation, G is the ground heat flux, and S is the sum of all storage fluxes of the ecosystem (see e.g. Leuning et al. 2012 for an overview). For some sites, G is not available, and for most sites, only a few components of S are measured. In `bigleaf` it is not a problem if G and/or S are missing (other than the results might be (slightly) biased), but special options exist for the treatment of missing S and G values. If the options `missing.G.as.NA = TRUE` or `missing.S.as.NA = TRUE`, then the output variable is not calculated for that time period. Otherwise missing S and G values are set to 0 automatically. Please note that the default is to ignore S and G values. If G and/or S are available, they always have to be added explicitly to the function call (by providing the column name of G/S or a vector).

Function walkthrough

In the following, we explain how to use several of the package's key functions. Further information on the functions can be found on the respective function help pages and the references therein.

Data filtering

For most applications it is meaningful to filter your data. There are two main reasons why we want to filter our data before we start calculating ecosystem properties. The first one is to exclude datapoints that do not fulfill the requirements of the EC technique or that are of bad quality due to e.g. instrument failure or gap-filling with poor confidence. Note that the quality assessment of the EC data is not the purpose of the `bigleaf` package. This is done by other packages (e.g. `REddyProc`), which often provide quality control flags for the variables. These quality control flags are used here to filter out bad-quality datapoints.

A second reason for filtering our data is that some derived properties are only meaningful if certain meteorological conditions are met. For instance, if we are interested in properties related to plant gas exchange, it makes most sense to focus on time periods when plants are photosynthetically active (i.e. in the growing season and at daytime).

The bigleaf package provides the function `filter.data` that filters the data according to the criteria described above. We start with an example where the data.frame is filtered only with respect to data quality (`quality.control=TRUE`):

```
tha_filtered1 <- filter.data(tha,quality.control=TRUE,vars.qc=c("LE","H","NEE","Tair","VPD","wind"),
                             quality.ext="_qc",good.quality = c(0,1),missing.qc.as.bad=TRUE)
#> Quality control:
#> LE: 0 data points (0%) set to NA
#> H: 2 data points (0.14%) set to NA
#> NEE: 7 data points (0.49%) set to NA
#> Tair: 0 data points (0%) set to NA
#> VPD: 0 data points (0%) set to NA
#> wind: 3 data points (0.21%) set to NA
```

In the function call above, `vars.qc` lists the variables that should be filtered with respect to their quality. This is usually a vector of type character that contains the column names of the variables that are to be filtered. `quality.ext` denotes the extension of the variable name that identifies the column as a quality control indicator of a given variable. The variables "LE" and "LE_qc", for example, denote the variable itself (latent heat flux), and the quality of the variable "LE", respectively. The argument `good.quality` specifies the values that the quality control indicator has to take in order to be considered as acceptable quality (i.e. to not be filtered). For example, if `good.quality=c(0,1)`, then all "LE" values whose "LE_qc" variable is larger than 1 are set to `NA`. The variable `missing.qc.as.bad` is required to decide what to do in case of missing values in the quality control variable. By default this is (conservatively) set to `TRUE`, i.e. all entries where the qc variable is missing is filtered out. The function prints some information on the amount of data filtered out. In this case, only a few values did not fulfill the quality criteria. In the next example, we filter for meteorological conditions only, including growing season (`filter.growseas=TRUE`):

```
tha_filtered2 <- filter.data(tha,quality.control=FALSE,filter.growseas=TRUE,
                             filter.vars=c("PPFD","ustar","LE","VPD"),
                             filter.vals.min=c(200,0.2,0,0.01), filter.vals.max=c(NA,NA,NA,NA),
                             NA.as.invalid = TRUE,
                             # arguments for growing season filter:
                             GPP="GPP",doy="doy",year="year",tGPP=0.4,ws=15,min.int=5)
#> Data filtering:
#> 0 data points (0%) excluded by growing season filter
#> 0 additional data points (0%) excluded by precipitation filter (0
#>  data points = 0 % in total)
#> 697 additional data points (48.4%) excluded by PPFD filter (697
```

```
#>  data points = 48.4 % in total)
#> 38 additional data points (2.64%) excluded by ustar filter (176
#>  data points = 12.22 % in total)
#> 75 additional data points (5.21%) excluded by LE filter (339 data points =
#> 23.54 % in total)
#> 0 additional data points (0%) excluded by VPD filter (0 data points = 0
#>  % in total)
#> 810 data points (56.25%) excluded in total
#> 630 valid data points (43.75%) remaining.
```

The arguments `filter.vars`, `filter.vals.min`, and `filter.vals.max` control the variables to be filtered (corresponding to the column names of the data.frame), the minimum and the maximum acceptable values, respectively. If there is no minimum or maximum, the respective entry can be set to `NA`. In this case we filter for time periods in which PPFD (photosynthetic photon flux density) has to be higher than 200 μmol m^{-2} s^{-1}, but no maximum limit is considered.

If `filter.growseas=TRUE`, the function implements a simple growing season filter based on daily smoothed GPP time series. The arguments `GPP`, `doy` and `year` are required at halfhourly/hourly time scale. GPP is aggregated to daily sums internally. The arguments `tGPP`, `ws`, and `min.int` determine how the growing season is filtered. `tGPP` determines how high daily GPP has to be in relation to its peak value within the year. In this case, the value of 0.4 denotes that smoothed GPP has to be at least 40% of the 95th quantile. `ws` controls the degree of smoothing in the timeseries (the purpose of which is to minimize the high variation of GPP between days), and should probably be between 10-20 days. `min.int` is a parameter that avoids that data are switching from inside the growing season and out from one day to the next, but determines the minimum number of days that the growing season should have. The growing season filter is applicable to all sites, with one more more growing seasons, but it's advisable that other parameter settings are used depending on the site.

In this case, it does not really make sense to filter for growing season, since it's only one month of which we know that vegetation is active at the site. Luckily, the algorithm realizes that as well and does not filter out any data if `filter.growseas=TRUE` (same will happen at sites with a year-round growing season). In the function output we further see that almost half of the data were filtered because radiation was not high enough (night-time). Another 23.5% were filtered because they showed negative LE values. However, most of them occur during the night, and only 5.2% of them were not already filtered by the radiation filter (denoted as "additional data points" above).

As a last step we will filter for precipitation events. This is often meaningful for ecophysiological studies because data during and shortly after rainfall events do not contain much information on the physiological activity of the vegetation (i.e. they comprise significant fractions of evaporation from the soil and plant surfaces). The purpose of such a filter is mostly to minimize the fraction of soil and interception evaporation on the total water flux. This filter simply excludes periods following a precipitation event. A precipitation event is here defined as any time step with a recorded precipitation higher than `tprecip` (in mm per timestep). The function then filters all time periods following a precipitation event. The number of subsequent time periods excluded is controlled by the argument `precip.hours`. Here, we exclude rainfall events and the following 24 hours.

```
tha_filtered3 <- filter.data(tha,quality.control=FALSE,filter.growseas=FALSE,
                             filter.precip=TRUE,precip="precip",tprecip=0.02,
                             records.per.hour=2,precip.hours=24)
#> Data filtering:
#> 0 data points (0%) excluded by growing season filter
#> 556 additional data points (38.61%) excluded by precipitation filter (556
#>  data points = 38.61 % in total)
#> 556 data points (38.61%) excluded in total
#> 884 valid data points (61.39%) remaining.
```

We can also do all the steps described above with a single function call, which is also the intention of the function:

```
tha_filtered <- filter.data(tha,quality.control=TRUE,filter.growseas=TRUE,
                            filter.precip=TRUE, filter.vars=c("PPFD","ustar","LE","VPD"),
                            filter.vals.min=c(200,0.2,0,0.01),filter.vals.max=c(NA,NA,NA,NA),
                            NA.as.invalid = TRUE,vars.qc=c("GPP","LE","H","NEE","Tair","VPD","wind"),
                            quality.ext="_qc",good.quality = c(0,1),missing.qc.as.bad=TRUE,
                            GPP="GPP",doy="doy",year="year",tGPP=0.4,ws=15,min.int=5,
                            precip="precip",tprecip=0.02,records.per.hour=2,precip.hours=24)
#> Quality control:
#> GPP: 7 data points (0.49%) set to NA
#> LE: 0 data points (0%) set to NA
#> H: 2 data points (0.14%) set to NA
#> NEE: 7 data points (0.49%) set to NA
#> Tair: 0 data points (0%) set to NA
#> VPD: 0 data points (0%) set to NA
#> wind: 3 data points (0.21%) set to NA
#> ----------------------------------------------------------------
#> Data filtering:
#> 0 data points (0%) excluded by growing season filter
#> 556 additional data points (38.61%) excluded by precipitation filter (556
#>  data points = 38.61 % in total)
#> 403 additional data points (27.99%) excluded by PPFD filter (697
#>  data points = 48.4 % in total)
#> 29 additional data points (2.01%) excluded by ustar filter (176
#>  data points = 12.22 % in total)
#> 25 additional data points (1.74%) excluded by LE filter (339 data points =
#> 23.54 % in total)
#> 0 additional data points (0%) excluded by VPD filter (0 data points = 0
#>  % in total)
#> 1013 data points (70.35%) excluded in total
#> 427 valid data points (29.65%) remaining.
```

When looking at the function output we see that we these settings, we exclude in total 1013 data points (70.35% of the data). In total, 29.65% of all data remained. The output of the `filter.data` function is another data.frame (tha_filtered), in which all filtered timesteps are set to NA. (Note that this is the default case. If we add `filtered.data.to.NA=TRUE`, the data are left untouched, but an additional column "valid" is added to the data.frame that specifies whether the time points fulfull the criteria or not). In the following examples we will work mostly with the filtered data.frame `tha_filtered`.

Aerodynamic conductance

An important metric for many calculations in the `bigleaf` package is the aerodynamic conductance (G_a) between the land surface and the measurement height. G_a characterizes how efficiently mass and energy is transferred between the land surface and the atmosphere. G_a consists of two parts: G_{am}, the aerodynamic conductance for momentum, and G_b, the canopy boundary layer (or quasi-laminar) conductance. G_a can be defined as $G_a = 1/(1/G_{am} + 1/G_b)$. In this tutorial we will focus on how to use the function `aerodynamic.conductance`. For further details on the equations, the reader is directed to the publication of the bigleaf package (Knauer et al. 2018) and the references therein. A good overview is provided by e.g. Verma 1989.

G_a and in particular G_b can be calculated with varying degrees of complexity. We start with the simplest version, in which G_b is calculated empirically based on the friction velocity (u_*) according to Thom 1972:

```
summary(aerodynamic.conductance(tha_filtered))
#>       Ga_m              Ra_m              Ga_h              Ra_h
#>  Min.   :0.0206   Min.   : 2.031   Min.   :0.0150   Min.   : 9.559
#>  1st Qu.:0.0927   1st Qu.: 5.606   1st Qu.:0.0469   1st Qu.:13.562
#>  Median :0.1277   Median : 7.832   Median :0.0593   Median :16.875
#>  Mean   :0.1368   Mean   : 9.307   Mean   :0.0598   Mean   :18.757
#>  3rd Qu.:0.1784   3rd Qu.:10.783   3rd Qu.:0.0737   3rd Qu.:21.328
#>  Max.   :0.4923   Max.   :48.500   Max.   :0.1046   Max.   :66.639
#>  NA's   :1013     NA's   :1013     NA's   :1013     NA's   :1013
#>       Gb_h              Rb_h              kB_h             zeta
#>  Min.   :0.0551   Min.   : 5.854   Min.   :1.487   Min.   : NA
#>  1st Qu.:0.0947   1st Qu.: 7.865   1st Qu.:1.948   1st Qu.: NA
#>  Median :0.1109   Median : 9.020   Median :2.108   Median : NA
#>  Mean   :0.1110   Mean   : 9.450   Mean   :2.097   Mean   :NaN
#>  3rd Qu.:0.1271   3rd Qu.:10.561   3rd Qu.:2.257   3rd Qu.: NA
#>  Max.   :0.1708   Max.   :18.139   Max.   :2.616   Max.   : NA
#>  NA's   :1013     NA's   :1013     NA's   :1013    NA's   :1440
#>      psi_h           Ra_CO2           Ga_CO2           Gb_CO2
#>  Min.   : NA    Min.   :11.42   Min.   :0.0138   Min.   :0.0419
#>  1st Qu.: NA    1st Qu.:16.07   1st Qu.:0.0408   1st Qu.:0.0719
#>  Median : NA    Median :19.89   Median :0.0503   Median :0.0842
#>  Mean   :NaN    Mean   :21.75   Mean   :0.0510   Mean   :0.0843
#>  3rd Qu.: NA    3rd Qu.:24.54   3rd Qu.:0.0622   3rd Qu.:0.0966
#>  Max.   : NA    Max.   :72.38   Max.   :0.0876   Max.   :0.1298
#>  NA's   :1440   NA's   :1013    NA's   :1013     NA's   :1013
```

Note that by not providing additional arguments, the default values are taken (type ?aerodynamic.conductance to see default values of the function arguments). We also do not need most of the arguments that can be provided to the function in this case (i.e. if `Rb_model="Thom_1972"`). These are only required if we use a more complex formulation of G_b. The output of the function is another data.frame which contains separate columns for conductances and resistances of different scalars (momentum, heat, and CO_2 by default). For comparison, we now calculate a second estimate of G_a, where the calculation of G_b is more physically-based (Su et al. 2001), and which requires more input variables compared to the first version. In particular, we now need LAI, the leaf characteristic dimension (D_l, assumed to be 1cm here), and information on sensor and canopy height (z_r and z_h), as well as the displacement height (assumed to be $0.7*z_h$):

```
Ga_Su <- aerodynamic.conductance(tha_filtered,Rb_model="Su_2001",LAI=LAI,zh=zh,d=0.7*zh,
                                 zr=zr,Dl=Dl)
summary(Ga_Su)
#>       Ga_m              Ra_m              Ga_h              Ra_h
#>  Min.   :0.0206   Min.   : 2.031   Min.   :0.0206   Min.   : 3.148
#>  1st Qu.:0.0927   1st Qu.: 5.606   1st Qu.:0.0723   1st Qu.: 8.801
#>  Median :0.1277   Median : 7.832   Median :0.0932   Median :10.730
#>  Mean   :0.1368   Mean   : 9.307   Mean   :0.0933   Mean   :12.420
#>  3rd Qu.:0.1784   3rd Qu.:10.783   3rd Qu.:0.1136   3rd Qu.:13.835
#>  Max.   :0.4923   Max.   :48.500   Max.   :0.3177   Max.   :48.540
#>  NA's   :1013     NA's   :1013     NA's   :1013     NA's   :1013
#>       Gb_h              Rb_h              kB_h             zeta
#>  Min.   : 0.1112   Min.   :0.0308   Min.   :0.0033   Min.   : NA
#>  1st Qu.: 0.2717   1st Qu.:2.5005   1st Qu.:0.5440   1st Qu.: NA
#>  Median : 0.3258   Median :3.0694   Median :0.7823   Median : NA
#>  Mean   : 0.7712   Mean   :3.1132   Mean   :0.7522   Mean   :NaN
#>  3rd Qu.: 0.3999   3rd Qu.:3.6803   3rd Qu.:1.0017   3rd Qu.: NA
#>  Max.   :32.4622   Max.   :8.9950   Max.   :1.5551   Max.   : NA
```

```
#>  NA's   :1013      NA's   :1013     NA's   :1013     NA's   :1440
#>      psi_h           Ra_CO2           Ga_CO2           Gb_CO2
#>  Min.   : NA    Min.   : 3.189   Min.   :0.0206   Min.   : 0.0845
#>  1st Qu.: NA    1st Qu.: 9.655   1st Qu.:0.0669   1st Qu.: 0.2064
#>  Median : NA    Median :11.676   Median :0.0856   Median : 0.2475
#>  Mean   :NaN    Mean   :13.404   Mean   :0.0853   Mean   : 0.5859
#>  3rd Qu.: NA    3rd Qu.:14.943   3rd Qu.:0.1036   3rd Qu.: 0.3038
#>  Max.   : NA    Max.   :48.553   Max.   :0.3136   Max.   :24.6623
#>  NA's   :1440   NA's   :1013     NA's   :1013     NA's   :1013
tha_filtered <- cbind(tha_filtered,Ga_Su)
```

We add the output of this function (`Ga_Su`) to our dataframe `tha_filtered`. We see that the values are different compared to the first, empirical estimate. This is because this formulation takes additional aerodynamically relevant properties (LAI, D_l) into account that were not considered by the simple empirical formulation.

Surface conditions

When we have an estimate of G_a, we are able to infer surface conditions of temperature and atmospheric humidity by inverting the bulk transfer relations of the sensible and latent heat fluxes. E.g. for temperature we can solve the following relation for T_s, the aerodynamic surface temperature:

$T_a = T_s - \frac{H}{(\rho \cdot G_{ah} \cdot c_p)}$

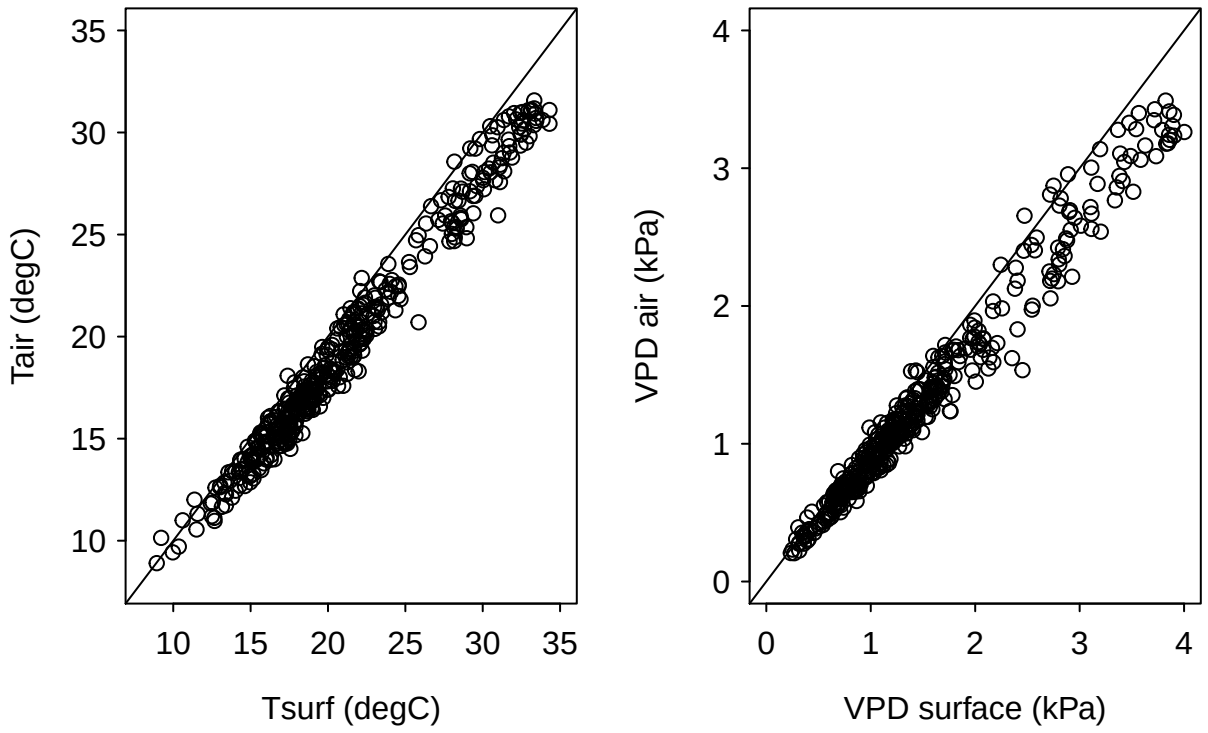
where T_a is air temperature, H is the sensible heat flux (W m^{-2}) ρ is air density (kg m^{-3}), G_{ah} is the aerodynamic conductance for heat (m s^{-1}), and c_p is the specific heat of air (J $K^{-1}kg^{-1}$). In `bigleaf`, the following function calculates conditions at the big-leaf surface:

```
surf <- surface.conditions(tha_filtered,calc.surface.CO2=TRUE)
summary(surf)
#>      Tsurf          esat_surf         esurf          VPD_surf
#>  Min.   : 8.934   Min.   :1.141   Min.   :0.6892   Min.   :0.2352
#>  1st Qu.:17.166   1st Qu.:1.954   1st Qu.:0.9194   1st Qu.:0.8861
#>  Median :19.500   Median :2.261   Median :1.0499   Median :1.2587
#>  Mean   :20.994   Mean   :2.621   Mean   :1.1332   Mean   :1.4879
#>  3rd Qu.:23.469   3rd Qu.:2.883   3rd Qu.:1.2196   3rd Qu.:1.7731
#>  Max.   :34.312   Max.   :5.402   Max.   :2.3386   Max.   :4.0024
#>  NA's   :1013     NA's   :1013    NA's   :1013     NA's   :1013
#>      qsurf           rH_surf         Ca_surf
#>  Min.   :0.0044   Min.   :0.2233   Min.   :378.7
#>  1st Qu.:0.0059   1st Qu.:0.3629   1st Qu.:385.1
#>  Median :0.0067   Median :0.4490   Median :389.2
#>  Mean   :0.0073   Mean   :0.4651   Mean   :390.9
#>  3rd Qu.:0.0078   3rd Qu.:0.5478   3rd Qu.:394.0
#>  Max.   :0.0150   Max.   :0.8680   Max.   :429.3
#>  NA's   :1013     NA's   :1013     NA's   :1013
tha_filtered <- cbind(tha_filtered,surf)
```

By default, the function calculates surface temperature and several humidity measures, including VPD and relative humidity. If we set `calc.surface.CO2=TRUE`, the CO_2 concentration at the surface is calculated additionally. Useful to know is that the expression "surface" depends on what kind of aerodynamic conductance we provide. If $G_a = G_{ah}$, we derive the conditions at the notional canopy surface (or the "big-leaf" surface). If $G_a = G_{am}$, we derive conditions in the intercanopy airspace (because G_a does not account for the leaf boundary layer).

We can compare the surface and air temperature:

```
par(mfrow=c(1,2),mar=c(5,4,2,0.5))
plot(tha_filtered[,"Tair"] ~ tha_filtered[,"Tsurf"],xlim=c(8,35),ylim=c(8,35),las=1,
     xlab="Tsurf (degC)",ylab="Tair (degC)",mgp=c(2.2,0.5,0),tcl=-0.2)
abline(0,1)
plot(tha_filtered[,"VPD"] ~ tha_filtered[,"VPD_surf"],xlim=c(0,4),ylim=c(0,4),las=1,
     xlab="VPD surface (kPa)",ylab="VPD air (kPa)",mgp=c(2.2,0.5,0),tcl=-0.2)
abline(0,1)
```



Both surface temperature and VPD are in most cases higher than the ones measured at tower height.

Surface conductance

Knowledge on G_a allows us to calculate the bulk surface conductance (G_s) of the site (In this case by inverting the Penman-Monteith equation). Gs represents the combined conductance of the vegetation and the soil to water vapor transfer (and as such it is not a purely physiological quantity). Calculating G_s in `bigleaf` is simple:

```
summary(surface.conductance(tha_filtered))
#> Ground heat flux G is not provided and set to 0.
#> Energy storage fluxes S are not provided and set to 0.
#>       Gs_ms             Gs_mol
#>  Min.   :0.0000   Min.   :0.0014
#>  1st Qu.:0.0026   1st Qu.:0.1039
#>  Median :0.0041   Median :0.1641
#>  Mean   :0.0044   Mean   :0.1752
#>  3rd Qu.:0.0059   3rd Qu.:0.2359
#>  Max.   :0.0172   Max.   :0.6970
```

```
#>  NA's   :1013     NA's   :1013
```

The function output is another data.frame with two columns which only differ in the unit of G_s (i.e. a hopeless attempt to make both physicists and physiologists happy). One in m s^{-1} and one in mol m^{-2} s^{-1}. In this function we have ignored the ground heat flux (G) and the storage fluxes (S), and the function politely reminds us of this omission by printing the first two lines of the output (it also tells us what it does, it assumes they are 0 in each time step). In this case we do not have information on the storage fluxes, but we have measurements on the ground heat flux, which we should add to the function call:

```
Gs <- surface.conductance(tha_filtered,G="G")
#> Energy storage fluxes S are not provided and set to 0.
summary(Gs)
#>      Gs_ms            Gs_mol
#>  Min.   :0.0000   Min.   :0.0014
#>  1st Qu.:0.0026   1st Qu.:0.1037
#>  Median :0.0041   Median :0.1647
#>  Mean   :0.0044   Mean   :0.1765
#>  3rd Qu.:0.0060   3rd Qu.:0.2386
#>  Max.   :0.0173   Max.   :0.7016
#>  NA's   :1013     NA's   :1013
tha_filtered <- cbind(tha_filtered,Gs)
```

Again, we have added the two output columns to our data.frame `tha_filtered`.

Stomatal slope parameter

With both G_s and G_a available, we can estimate the stomatal slope parameter g_1. The g_1 parameter characterizes the slope between the surface conductance and the gross carbon uptake (GPP) of the ecosystem, and is thus strongly related to the ecosystem-level intrinsic water-use efficiency. However, it corrects for the confounding effects of VPD and C_a, and is thus better comparable across sites than e.g. GPP/G_s.

```
## stomatal slope from the USO model (Medlyn et al. 2011)
g1_USO <- stomatal.slope(tha_filtered,model="USO",g0=0,robust.nls=TRUE)
#> Respiration from the leaves is ignored and set to 0.
g1_USO
#> Nonlinear regression model
#>   model: Gs ~ g0 + DwDc * (1 + g1/sqrt(VPD)) * GPP/Ca
#>    data: parent.frame()
#>    g1
#> 1.016
#>  weighted residual sum-of-squares: 1.916
#>
#> Number of iterations to convergence: 1
#> Achieved convergence tolerance: 3.132e-09
```

In this case, we have estimated g_1 from the USO (optimal stomatal optimization) model as described in Medlyn et al. 2011. The output is a model object that prints the model formulat that is used to estimate g_1, the estimated parameter value(s), as well as the weighted residual sum-of-squares. Further information on this model object can be obtained using the `summary` function. In this case we have fixed the model intercept g_0 to 0 (this could also be any other value). We can also try to estimate g_1 and g_0 simultaneously (if we add `fitg0=TRUE` to the function call above), but note that the two parameters are usually correlated, and that the values of g_0 are not straightforward to interpret (especially at ecosystem level). The option `robust.nls=TRUE` specifies that g_1 is determined by a robust non-linear regression routine (from the `robustbase` package). We recommend to use this option since otherwise the parameter estimates are sensitive to outliers in G_s, which often occur even in filtered EC datasets. By default, the model takes VPD and atmospheric CO_2 concentration as measured at the tower as input. We can also calculate g_1 by taking the surface conditions,

which are probably more relevant for plant physiological processes than those measured a certain distance above the canopy:

```
## stomatal slope from the USO model (Medlyn et al. 2011)
stomatal.slope(tha_filtered,Tair="Tsurf",VPD="VPD_surf",Ca="Ca_surf",model="USO",
               g0=0,robust.nls=TRUE)
#> Respiration from the leaves is ignored and set to 0.
#> Nonlinear regression model
#>   model: Gs ~ g0 + DwDc * (1 + g1/sqrt(VPD)) * GPP/Ca
#>    data: parent.frame()
#>    g1
#> 1.078
#>  weighted residual sum-of-squares: 1.898
#>
#> Number of iterations to convergence: 1
#> Achieved convergence tolerance: 1.467e-09
```

which in this case, does not change our g_1 value significantly.

We can also calculate g_1 using two different models. One is the long-standing Ball & Berry model (Ball et al. 1987), and the other one is a modification of the Ball & Berry model suggested by Leuning 1995:

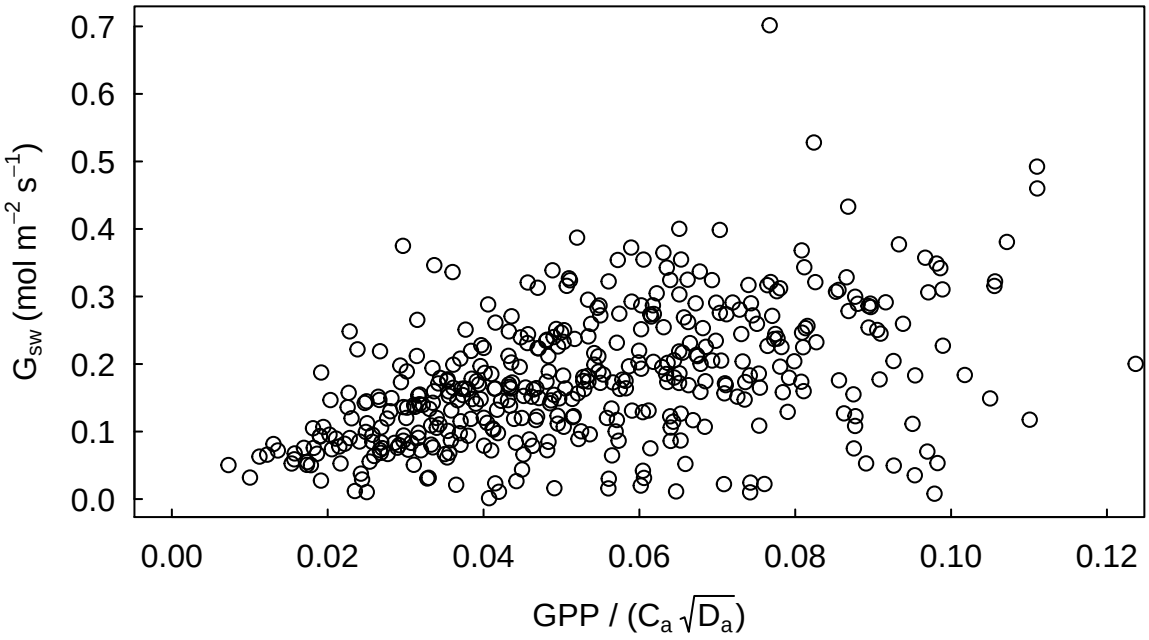
```
## Ball&Berry slope
stomatal.slope(tha_filtered,model="Ball&Berry",g0=0,robust.nls=TRUE)
#> Respiration from the leaves is ignored and set to 0.
#> Nonlinear regression model
#>   model: Gs ~ g0 + g1 * (GPP * rH)/Ca
#>    data: parent.frame()
#>    g1
#> 6.426
#>  weighted residual sum-of-squares: 2.466
#>
#> Number of iterations to convergence: 1
#> Achieved convergence tolerance: 2.159e-10
```

```
## Leuning slope
stomatal.slope(tha_filtered,model="Leuning",g0=0,fitD0=TRUE,robust.nls=TRUE)
#> Respiration from the leaves is ignored and set to 0.
#> Nonlinear regression model
#>   model: Gs ~ g0 + g1 * GPP/((Ca - Gamma) * (1 + VPD/D0))
#>    data: parent.frame()
#>     g1     D0
#>  3.044 17.694
#>  weighted residual sum-of-squares: 1.735
#>
#> Number of iterations to convergence: 5
#> Achieved convergence tolerance: 3.954e-06
```

Note that the absolute value of the g_1 parameter depends on the model. In the Leuning model, we have a third parameter D_0 that can again either be estimated (as in the example above) or fixed to a pre-defined value (by default 1.5 kPa). D_0 describes the stomatal sensitivity to VPD (higher values correspond to a lower stomatal sensitivity to VPD - note however that g_1 and D_0 are strongly correlated, which makes an independent estimates of D_0 difficult to achieve).

We can visualize the g_1 parameter by plotting G_s against the "stomatal index":

```
stomatal_index <- tha_filtered[,"GPP"] / (tha_filtered[,"Ca"] * sqrt(tha_filtered[,"VPD"]))

plot(tha_filtered[,"Gs_mol"] ~ stomatal_index,las=1,
     xlab=expression("GPP / (C"["a"]~sqrt("D"["a"])*")"),
     ylab=expression("G"["sw"]~"(mol m"^{-2}~"s"^{-1}*")"),
     tcl=0.2,mgp=c(2.2,0.5,0),xlim=c(0,0.12))
```

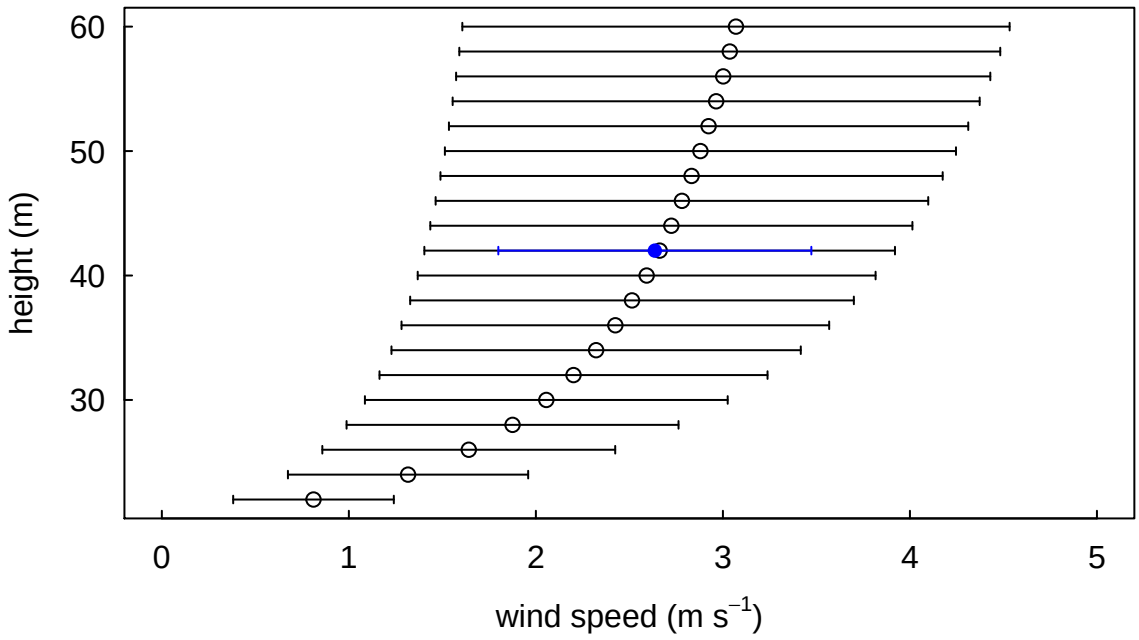


Wind profile

The 'big-leaf' framework assumes that wind speed is zero at height d + z_{0m} (where z_{0m} is the roughness length for momentum) and then increases exponentially with height. The shape of the wind profile further depends on the stability conditions of the air above the canopy. In `bigleaf`, a wind profile can be calculated assuming an exponential increase with height, which is affected by atmospheric stability. Here, we calculate wind speed at heights of 22-60m in steps of 2m. As expected, the gradient in wind speed is strongest close to the surface and weaker at greater heights:

```
wind_heights <- seq(22,60,2)
wp <- wind.profile(tha_filtered,heights=wind_heights,zh=zh,zr=zr)
wp_means <- colMeans(wp,na.rm=TRUE)
wp_sd    <- apply(wp,2,sd,na.rm=TRUE)

plot(wind_heights ~ wp_means,xlab=expression("wind speed (m s"^{-1}*")"),ylab="height (m)",
     las=1,mgp=c(2.2,0.5,0),tcl=0.2,xlim=c(0,5))
arrows(wp_means-wp_sd,wind_heights,wp_means+wp_sd,wind_heights,angle=90,
       length=0.02,code=3)
points(x=mean(tha_filtered[,"wind"],na.rm=TRUE),y=zr,col="blue",pch=16)
```

```
arrows(mean(tha_filtered[,"wind"],na.rm=TRUE)-sd(tha_filtered[,"wind"],na.rm=TRUE),
       zr,mean(tha_filtered[,"wind"],na.rm=TRUE)+sd(tha_filtered[,"wind"],na.rm=TRUE),
       zr,angle=90,length=0.02,code=3,col="blue")
```



Here, the points denote the mean wind speed and the bars denote the standard deviation. The blue point/bar represent the values that were measured at 42m. In this case we see that the wind speed as "back-calculated" from the wind profile agrees well with the actual measurements.

Potential evapotranspiration

For many hydrological applications, it is relevant to get an estimate on the potential evapotranspiration (PET). At the moment, the `bigleaf` package contains two formulations for the estimate of PET: the Priestley-Taylor equation, and the Penman-Monteith equation:

```
summary(potential.ET(tha_filtered,G="G",approach="Priestley-Taylor"))
#> Energy storage fluxes S are not provided and set to 0.
#>      ET_pot           LE_pot
#>  Min.   :0e+00   Min.   :  5.416
#>  1st Qu.:1e-04   1st Qu.:182.321
#>  Median :1e-04   Median :341.978
#>  Mean   :1e-04   Mean   :346.476
#>  3rd Qu.:2e-04   3rd Qu.:503.071
#>  Max.   :3e-04   Max.   :711.343
#>  NA's   :1013    NA's   :1013
summary(potential.ET(tha_filtered,G="G",approach="Penman-Monteith"),
        Gs_pot=quantile(tha_filtered$Gs_mol,0.95,na.rm=TRUE))
#> Energy storage fluxes S are not provided and set to 0.
```

```
#>      ET_pot          LE_pot
#>  Min.   :0e+00   Min.   : 56.47
#>  1st Qu.:1e-04   1st Qu.:201.42
#>  Median :1e-04   Median :283.86
#>  Mean   :1e-04   Mean   :310.05
#>  3rd Qu.:2e-04   3rd Qu.:384.33
#>  Max.   :3e-04   Max.   :751.18
#>  NA's   :1013    NA's   :1013
```

In the second calculation it is important to provide an estimate of `Gs_pot`, which corresponds to the potential surface conductance under optimal conditions. Here, we have approximated `Gs_pot` with the 95$^{\text{th}}$ percentile of all G_s values of the site.

Energy balance closure (EBC)

The `bigleaf` package offers a function which characterizes the degree of the EBC (i.e. $A = \lambda E + H$, where A is available energy, λE is the latent heat flux, and H is the sensible heat flux, all in W m^{-2}). We can calculate the EBC with the following command:

```
energy.closure(tha)
#> Ground heat flux G is not provided and set to 0.
#> Energy storage fluxes S are not provided and set to 0.
#>         n intercept     slope       r^2       EBR
#>  1440.000     0.796     0.685     0.888     0.690
```

The output tells us the number of observations that were used for the calculation of the EBC (n; note that we took the unfiltered data.frame here), the intercept and slope of the LE + H ~ A plot, the r^2 of the regression, and the energy balance ratio (EBR = $\frac{\lambda E + H}{R_n - G - S}$). Thus, the degree of EBC is characterized by two metrics, the slope of the λE + H ~ A relationship, and the EBR. In this case they agree relatively well; both indicate a gap in the energy balance of ~ 30%. In the calculations above, we did not include the ground heat G into the equation, which is the default setting (i.e. A was assumed to equal R_n). We can now have a look to what extent the EBC improves when we consider G (i.e. $A = R_n - G$):

```
energy.closure(tha,G="G")
#> Energy storage fluxes S are not provided and set to 0.
#>         n intercept     slope       r^2       EBR
#>  1440.000     0.633     0.699     0.885     0.703
```

In this case the ground heat flux improves the EBC, but only marginally. This implies that there are other reasons for the EBC, including an underestimation of the turbulent fluxes. It should be clear, however, that this example is not representative for all EC sites. In general, G is more important (and S is less important) at sites with low biomass and short vegetation.

Meteorological variables

The `bigleaf` package provides calculation routines for a number of meteorological variables, which are basic to the calculation of many other variables. A few examples on their usage are given below:

```
# Saturation vapor pressure (kPa) and slope of the saturation vapor pressure curve (kPa K-1)
Esat.slope(Tair=25)
#>       Esat     Delta
#> 1 3.160057 0.1883055
```

```
# psychrometric constant (kPa K-1)
psychrometric.constant(Tair=25,pressure=100)
#> [1] 0.0661611
```

```
# air density (kg m-3)
air.density(Tair=25,pressure=100)
#> [1] 1.168408
```

```
# dew point (degC)
dew.point(Tair=25,VPD=1)
#> [1] 18.764
```

```
# wetbulb temperature (degC)
wetbulb.temp(Tair=25,pressure=100,VPD=1)
#> [1] 20.648
```

```
# estimate atmospheric pressure from elevation (hypsometric equation)
pressure.from.elevation(elev=500,Tair=25)
#> [1] 95.68129
```

Unit interconversions

The package further provides a number of useful unit interconversions, which are straightforward to use (please make sure that the input variable is in the right unit, e.g. rH has to be between 0 and 1 and not in percent):

```
# VPD to vapor pressure (e, kPa)
VPD.to.e(VPD=2,Tair=25)
#> [1] 1.160057
```

```
# vapor pressure to specific humidity (kg kg-1)
e.to.q(e=1,pressure=100)
#> [1] 0.006243601
```

```
# relative humidity to VPD (kPa)
rH.to.VPD(rH=0.6,Tair=25)
#> [1] 1.264023
```

```
# conductance from ms-1 to mol m-2 s-1
ms.to.mol(G_ms=0.01,Tair=25,pressure=100)
#> [1] 0.4033932
```

```
# umol CO2 m-2 s-1 to g C m-2 d-1
umolCO2.to.gC(CO2_flux=20)
#> [1] 20.75501
```

Useful hints for advanced users

Hide function messages

As shown earlier in this tutorial, many functions of the `bigleaf` package print messages to make the reader aware that e.g. some flux components are missing. This output can be a bit annoying when functions are used in loops or `apply`-functions. A simple way to not show these messages is to use a combination of `invisible` and `capture.output`:

```
## instead of
PET <- potential.ET(Tair=25,pressure=100,Rn=200)
#> Ground heat flux G is not provided and set to 0.
```

```
#> Energy storage fluxes S are not provided and set to 0.
## one can use
invisible(capture.output(PET <- potential.ET(Tair=25,pressure=100,Rn=200)))
```

Constants

The `bigleaf` package contains a single list of constants (see `?bigleaf.constants`). Whenever one or more constants are used in a function, this list is provided as a default argument, so the user does usually not need to interact with this list. However, should you wish to change a certain constant for the calculations (which could make sense in some cases, e.g. using a different value for the von-Karman constant (k)), individual constants can be changed within a function call. As an example, let's call a function with the `bigleaf` default value of k=0.41, and the alternative, often used value of k=0.4:

```
summary(aerodynamic.conductance(tha_filtered,wind_profile=TRUE,zr=zr,d=0.7*zh,z0m=2.65)[,"Ga_h"])
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#>  0.0278  0.0637  0.0742  0.0714  0.0813  0.1031    1013
summary(aerodynamic.conductance(tha_filtered,wind_profile=TRUE,zr=zr,d=0.7*zh,z0m=2.65,
                                constants=bigleaf.constants(k=0.4))[,"Ga_h"])
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#>  0.0275  0.0628  0.0733  0.0705  0.0803  0.1018    1013
```

We see that in this case, small changes in k have an effect on the calculated values of G_{ah}, but they do not change the results significantly (however, the same value of k should be used for all calculations).

Boundary layer conductance for trace gases

By default, the function `aerodynamic.conductance` returns the (quasi-laminar) canopy boundary layer (G_b) for heat and water vapor (which are assumed to be equal in the `bigleaf` package), as well as for CO_2. The function further provides the possibility to calculate G_b for other trace gases, provided that the respective Schmidt number is known. Further, if we are only interested in G_b (or the kB^{-1} parameter) we can use one of the following functions: `Gb.Thom`, `Gb.Choudhury`, `Gb.Su`. These functions are integrated in the main function `aerodynamic.conductance`, but the modular design of the package allows them to be called separately. We can demonstrate the calculation of G_b for methane (CH_4) with a simple example:

```
summary(Gb.Thom(tha_filtered$ustar))
#>       Gb_h              Rb_h              kB_h           Gb_CO2
#>  Min.   :0.0551   Min.   : 5.854   Min.   :1.487   Min.   :0.0419
#>  1st Qu.:0.0947   1st Qu.: 7.865   1st Qu.:1.948   1st Qu.:0.0719
#>  Median :0.1109   Median : 9.020   Median :2.108   Median :0.0842
#>  Mean   :0.1110   Mean   : 9.450   Mean   :2.097   Mean   :0.0843
#>  3rd Qu.:0.1271   3rd Qu.:10.561   3rd Qu.:2.257   3rd Qu.:0.0966
#>  Max.   :0.1708   Max.   :18.139   Max.   :2.616   Max.   :0.1298
#>  NA's   :1013     NA's   :1013     NA's   :1013    NA's   :1013
summary(Gb.Thom(tha_filtered$ustar,Sc=0.99,Sc_name="CH4"))
#>       Gb_h              Rb_h              kB_h           Gb_CO2
#>  Min.   :0.0551   Min.   : 5.854   Min.   :1.487   Min.   :0.0419
#>  1st Qu.:0.0947   1st Qu.: 7.865   1st Qu.:1.948   1st Qu.:0.0719
#>  Median :0.1109   Median : 9.020   Median :2.108   Median :0.0842
#>  Mean   :0.1110   Mean   : 9.450   Mean   :2.097   Mean   :0.0843
#>  3rd Qu.:0.1271   3rd Qu.:10.561   3rd Qu.:2.257   3rd Qu.:0.0966
#>  Max.   :0.1708   Max.   :18.139   Max.   :2.616   Max.   :0.1298
#>  NA's   :1013     NA's   :1013     NA's   :1013    NA's   :1013
#>      Gb_CH4
#>  Min.   :0.0441
#>  1st Qu.:0.0758
```

```
#>  Median :0.0887
#>  Mean   :0.0888
#>  3rd Qu.:0.1018
#>  Max.   :0.1367
#>  NA's   :1013
```

In the first line we get the standard output of the function, whereas in the second line we get in addition the G_b for methane.

Dealing with uncertainties

It is important to note that the `bigleaf` package does not calculate uncertainties of most variables. This is firstly because it is challenging to properly account for all the uncertainties present in EC data, and secondly because this would lead to much slower and more complex function calls. Nevertheless, uncertainties of the calculated ecosystem properties should not be ignored. Here, we present two main strategies on how to quantify uncertainties: 1) bootstrapping, and 2) Monte Carlo analysis. In general, we leave the calculations/function calls untouched, but we add wrapper functions that use different techniques (e.g. bootstrapping) to calculate uncertainties of the output variables.

Bootstrapping

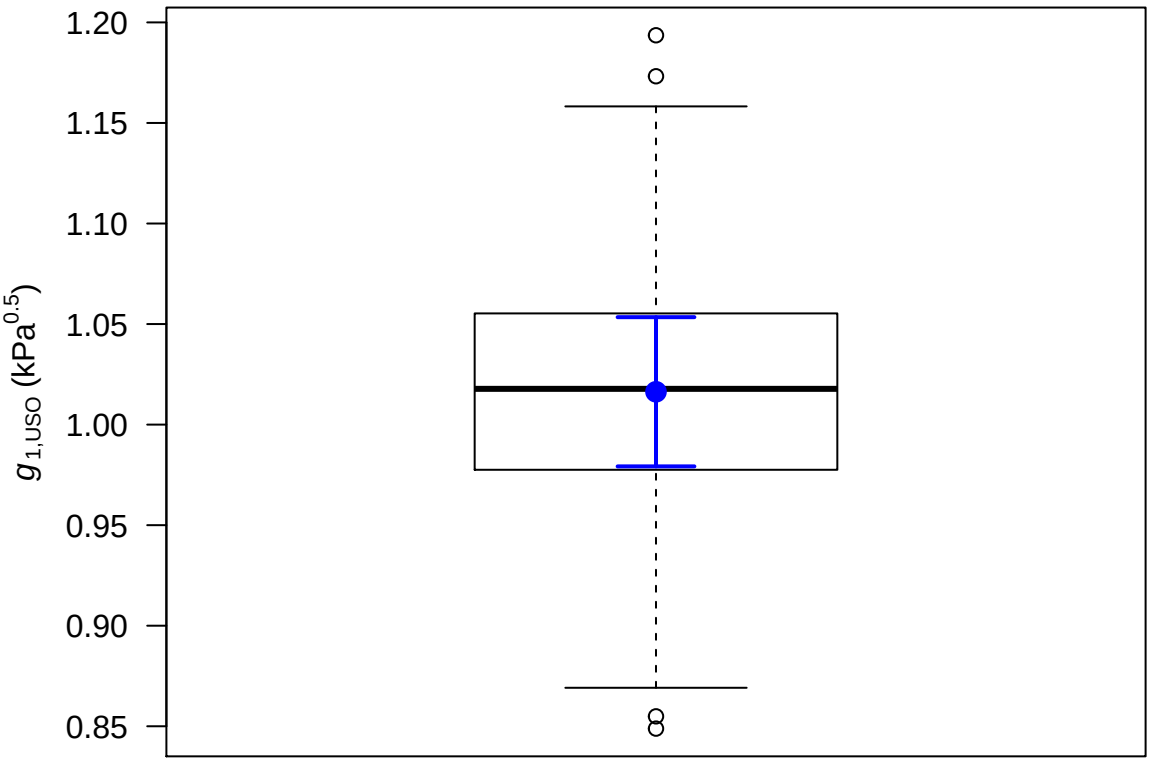
As a first example, we use bootstrapping to estimate the uncertainty of the g_1 parameter calculated above. The principle is easy: we calculate g_1 a given number of times (in this case 300 times), and each time we only use a (different) subset of the data. In each iteration, 25% of the data are discarded. To do this, we can define the following function (note that this function can be written in a more efficient way, but by using a loop the principle becomes clear):

```
G1.bootstrap <- function(dat,LoopNum,SampSizeRel){
  # dat         = input data.frame
  # LoopNum     = number of iterations
  # SampSizeRel = fraction of data sampled for each iteration
  dfout=data.frame(matrix(NA,nrow = LoopNum,ncol = 0)) #Define output dataframe
  dat$RunNum=1:nrow(dat)
  SampSize=round(length(dat$RunNum)*SampSizeRel) #calculate number of data used for resampling

  for (m in 1:LoopNum){
    # sample data:
    SampIDX=sample(x = dat$RunNum,size = SampSize,replace = T)
    # run the function on the sample data:
    dfout$G1[m]=summary(stomatal.slope(data = dat[SampIDX,],
                                       Tair = dat$Tair[SampIDX],
                                       Gs=dat$Gs_mol[SampIDX],
                                       pressure = dat$pressure[SampIDX],
                                       GPP = dat$GPP[SampIDX],
                                       VPD = dat$VPD[SampIDX],
                                       Ca = dat$Ca[SampIDX],
                        model="USO",g0=0,robust.nls=T))$coef[1,1]
  }

  return(dfout) # return output dataframe
}
```

We can use this function with our data:

```
# 300 times resampling; each time 75 % of the data:
tha_G1BT <- G1.bootstrap(dat = tha_filtered,LoopNum = 300,SampSizeRel = 0.75)
```

```
# estimate using all data:
g1_mean <- summary(g1_USO)$coef[1,1]
g1_se   <- summary(g1_USO)$coef[1,2]

par(mar=c(2,6,1,1))
boxplot(tha_G1BT,ylab=expression(italic("g")["1,USO"]~"(kPa"^{0.5}*")"),las=1,mgp=c(2,0.5,0))
points(g1_mean,col="blue",pch=16,cex=1.5)
arrows(1,g1_mean - g1_se,1,g1_mean + g1_se,angle=90,length=0.2,code=3,col="blue",lwd=2)
```



The blue point shows the estimate (+/- standard error) when we take all data, as calculated above. The two estimates agree very well, indicating that in this case we can be very confident on the calculated g_1 estimate. The bootstrapping technique can also be applied to other (regression-based) functions in the package.

Monte Carlo analysis

In the second example we implement a simple Monte Carlo analysis in which we propagate uncertainties in the calculation of G_a to uncertainties in G_s (which takes G_a as input). To do this, we first estimate the uncertainty in G_a that is caused by uncertainties in three of its input parameters: the leaf characteristic dimension D_l, the LAI, and the roughness length z_{0m}. The uncertainty of other parameters could be included, but for demonstration purposes we only use these three. First, we have to assess the mean and the error distribution of the input parameters. We estimated D_l = 1cm, LAI=7.6 (as measured at the site), and $z_{0m} = 2.65m$ (10% of the vegetation height), and we assume that their errors are normally distributed with a standard deviation (sd) equal to 25% of the mean in case of z_{0m} and D_l. In case of LAI we assume a sd of 0.5.

```
n_pert <- 200
z0m1   <- 2.65
Dl1    <- 0.01
```

```
LAI1   <- 7.6
z0m_sample <- pmax(rnorm(n=n_pert,mean=z0m1,sd=0.25*z0m1),0)
Dl_sample  <- pmax(rnorm(n=n_pert,mean=Dl1,sd=0.25*Dl1),0)
LAI_sample <- rnorm(n=n_pert,mean=LAI1,sd=0.5)
```

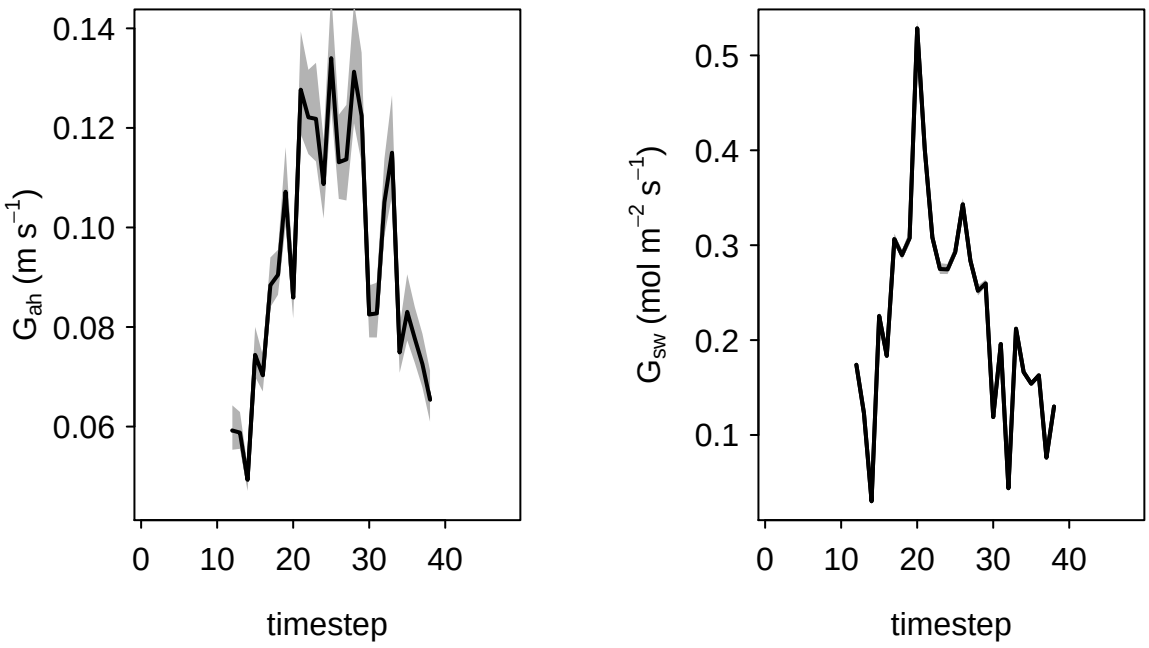
In the example above we create a parameter space that we use for the subsequent calculations. We have chosen the most simple settings here, that means we assume that parameters have a normal error distribution and that they are independent of each other. In many cases these assumptions are not valid. For example, measured fluxes are more likely to have a Laplace error distribution (Hollinger & Richardson 2005), which would be better sampled using `rlaplace` from the `rmutil` package instead of `rnorm`. In many cases, the parameters are also not independent of each other. In our case, z_{0m} and D_l may not be strongly correlated, but one would possibly expect a correlation between LAI and z_{0m}. We can account for dependencies among variables by doing the sampling based on a variance-covariance matrix that prescribes correlations between variables.

```
unc_all <- mapply(aerodynamic.conductance,Dl=Dl_sample,z0m=z0m_sample,LAI=LAI_sample,
                        MoreArgs=list(data=tha_filtered,zr=42,zh=26.5,d=0.7*26.5,
                                      N=2,stab_correction=T,
                                      stab_formulation="Dyer_1970",
                                      Rb_model="Su_2001")
                  )

# select "Ga_h" output variable and convert to matrix
unc_Ga_h <- matrix(unlist(unc_all["Ga_h",]),ncol=n_pert,byrow=FALSE)

# calculate 2.5th, 50th, and 97.5th quantile of the n_pert calculations for every timestep
Ga_low  <- apply(unc_Ga_h,1,quantile,0.025,na.rm=T)
Ga_mean <- apply(unc_Ga_h,1,quantile,0.5,na.rm=T)
Ga_high <- apply(unc_Ga_h,1,quantile,0.975,na.rm=T)
Ga <- cbind(Ga_low,Ga_mean,Ga_high)
summary(Ga)
#>      Ga_low          Ga_mean          Ga_high
#>  Min.   :0.0206   Min.   :0.0206   Min.   :0.0206
#>  1st Qu.:0.0691   1st Qu.:0.0726   1st Qu.:0.0765
#>  Median :0.0884   Median :0.0935   Median :0.0996
#>  Mean   :0.0882   Mean   :0.0938   Mean   :0.1011
#>  3rd Qu.:0.1072   3rd Qu.:0.1142   3rd Qu.:0.1250
#>  Max.   :0.3135   Max.   :0.3177   Max.   :0.3208
#>  NA's   :1013     NA's   :1013     NA's   :1013


# calculate the Gs for the three Ga estimates
Gs_low  <- surface.conductance(tha_filtered,Ga=Ga[,"Ga_low"],G="G")[,"Gs_mol"]
#> Energy storage fluxes S are not provided and set to 0.
Gs_mean <- surface.conductance(tha_filtered,Ga=Ga[,"Ga_mean"],G="G")[,"Gs_mol"]
#> Energy storage fluxes S are not provided and set to 0.
Gs_high <- surface.conductance(tha_filtered,Ga=Ga[,"Ga_high"],G="G")[,"Gs_mol"]
#> Energy storage fluxes S are not provided and set to 0.
Gs <- cbind(Gs_low,Gs_mean,Gs_high)
summary(Gs)
#>      Gs_low          Gs_mean          Gs_high
#>  Min.   :0.0013   Min.   :0.0014   Min.   :0.0014
#>  1st Qu.:0.1034   1st Qu.:0.1037   1st Qu.:0.1041
#>  Median :0.1633   Median :0.1648   Median :0.1666
```

```
#>  Mean   :0.1747   Mean   :0.1766   Mean   :0.1789
#>  3rd Qu.:0.2361   3rd Qu.:0.2388   3rd Qu.:0.2405
#>  Max.   :0.7204   Max.   :0.7002   Max.   :0.6789
#>  NA's   :1013     NA's   :1013     NA's   :1013
```

The first and the last columns of the output give us now an uncertainty envelope around our G_a and G_s calculations. The example shows that variations in the three input parameters are sensitive for the estimation of G_a, but not so much for G_s:

```
par(mfrow=c(1,2))
ind <- c(1:48) # first day
plot(Ga_mean[ind],type="l",lwd=2,xlab="timestep",ylab=expression("G"["ah"]~"(m s"^{-1}*")"),
     las=1,mgp=c(2.2,0.5,0),tcl=-0.2,ylim=c(0.045,0.14))

ok <- which(!is.na(Ga_mean[ind]))
polygon(c(ok,rev(ok)),c(Ga_high[ind][ok],rev(Ga_low[ind][ok])),
        col="grey70",border=NA)
points(Ga_mean[ind],type="l",lwd=2)


plot(Gs_mean[ind],type="l",lwd=2,xlab="timestep",tcl=-0.2,
     ylab=expression("G"["sw"]~"(mol m"^{-2}~"s"^{-1}*")"),las=1,mgp=c(2.2,0.5,0))

ok <- which(!is.na(Gs_mean[ind]))
polygon(c(ok,rev(ok)),c(Gs_high[ind][ok],rev(Gs_low[ind][ok])),
        col="grey70",border=NA)
points(Gs_mean[ind],type="l",lwd=2)
```



In general, these operations are more effectively implemented elsewhere, and we just show an example for demonstration purposes. The reader might be interested in the `FME` package (in particular the `sensRange` function). The package also provides functions (e.g. `Norm`) that generates parameter sets based on a parameter variance-covariance matrix.

References

Ball, J. T.; Woodrow, I. E. & Berry, J. A. Biggins, J. (Ed.) A model predicting stomatal conductance and its contribution to the control of photosynthesis under different environmental conditions Progress in photosynthesis research, Martinus Nijhoff Publishers, Dordrecht, Netherlands, 1987, 221-224.

Grünwald, T. & Bernhofer, C. A decade of carbon, water and energy flux measurements of an old spruce forest at the Anchor Station Tharandt Tellus B, Wiley Online Library, 2007, 59, 387-396.

Hollinger, D. & Richardson, A. Uncertainty in eddy covariance measurements and its application to physiological models. Tree physiology, 2005, 25, 873-885.

Knauer, J., El-Madany, T.S., Zaehle, S., Migliavacca, M. An R package for the calculation of physical and physiological ecosystem properties from eddy covariance data. PLoS ONE, 2018, e0201114.

Leuning, R. A critical appraisal of a combined stomatal-photosynthesis model for C3 plants Plant, Cell & Environment, Wiley Online Library, 1995, 18, 339-355.

Leuning, R.; Van Gorsel, E.; Massman, W. J. & Isaac, P. R. Reflections on the surface energy imbalance problem Agricultural and Forest Meteorology, 2012, 156, 65-74.

Medlyn, B. E.; Duursma, R. A.; Eamus, D.; Ellsworth, D. S.; Prentice, I. C.; Barton, C. V.; Crous, K. Y.;

de Angelis, P.; Freeman, M. & Wingate, L. Reconciling the optimal and empirical approaches to modelling stomatal conductance. Global Change Biology, 2011, 17, 2134-2144.

Su, Z.; Schmugge, T.; Kustas, W. & Massman, W. An evaluation of two models for estimation of the roughness height for heat transfer between the land surface and the atmosphere. Journal of Applied Meteorology, 2001, 40, 1933-1951.

Thom, A. Momentum, mass and heat exchange of vegetation. Quarterly Journal of the Royal Meteorological Society, 1972, 98, 124-134.

Verma, S. Black, T.; Spittlehouse, D.; Novak, M. & Price, D. (Eds.) Aerodynamic resistances to transfers of heat, mass and momentum Estimation of areal evapotranspiration, Estimation of areal evapotranspiration, International Association of Hydrological Sciences, 1989, 177, 13-20.