MAIDENiso v4 Documentation

The MAIDENiso developer team

February 7, 2023

Lead author

Ignacio Hermoso de Mendoza[1,2]

Contributing authors

Fabio Gennaretti[3], Aliénor Lavergne[4], Guillermo Gea-Izquierdo[5,6], Marceau Baradoux[3], Lucie Nina Barbier[3]

First model developers

[†]Laurent Misson[7] (MAIDEN), Pierre-Alain Danis[8] (isotopic module)

Current developers

Guillermo Gea-Izquierdo[5,6], Etienne Boucher[1,2], Fabio Gennaretti[3], Aliénor Lavergne[4], Ignacio Hermoso de Mendoza[1,2], Joel Guiot[9], Laia Andreu-Hayles[10,11,12]

Affiliations

[1]Centre de Recherche sur la dynamique du système Terre (GEOTOP), Université du Québec à Montréal (UQAM), Canada

[2]Centre d'études nordiques (CEN), Université Laval, Canada

[3]Institut de Recherche sur les Forêts (IRF), Université du Québec en Abitibi-Témiscamingue (UQAT), Canada

[4]Carbon Cycle Research Group, Space and Atmospheric Science, Physics Department, Imperial College London, United Kingdom

[5]Instituto Nacional de Investigación y Tecnología Agraria (INIA), Spain

[6]Consejo Superior de Investigaciones Científicas (CSIC), Spain

[7]Université de Montpellier, France

[8]Office Français de la Biodiversité, France

[9]CEREGE, Aix-Marseille University, CNRS, IRD, France

[10]Tree-Ring Laboratory, Lamont-Doherty Earth Observatory, Columbia University, USA

[11]Ecological and Forestry Applications Research Centre (CREAF), Spain

[12]Catalan Institution for Research and Advanced Studies (ICREA), Spain

Contents

Part I

User Guide

Chapter 1: Overview

1.1 Preamble

MAIDENiso (Modelling and Analysis in Dendroecology + isotopes) is a mechanistic, process-based model simulating the physical and physiological processes of a virtual tree and its environment. MAIDEN simulates the water and carbon fluxes exchanged between forests and the atmosphere, including the influence of phenology on the production and allocation of carbon to different parts of the tree. Because it requires a very limited number of meteorological inputs, the application of the model is possible in regions where data are scarce. MAIDENiso provides two main advantages over other process-based models:

1. The outputs are directly comparable to tree-ring proxies, in addition to carbon and water ecosystem fluxes.
2. It is an isotope-enabled model, allowing users to track down the origin of the climate signal recorded therein.

The original version of the model, MAIDEN [Misson, 2004], was specifically designed to improve the interpretation of tree-ring proxies based on our knowledge about ecophysiological processes and relationships between climate and tree growth. The isotope-enabled version, MAIDENiso v1 [Danis et al., 2012], incorporates calculations of the stable isotopic composition of oxygen (δ^{18}O) and carbon (δ^{13}C) in the different components of the tree. MAIDEN was originally created for tree species in Mediterranean climates, and it has been optimized for *Quercus petraea* (Matt.) Liebl. and 12 Mediterranean species [Misson, 2004, Gaucherel et al., 2008, Boucher et al., 2014, Gea-Izquierdo et al., 2015]. Since then, the phenology and allocation modules have been developed in relation to climatic drivers [Gea-Izquierdo et al., 2015], the phenology and physiological processes have been adapted to simulate tree radial growth in boreal northeastern American forests [Gennaretti et al., 2017] and used to simulate tree-ring cellulose δ^{18}O in boreal and temperate forests of eastern Canada and southern South America [Lavergne et al., 2017]. The latest version, MAIDENiso v4 [Hermoso de Mendoza et al., 2022], incorporates a thermal module and an update to the hydrological module to incorporate a snow layer and ice layers for the soil and the canopy.

1.2 First User Manual

This is the first User Manual ever written for MAIDENiso. With this document, we aim to guide new users towards using MAIDENiso without the need of figuring everything out, as previous users had to do.

Chapter 2: Installation

2.1 Installing a C++ compiler

MAIDENiso is a C++ project. Interpreted programming languages (like R, Python or Matlab) are directly read and executed by an interpreter and therefore do not need to be compiled. Instead, compiled programming languages like C++ must be translated into machine code that the machine can execute, via the process known as compilation. Therefore, the first step towards installing MAIDENiso is to install a C++ compiler that can interpret the code, and create an executable appropriate for your machine.

There are many C++ compilers, here we will guide you towards installing GCC (the GNU Compiler Collection) in three different Operating Systems: Windows, MacOSX and Ubuntu.

2.1.1 Windows installation

For Windows, we recommend the installation of MinGW (Minimalist GNU for Windows). You can do that by following these steps:

1. Download the MinGW installation manager. You can visit the MinGW website (`https://www.mingw-w64.org/downloads/`) for a specific package, or download directly from their repository in SourceForge: `https://sourceforge.net/projects/mingw/`

2. Double click and open the .exe MinGW file and click install. It will automatically start downloading all the setups for the MinGW.

3. After all of the setup click Continue. Now the MinGW installation manager will pop up.

4. In the installation manager right click on mingw32-gcc-g++ (and the other options if you also want to install them) and then click Mark for Installation.

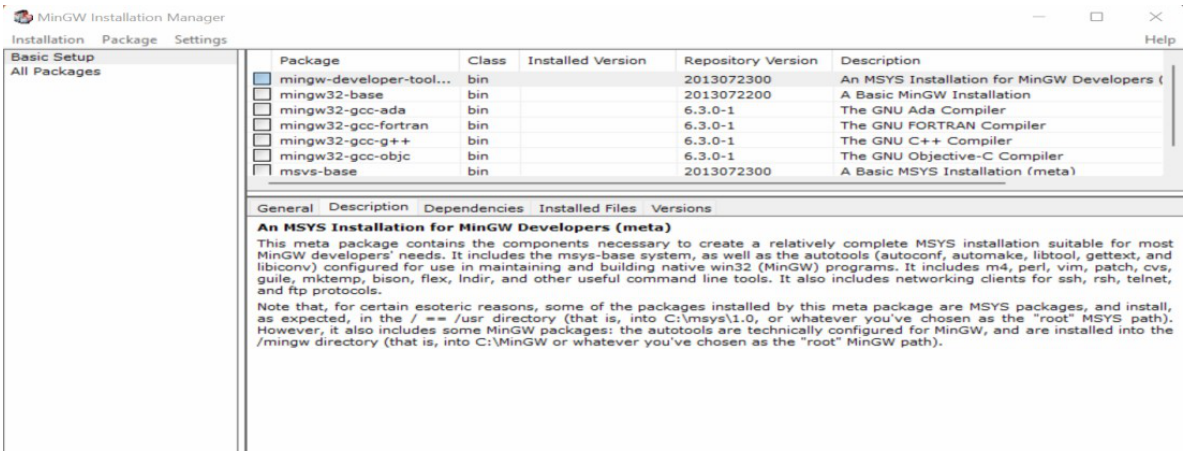

Figure 2.1: MinGW installation manager

5. In the Installation, option-click Apply changes. And then select Apply. It will start downloading all the files (it will take several minutes). After finishing click on Close.

Now MinGW is installed, but if you try opening the terminal and typing "gcc", terminal will say that it does not recognize this command. The reason is that MinGW does not only need to be installed, it also needs to be added to the "path" environment variable of your system. This is, when you type any command, it looks in this "path" looking for binary files with the name of that command to know what to do.

1. Go to the C drive on your device and search for the MinGW folder. And in the MinGW folder go to the bin folder and copy its path.

2. Go to the control panel then go to System then Advanced system settings and then Environment variables. It may also work to simply search "environment variables" in the the search button of Windows.

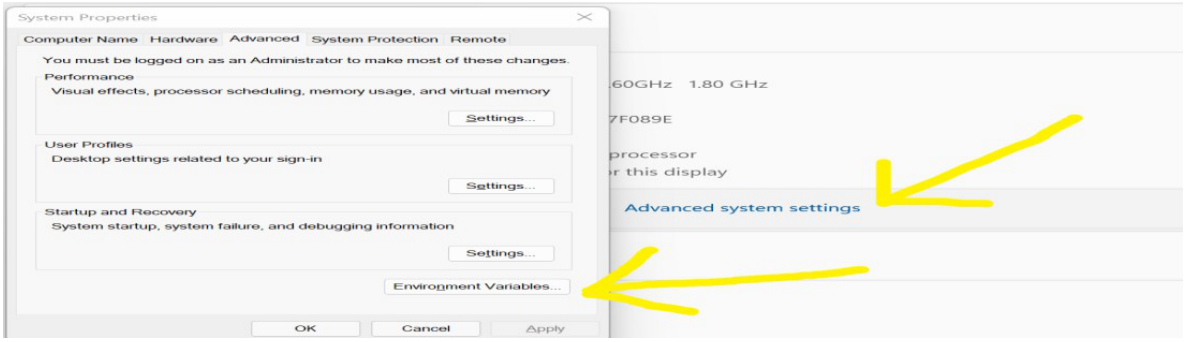

Figure 2.2: Environment variables

3. In the system variables search for path and then select Edit. Now add a new path to it by clicking New. Now paste the path from step 1 and click ok.

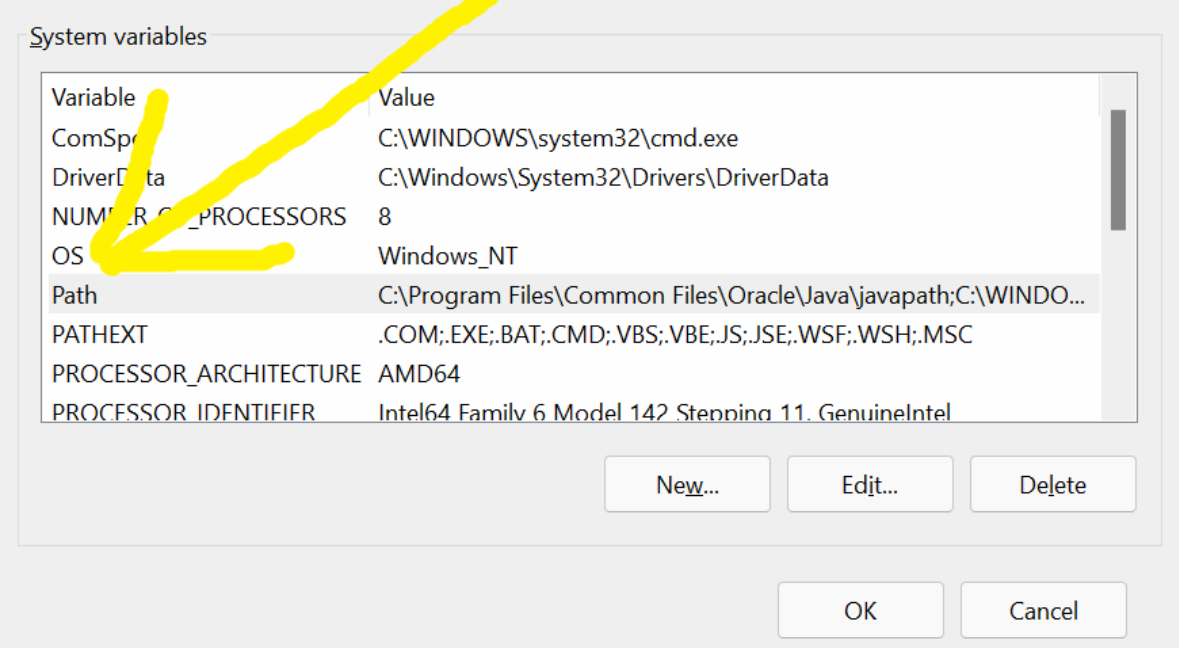

Figure 2.3: System path

Now MinGW is installed and linked. When running "gcc" in terminal, it will not tell you that the command is unrecognized. Instead, it will protest because you did not specify any input files (which files should be compiled). This is good, because now we can proceed to downloading MAIDENiso and compiling it.

2.1.2 MacOSX installation

For MacOSX, installing GCC is easier, we can do that by using MacPorts. It will likely work for later versions of GCC as well. Just substitute "gccx" for "gcc5", where "x" is the desired version.

1. Download and install Xcode from the Mac App Store.
2. Download and install the MacPorts "pkg" installer appropriate for your OS version (e.g., Mavericks is 10.9. Go to "Apple menu ¿ About This Mac" if unsure): Installing MacPorts.
3. Open a Terminal window and enter "sudo port -v selfupdate".
4. Enter your password, and wait for the update to finish.

5. Once finished, enter "sudo port install gcc5".
6. Enter your password if needed. It may take an hour or longer for the download and compile to complete.
7. Once the installation is complete, attempt to compile a C++14 source file using the following command: "g++-mp-5 -Wall -std=c++14 Program.cc -o Program". Note there is a hypen (-) between "g++" and "mp".

Another option is to use homebrew (brew):

1. Download and install Xcode from the Mac App Store.
2. Download and install homebrew. You can do this from the command line by running:

   ```
   ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
   ```

3. Make sure homebrew is up to date by running in command line:

   ```
   brew update
   ```

4. Update the actual packages (to match the versions in the updated local git repository) by running in command line:

   ```
   brew upgrade
   ```

5. Now get information on GCC versions by running:

   ```
   brew info gcc
   ```

6. Now install GCC! Run in command line:

   ```
   brew install gcc
   ```

7. Finally, remove previous application/dependency revisions, thus saving considerable space, by running:

   ```
   brew cleanup
   ```

Either way, now you should have GCC installed and ready to compile.

2.1.3 Ubuntu installation

Finally, let us see how to install GCC in Ubuntu. To be able to add new repositories and install packages on your Ubuntu system, you must be logged in as root or user with sudo privileges. The steps are:

1. Start by updating the packages list:

   ```
   sudo apt update
   ```

2. Install the "build-essential" package (which includes a bunch of packages including gcc, g++ and make.) by typing:

   ```
   sudo apt install build-essential
   ```

3. To validate that the GCC compiler is successfully installed, use the ¡gcc –version command¿ which prints the GCC version:

   ```
   gcc --version
   ```

Now GCC is installed and ready for use.

2.2 Downloading MAIDENiso

The MAIDENiso developer team mantains a webpage under the Université du Québec en Abitibi-Temiscamingue (UQAT), `https://dendro-eco.uqat.ca/maiden/`. This is the main webpage where the publicly available content related to MAIDENiso is released. This includes the published versions of the code, which are available in a Zenodo repository from where anybody can download the code.

MAIDENiso v4, the version of MAIDENiso that this user manual was written for (the model with snow module released on October 2021), can be found in `https://zenodo.org/record/5597877`. To download it, simply download all the files in this Zenodo repository.

2.3 Compiling MAIDENiso

MAIDENiso is a C++ program composed of several source code files (.cpp) and libraries (.h). While compiling single-file programs is quite easy, the compilation of multi-file programs (called projects) requires a rather complicated command, which is coded inside a makefile. A makefile is a file (by default named "Makefile" or "makefile")

containing a set of directives used by a make build automation tool to generate a target/goal. In our case, it contains the call to the compiler with the appropriate flags and the list of files to be included in the compilation. It looks like this:

```
c++ -o executable file1.cpp file2.cpp ...
```

Along with the .cpp and .h files that you downloaded with MAIDENiso, you will notice two additional files: runme.bat and Makefile. These files contain the full instruction to compile MAIDENiso, you use one or the other depending on the Operating System of your machine. Makefile is used for UNIX (MacOSX and UBUNTU) while runme.bat is used in Windows:

- Windows: From the command line, go to the folder where you downloaded all the MAIDENiso files. Then run:

  ```
  runme.bat
  ```

  This will run the commands inside runme.bat. If the compilation succeeds, an executable file MAIDEN.exe will be created.

- UNIX: From the command line, go to the folder where you downloaded all the MAIDENiso files. Then run the command "make". This will automatically look for a file called "Makefile" or "makefile" (as the one included in MAIDENiso) and run it. If the compilation succeeds, an executable file called "a.out" will be created.

Alternatively, you can perform the compilation from within an Integrated Development Environment (IDE). This is an editor program, with utilities that help coding and visualizing code. Examples include Code::Blocks, Dev-C++, or XCode (for MacOSX). Having an IDE will make easy to define a project to include all the MAIDENiso files, and automatically create and run the appropriate Makefile.

Regardless of how you do it, you will obtain an executable (binary) file. This will be the MAIDENiso executable.

Chapter 3: Running MAIDENiso

3.1 Inputs

Once the executable exists, MAIDENiso can be run. The runs happen within the simulation directory/folder, which is always the (current) working directory. MAIDENiso is executed by calling the executable file, which can be done remotely (the executable does not need to be inside the simulation folder). Once the executable is called, it will look for a set of input files within the working directory. These files must have these exact names, and their presence is absolutely necessary to run MAIDENiso:

- **"inmet.txt"**: The input meteorology file. See subsection 3.1.1.
- **"inpar2.txt"**: The input parameter file. See subsection 3.1.2.

In MAIDENiso v4 we introduced an **optional** file (whose presence can be detected by MAIDENiso, but nothing bad happens if it cannot be found):

- **"exportflags.txt"**: This file specifies which output files will be exported, using a series of flags (1 for export, 0 for no export) for each of the possible output files. MAIDENiso expects these flags to be in a particular order (we typically add a comment with the name of the flag, but MAIDENiso does not read this, so do not change the order of the lines). If this file is not found, MAIDENiso will assume that all outputs should be exported. Writing all the outputs takes as much time as the simulation process itself, therefore consider using this file to export only the outputs of interest and saving a lot of runtime.

3.1.1 Inmet file

The input meteorology file "inmet.txt" contains several columns standing for different meteorological fields. Previous to MAIDENiso v4, a global integer called meteo_site (defined in Constants.h) determined what fields were expected to be found in the inmet file. Posterior versions use of the header of the inmet file to tell which are the fields to be read from it. The possible headers are:

- **year-day-tmax-tmin-rad-prec-rhavg-wind-CO2**.
- **year-day-tmax-tmin-prec-CO2**.
- **year-day-tmax-tmin-prec-CO2-d18Op-d18Ov**.
- **year-day-tmax-tmin-prec-CO2-d13C**.
- **QPFFN-XXXX-XXXX-Obs-Adjusted-No-RIEN**. This is a relic from initial development phases, the name coming from the a standardized system of naming variables in French meteorological stations. It is functionally equal to year-day-tmax-tmin-prec-CO2-d13C.

Each of these headers identifies the variables that are supposed to follow the header, each variable as a column. These variables are described in Table 3.1.

VARIABLE	UNITS	DESCRIPTION
year	year	Year
day	day	Day of the year
tmax	°C	Atmospheric maximum temperature
tmin	°C	Atmospheric minimum temperature
rad	W m−2	Daily radiation
prec	cm	Precipitation
rhavg	%	Relative humidity
wind	m s^{-1}	Wind speed
CO2	ppm	Atmospheric concentration of CO_2
d18Op	‰	Precipitation $\delta^{18}O$
d18Ov	‰	Atmospheric water vapor $\delta^{18}O$
d13C	‰	Atmospheric $\delta^{13}C$

Table 3.1: Possible fields in the input meteorology file.

Note that MAIDENiso does not accept leap years, and no gap may exist in the meteorological data, which must have 365 days every year, for every single year of the run. It must also match exactly the **run** years specified in the input parameter file by the parameters *year*0 and *yearn*, which specify the first and last year of the run, respectively (see subsection 3.1.2).

3.1.2 Inpar file

The input parameter file is called "inpar2.txt", the name including a "2" as a relic from development. It contains a number of rows (121 as for the current date) defining the values of the model parameters. MAIDENiso expects these parameters to be in a specific order, as it does not read anything from each row except for the first numerical value (separated by space/tab). This allows to write anything after the value of each parameter, for instance the name of the parameter.

It is important to note that not all of these parameters written inside the inpar file are used, depending on what model configuration we are using. Many of the parameters are simply site parameters that define the geographical location of the site, the simulation environment (local characteristics such as soil composition), and the time frame of the simulation (the period for which outputs will be written) and the run. Other parameters control the simulation within specific modules or functions (e.g. photosynthesis or isotopes). This means that not all parameters are used, for instance $\delta^{18}O$ calculations are deactivated by having the parameter "d18O_flag" set to 0, and therefore all the other parameters controlling $\delta^{18}O$ calculations will not be used.

Given the high number of parameters in the inpar file, we do not include a table of parameters in this document. Instead, a file called "Parameters.xlsx" includes a detailed description of all the parameters, including their units, initialization values, symbols used in the code and the parameter file, and the order in which the parameters were written by the current and previous versions of the code. Here we will only describe the parameters that control the time, seen in Table 3.2. These describe the start and end of the **simulation** and the **run**, concepts we explain in section 4.3.

VARIABLE	UNITS	DESCRIPTION
$ndays$	days	Number of days of the run
$year0$	year	Starting year of the run
$yearn$	year	End year of the run
$year0\_sim$	year	Starting year of the simulation
$yearn\_sim$	year	End year of the simulation

Table 3.2: Time parameters in the input parameters file.

The parameter $ndays$ is arguably the most important one, as it is directly used for the model to know just how much memory should allocate to all the variables. The parameters in Table 3.2 have two constrains that the user must take into account to avoid errors:

- The parameters $yearn$ and $yearn\_sim$ should always be equal. $yearn > yearn\_sim$ means the run continues after the simulation, but because only the simulation is exported, it runs for nothing and just wastes time. On the other hand, if $yearn < yearn\_sim$ the model will try to export outputs beyond the end of the run, for which memory was not allocated, and an error will happen.
- The parameter $ndays$ should always be $ndays = 365 \times (yearn - year0 + 1)$, because the run always start on 1st January $year0$ and ends on 31st December $yearn$.

The two constrains means that the five time parameters could just be reduced to three: $year0$, $year0\_sim$ and $yearn$, the other two being a function of these three. The fact that we have five parameters is a relic from development, when it was considered that the model could be used to make runs shorter than a year, though this idea was abandoned

without getting rid of the extra parameters. MAIDENiso v4 still requires the user to specify the five parameters in Table 3.2, so the user must be careful to respect these constrains to avoid errors.

File nomenclature

To keep track of the multitude of input files that can be accumulated for different sites or datasets, it is recommended to keep a standardized naming system. The components of this nomenclature system should be:

- Site: The name or code of the site that MAIDENiso is attempting to simulate. For example, the site of Caniapiscau in the province of Quebec could be called "QC-CAN".
- Dataset: The name of the dataset where the meteorological data has been taken from. For example, if the data was taken from a meteorological station, it could be "stations", or if it was taken from the North American Reanalysis, it should be called "NARR".
- Version: If there is more than one version of this data (some meteorological field could have been corrected, or a parameter changed), you can add a unique code to differentiate different versions. For example, "v1", "v20200731", or "vIH".

As inmet and inpar files are typically coupled, this nomenclature also helps to keep track and avoid mixing them. For example, the input files corresponding to the Caniapiscau site, with data taken from the North American Reanalysis, modified by IH, are identified as "inmet_NARR_QC-CAN_vIH.txt" and "inpar_NARR_QC-CAN_vIH.txt".

Note however that this nomenclature system is completely optional and it is only a suggestion to the user. It is not integrated in the functioning of MAIDENiso.

3.2 Outputs

The output of MAIDENiso is organized into several output files, roughly organized by the module they correspond to. It is also possible for the same output to appear in two different output files. Some of the output variables are yearly, most of them are daily. These output files are:

- **outalloc_d**: Daily allocation values.
- **outalloc_y**: Yearly allocation values.
- **outd2H**: Daily δ^2H values.

- **outd13C_d**: Daily δ^{13}C values.
- **outd13C_y**: Yearly δ^{13}C values.
- **outd18O_d**: Daily δ^{18}O values.
- **outd18O_y**: Yearly δ^{18}O values.
- **outmet**: Daily meteorological values.
- **outphenol**: Daily phenological values.
- **outphoto**: Daily photosynthesis values.
- **outrad**: Daily radiation values.
- **outroot**: Daily root values.
- **outsim**: Yearly summer stem allocation.
- **outsnow**: Daily snow values.
- **outsoil**: Daily soil values.
- **outsoilev**: Daily soil evaporation values.
- **outtemp**: Daily temperature values.
- **outwatbalance**: Yearly water balance values.
- **outwater**: Daily water values.
- **outcustom**: Custom files for user defined outputs.

The output files take a considerable portion of the run time to be written. Because more often than not these outputs are not needed by the user, the unneeded output files should be deactivated. In MAIDENiso v1 and previous versions, this was not possible, but in MAIDENiso v2 and MAIDENiso v3 this could be done by setting to 0 the corresponding flags located inside Maiden.cpp (e.g. outmetPrint=1) and compiling the program again. The optional input file "exportflags.txt" introduced with MAIDENiso v4 can be used to set these flags without needing to modify the code directly. While it is a good idea to only export the outputs you need, it is recommended to do a test run with all output files active before launching a large series of runs. This should be done to quickly check for flagrant problems in the output files (like missing outputs or "nan" values), or to run general diagnostics on the output with an external script.

A detailed description of the output variables can be found in Appendix A.

Chapter 4: The simulation process

4.1 State and process variables

To understand how MAIDENiso works, it is important to make a distinction between two types of variables:

- **State variables**: Variables that define the state of the system at a single moment in time.
- **Process variables**: Variables that describe transitory quantities associated to a process.

To visualize this, suppose we want to describe how much water there is in the ground at a particular instant. This will be fully described by the variables that describe the amount of liquid water and solid water in every layer, therefore, these are state variables. The variables that describe how much water moves from one layer to another do not describe this, but are transitory quantities that depend on the state variables, and determine the state variables for the next point in time. These are thus process variables.

The set of all state variables conform what we can call "the state of the system". The set of state variables at time t determine the process variables (all except for the model inputs, which are process variables forced into the system) at time t. The set of all process variables determine the state variables at time $t+1$. Thus, the model moves the state of the system from one point in time to the next, passing through the process variables:

$state[t] \rightarrow process[t] \rightarrow state[t+1] ... process[t+n-1] \rightarrow state[t+n]$

This distinction between the two types of variables is important from a theoretical point of view. In a mechanistic model, the future state of the system must depend exclusively on the state of the system in the past, and the forcings (inputs) to the model during the time between the two states. On a practical level, the set of state variables can be taken to fully determine the future states.

In MAIDENiso v3 and MAIDENiso v4, a statistical model is used to determine some

variables of the tree (specifically how much carbon it should allocate to the canopy) based on a spline over the temperature and precipitation of all the meteorological period used as input for the model. While this makes sense from a statistical point of view, it does not make sense mechanistically, as it invalidates the mechanistic rules and allows the tree to "act" based on the future. Therefore, these two versions are not purely mechanistic, but in small part statistic. The substitution of this allocation model for another one that is mechanistically valid can be expected for future versions of the model.

4.2 Initial conditions and steady state

We define the **initial state** $state[0]$ as the state of the system (this is, the values of the state variables) at $t = 0$, the first time-step of the run. Because there are no previous values the initial values are arbitrary and lack any justification.

Because the $state[t + 1]$ is based on $state[t]$, we can not expect $state[t + 1]$ to make sense if $state[t]$ does not. However, when forcing the model with the particular meteorology of one site, the state of the system drifts slowly but surely towards a state more defined by the forcings than the initial state. Therefore, it is possible to cycle one or a few years of meteorological forcing to make a transition towards a state in dynamical quasi-periodic equilibrium, called the **steady state**. This steady state is a "reasonable" state of the system given the meteorological forcings, which we can use as an alternative departing point instead of the initial state.

There is no strict definition to determine that the state of the system has reached the steady state. In practice, we run the model for 10-30 years before we assume that the steady state has been reached. A more strict approach that the user can use would be to monitor all the state variables of the system, and observe that this transition towards a dynamic equilibrium has happened for all of them. It has to be noticed that we approach this dynamical equilibrium asymptotically, and that we need an infinite amount of time to reach it. Thus, an assumption has to be made that at some point in time we are "close enough" to the steady state. This is a fairly standard approach used by many models, including global climate models, when they lack a way of obtaining initial conditions.

4.3 Run, training, and simulation

In order to be accurate, we distinguish three terms when we talk about the periods of time that MAIDENiso is working with, based on the initial and steady states:

- **Run**: It is the whole period of time that MAIDENiso is run for, thus we call it "run".

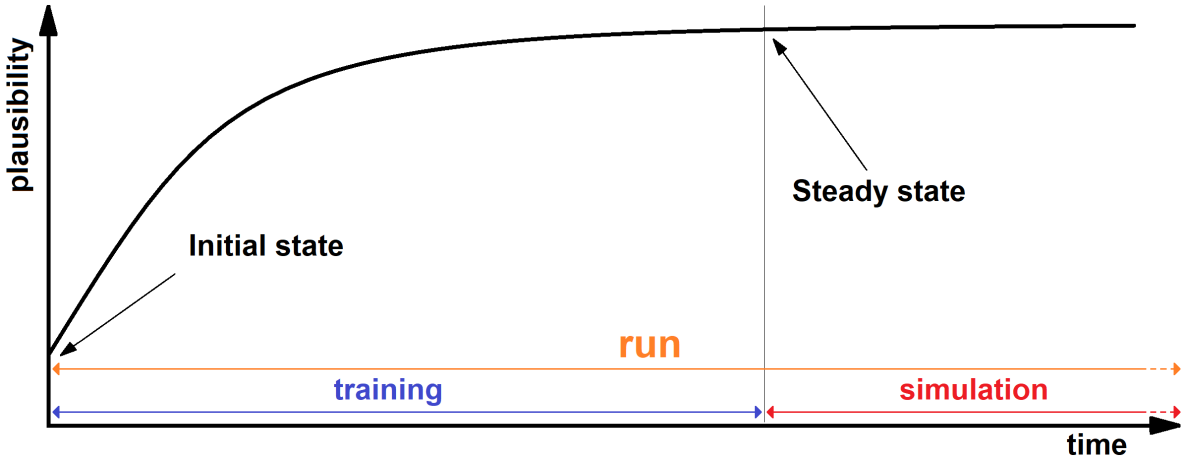

Figure 4.1: Diagram showing the theoretical increase in plausibility between the initial and steady states, which are use a departing points of the training (and the run) and the simulation period, respectively.

- **Training**: It is the period between the initial state and what we assume to be the steady state.
- **Simulation**: It is the period of time that is exported in the outputs, therefore it is the period for which want to analyze the data, and it is supposed to depart from a steady state.

Notice that **run = training + simulation**.

From the practical point of view, in MAIDENiso we specify the beginning and end of the simulation and run periods. The run period is all the time that the model runs, while the simulation period is the period of time that the model exports in the outputs. The training period is just the difference between the starts of run and simulation (having no training if the simulation is said to start at the same time as the run). The endings of the simulation and the run should always be made coincidental, because the simulation ending after the run will try to export outputs that were not run (producing error), while the simulation ending before the run means that the model will keep running after the simulation period (without exporting it). Future model versions may disregard the parameter controlling the end of the simulation to solve this issue.

Also, notice that it is the **run** period that must coincide exactly with the **input meteorology**. The length of the run is used to allocate memory for the input meteorology, therefore if the input meteorology is longer it will run out of space while reading it (producing a segmentation fault), while if it is shorter it will leave the rest of allocated space blank (using default values, either 0 or NA, producing nonsense).

Chapter 5: The C++ code

5.1 Files

MAIDENiso is composed of a number of files. Other than auxiliar files, there are two main types:

- **.cpp** files: These contain the proper C++ code, this is the functions that define MAIDENiso.
- **.h** files: Header files, files written in C++ code that contain declarations used on several .cpp files in MAIDENiso.

When compiling MAIDENiso, the compiler is given instructions to include only the .cpp files in the compilation. The .h files are instead "included" by declaring them at the beginning of the .cpp files. The header files contained in the MAIDENiso code are:

- Maiden_struct.h: Contains the definitions of structured variables.
- Maiden_funct.h: Contains the definitions of functions.
- Constants.h: Contains the definitions of constants.
- Malloc.h: This is not exclusive to MAIDENiso but typically included with the C++ compiler, however MacOSX systems seemed to be unable to find it. To avoid this problem, we include it with the MAIDENiso files.

The .cpp files can include other libraries like stdlib.h, stdio.h, string.h, time.h, math.h, float.h, iostream.h or fstream.h. These are all libraries included with the C++ compiler, than unlike malloc.h do not have trouble being found in any operating system where we tested MAIDENiso.

The .cpp files are:

- **Maiden.cpp**: Contains the main() function and the functions that read and write outputs, allocate and free data, etc.

- **MyFunctions.cpp**: Contains miscellaneous auxiliary functions.
- **allocation_FG_with_GGI_alloc.cpp** (other versions will have this file with alternative names in the fashion of allocation_something.cpp): Contains the module function allocation().
- **meteorology.cpp**: Contains the pre-run functions calc_tair(), calc_prcp(), calc_d18O2H_PV(), snowpack(), calc_srad_humidity_iterative() and the auxiliary functions calc_pet() (to calculate the potential evapotranspiration for aridity corrections), atm_pres() (to calculate the atmospheric pressure as a function of elevation) and pulled_boxcar() (to calculate a moving average of antecedent values in an array).
- **phenology.cpp**: Contains the module function phenology().
- **photosynthesis_AL.cpp**: Contains the module function photosynthesis().
- **radiation.cpp**: Contains the module function radtrans().
- **root.cpp**: Contains the module function root().
- **soih2o_iso.cpp**: Contains the module function soih2o_iso(), and the auxiliary functions tridia() and tridia_tem().
- **soil_evap.cpp**: Contains the module function soil_evap().
- **throughfall_iso.cpp**: Contains the module function throughfall_iso().
- **transpiration.cpp**: Contains the module function transpiration().
- **TRC_isotopes.cpp**: Contains the module function TRC_iso().

5.2 Functions

For clarity, we distinguish several kinds of functions in MAIDENiso depending on what they do:

- The main() function: In any C++ program, the only function that is directly executed.
- Core functions: Functions that are essential to the working of the C++ program, but are not part of the MAIDENiso model itself. For instance, read inputs, write outputs, allocate space, etc.
- Pre-run functions: These perform needed calculations previous to running the main loop.

- Module functions: Contain the different modules of MAIDENiso that are run in the main loop.
- Auxiliary functions: These perform calculations that may appear often in the code. It is cleaner and simpler to write them once as a function and call them instead of writing in the code the same calculation every time it is needed.

Every C++ program operates the same way. The program only executes directly the function called main(), and no other. Every other function has instead to be called from within the function main() to be executed, or from within another function that is being executed.

The function main(), localized in Maiden.cpp, does the following:

1. Read input parameters (information needed to allocate space).
2. Allocate space to all variables.
3. Read input meteorology.
4. Execute the pre-run functions, that calculate the pre-run estimates.
5. Run the main loop of module functions, over *ndays* number of days.
6. Write the outputs.
7. Free space

It is to be noted that there is no initialization module in MAIDENiso v4. The initialization of the state variables is done within the module functions.

The pre-run functions are calculated in this order:

1. **calc_tair()**: Estimates daily air temperatures.
2. **calc_prcp()**: Estimates daily precipitation.
3. **calc_d18O2H_PV()**: Estimate daily snowpack for radiation corrections.
4. **snowpack()**: Estimate daily snowpack.
5. **calc_srad_humidity_iterative()**: Estimate short wave radiation and humidity.

The main loop runs the module functions (that constitute the different modules of MAIDENiso), for every day of the run, in this order:

1. **radtrans()**: Calculates the projected leaf area and absorbed photosynthetic photon flux density for sunlit and shaded fractions of the canopy.

2. **throughfall_iso()**: Calculates the partition of precipitation above ground into different pools due to canopy interception.
3. **soil_evap()**: Calculates evaporation from the snow or the soil.
4. **photosynthesis()**: Calculates photosynthesis, stomatal conductance and leaf respiration of the tree.
5. **transpiration()**: Calculates the amount of water transpirated by the canopy.
6. **soih2o_iso()**: Calculates the hydrology and temperature of the snow and ground layers.
7. **TRC_iso()**: Calculates the tree-ring cellulose isotopes (δ^{18}O and δ^{13}C).
8. **phenology()**: Controls the transition between phenological phases.
9. **allocation()**: Determines how much carbon is distributed to different parts of the tree and autotrophic respiration.
10. **root()**: Calculates the root distribution in the soil layers.

5.3 Variables

Users with some experience in compiled languages like C, C++ or Fortran might already be familiar with the concept of variables: these are containers to store data values. The most basic way to classify variables is by their **type**, for example:

- **int**: Stores integers (whole numbers) without decimals, i.e. 12 or -26.
- **double**: Stores floating point numbers with decimals, i.e. 2.34 or -5.00.
- **char**: Stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.
- **string**: Stores text, such as "Hello World". String values are surrounded by double quotes.
- **bool**: Stores values with two states: true or false.

Another important aspect of variables for which they can be classified is their **scope**:

- **Global variables**: Defined outside all functions and valid for all the code. We avoid using them in MAIDENiso because they may easily create confusion..

- **Local variables**: Defined within a function or block of code and only valid within.
- **Formal parameters**: These are local variables in a function that are assigned values from the function's arguments.

In MAIDENiso global variables are normally avoided in favour of local variables and constants (defined in Constants.h). In contrast to global variables, **constants** are unchangeable and read-only, which makes them tidier to use. These are used for quantities that are always and absolutely constant values, for instance physical or mathematical constants like $\Pi = 3.14159265$.

Local variables are created when a function starts and only valid within the function, being erased after the end of the function. Thus, a function called several times will not remember the values of the local variables of the previous call. A typical example is the use of int variables called i, j and k to use as loop indexes, such as:

```
int i=0;
for (i=0 ; i<ndays ; i++){
// some code
}
```

Code like this will work properly even though every function where there is a loop will likely use a variable of the same name i.

5.3.1 Structured variables

Most variables in MAIDENiso, such as most formal parameters, are **structured variables**. Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure. Unlike an array, a structure can contain many different data types (int, string, bool...). Since we have a great number of variables in MAIDENiso, structured variables are useful to organize the variables by theme and pass them to functions with a single name. The structure of the structured variables is defined in Maiden_struct.h, as this example shows:

```
typedef struct
{
double C_stem_init;
... //more than a hundred of parameters here
int meteo_site;
} param_struct;
```

Then this structure can be used to define a structured variable at the beginning of the main() function:

```
param_struct par;
```

And we can now pass around a single structured variable "par" instead of many individual variables.

5.3.2 Use of pointers in functions

The user may notice that some functions have arguments that are normal variable names (for instance, My_min(double value1, double value2)) while other arguments (in particular structured variables) use pointers (denoted by the "*" symbol in the declaration and definition and the "&" symbol when called), like read_init(site_struct *site, param_struct *par), called as read_init(&site,&par). For users not used to C and C++, this deserves some explanation. Let us take an example for the first kind of function, without pointer arguments:

```
double My_min(double value1, double value2) // test ok 271008
{ return ( (value1 < value2) ? value1 : value2); }
```

Here we see that the My_min function takes value1 and value2, and with the function return() returns value1 if $value1 < value2$, otherwise returning value2. Simple enough.

However, most functions you may add will not follow the classical structure of taking several input, return one output. None of the pre-run and module functions do this, instead they take pointers to structured variables as arguments (inputs) and change these structured variables, using the return() function only to return a logical value (1 or 0) to indicate that the function finished successfully.

There is a reason the functions use pointers to variables instead of the variables themselves: The arguments of the functions become formal parameters, which the functions cannot change. These formal parameters act as local variables, and any change the function does to them will not affect the original variable that was fed as argument to the function. For example, if we have a function that adds 1 to the input variable:

```
double Myfunction1(double myvar)
{myvar += 1;}
```

and in our code we run:

```
thisvar = 16;
Myfunction1(thisvar);
```

the variable "thisvar" will still have a value 16. In this particular example, an easy solution would be to use return() to return the modified value:

```
double Myfunction2(double myvar)
{myvar += 1;
return(myvar)}
```

and running this code:

```
thisvar = 16;
thisvar = Myfunction2(thisvar);
```

Now "thisvar" will have a value of 17. However, we can only use return on a single object. Unless we create an extremely complicated object that contains all the arguments we may change to use it with return(), it is not realistic to use this approach. Instead, we use pointers.

Using a pointer as argument is a good way of changing a variable while still being able to pass it as argument to a function. We pass (point to) the memory address where an object begins, then we change the contents of that section of the memory within the function. When the function ends, the content of the object remains changed. Without the pointer, the function is creating a new variable in other part of the memory, changing that, and scraping it after it is done.

5.4 Modifying the code

New users may have the idea of introducing their own modifications to the MAIDENiso code and test it for their own objectives. Here we provide a short manual for users not fully experienced with C++ (a basic knowledge is assumed for users that are trying to modify the code) on how to do it.

5.4.1 Best practices

When introducing a modification to the code, try to be organized an methodical by following these recommendations:

- Try to maintain the style used in the rest of the code. It helps visualization and understanding of the code for other users.
- Keep a log (an external text file) of your modifications.
- Document your modifications in the code itself by using (short) comments.

- Write in English (previous developers wrote code and comments in their respective languages, resulting in a melange of English, French, Spanish, and Italian).
- Keep a safe and organized archive of your (working and stand-alone) modified versions, so you can easily roll back if your latest modification fails, or you lose your work because of software/hardware failure. You can do this easily with the version control software Git (`https://git-scm.com`).
- Name your modified versions after the parent version, e.g. if you modify version_X, call the new version something like version_X_mod1.
- Make modifications one step at a time. For instance, if you want to modify two modules, first modify only one into version_X_mod1, test it, and then modify the second module into version_X_mod1_mod2.
- Test your version while comparing it to its parent version. It can help visualize flagrant differences between the two versions that perhaps you did not expect.

5.4.2 Adding an element

5.4.2.1 Adding a global variable

Adding a global variable is easy. You just need to declare it in "Maiden.cpp", before the main() function, by adding a line like:

```
int global_variable;
```

5.4.2.2 Adding a constant

You can define a new constant in "Constants.h" by adding a line like:

```
#define MYCONSTANT 1234.5 //comment explaining what this is
```

Note here that we do not need to declare if the constant is an integer or a float, because C++ does not need to know that to assign space to it. The constant is what it is.

Know that it is possible to declare constants whose value will be determined externally. One example would be the type of tree leaf, which is constant through the run but we can define from the parameter file. This is done by having in "Constants.h" the line

```
extern int exp_site;
```

and in the function read_init() that reads the input parameter file, we read the parameter exp_site (as part of the structured variable called "par") and assign it to the constant exp_site:

```
fscanf(inpar,"%i%*[^\n]",&par->exp_site);
exp_site = par->exp_site; // 1: Deciduous ; 2: Evergreen
```

Constants, just like global variables, can be used throughout the code without needing to declare them inside every function you use them in, hence their utility. To identify constants in the code, we typically write them with capital letters (unless they are externally defined constants).

5.4.2.3 Adding a local variable

Local variables are tidier that global variables, because you do not have to worry about having declared something else of the same name somewhere else in the code. These are used only inside the function they are declared, and you can have local variables of the same name in other functions: they will have nothing to do with each other, and will not interfere. These are simply declared at the beginning of a function by writing first the type (integer, float, character, logical...) and name.

```
int function(){
int integer_variable;
double floating_variable=1.4;
integer integer_array[6];
}
```

In this example, the function has declared one integer variable, one floating variable (write "double" to allocate double decimal digits as compared to "float") and one integer array of length 6 (in C++, elements of an array of n elements are numbered 0 to n-1). Variables can be initialized in the declaration itself ("floating variable=1.4"), doing it there or somewhere after depends on tidiness and convenience.

5.4.2.4 Adding a structured variable

When adding a new structured variable (a structure of variables), first you should decide on what variables it should contain, this is what type of structure has. The structure you want may already respond to an existing one, which are defined in the library "Maiden_struct.h". If you want a new structure that does not correspond to an existing one, you must define the type in "Maiden_struct.h". You can define a new structure (in this example we call it "mystructure") by adding this code (adapted to your purposes) to the file:

```
typedef struct
{
int integer_variable
double float_variable
} mystructure_struct;
```

Additionally,

With the structure type defined, you have to declare the new structure in the function main() (found in Maiden.cpp). Definition of variables is done immediately at the start of main(), where you will see the list of declarations of structured variables. Add your new structure (called "mystructvar" in the example) at the end of the list by adding:

```
mystructure_struct mystructvar
```

The structure type only needs to be defined once in "Maiden_struct.h", but note that the same structure can be use for several structured variables:

```
mystructure_struct structure1
mystructure_struct structure2
```

Also note that to use the structure in any function that is not main(), you need to add it to the arguments of the function. This means modifying code in several places:

- In the declaration of the function in "Maiden_funct.h", you must add the structure type of the structure plus the internal name it will use inside the function (which can be different than the name you defined it in main(), but in practice we always use the same name to avoid confusion). If you had a function

  ```
  int funct(oldvariables)
  ```

  now it should look like:

  ```
  int funct(oldvariables, const mystructure_struct *mystructvar)
  ```

  Notice that the structured variable is introduced here with the symbol "*", indicating that is a pointer.

- Whenever you call the function, be in main() or within any other function, add the name of the structured variable (the one you declared in main if in main(), or if in another function the name it uses inside this function) to the list of arguments, **in the same order as in the declaration of the function**. If you had

```
funct(oldvariables)
```

now it should be

```
funct(oldvars, &mystructvar)
```

Notice that the structured variable is introduced here with the symbol "&", indicating that we are passing a reference to an object (a pointer) rather than the object itself.

- In the definition of the function, you will have to add the structure type of the structure plus the internal name it will use inside the function, also **in the same order as in the declaration of the function**. If you had

```
int funct(oldvariables)
```

now it will look like:

```
int funct(oldvariables, const mystructure_struct *mystructvar)
```

Notice that the structured variable is introduced here with the symbol "*", indicating that is a pointer.

5.4.2.5 Adding a variable to a structure

If you want to add a new variable (let us call it "newvar") to an existing structure (let us call it "structure_struct"), make sure to do everything in this checklist:

- **Declare the variable**. Go to the structure you want to add it to, in "Maiden_struct.h", and add it to the list. In front, you must add the type of variable it is. For instance, if it is a float, add:

```
double newvar;        //comment explaining what newvar is
```

- **Allocate space for the variable**. Suppose the variable will define some kind of value that changes daily, therefore you want to to define a value for each day of the run. You will have to do so for every structured variable that is using the structure you are adding your variable to (let us suppose you only have one structured variable "structvar" using the structure "structure_struct"). Then you will have to add this line inside the function data_alloc() (in "Maiden.cpp"):

```
if (ok && !(structvar->newvar = (double*) malloc(ndays * sizeof(double))))
{printf("Error allocating (structvar->newvar)\n");ok=0;}
```

The function "malloc" allocates the space, we are simply telling it exactly how much space we want. We are allocating the space needed for a double variable, times the number of elements we have for this variable (the number of days), which will be an array. We are also printing an error message in case the allocation fails. **Important:** Notice that sometimes you must use ndays+1 instead of ndays, this can be the case for state variables (see section 4.1). For instance, if we run the model for ndays=1 day, we will have one initial value and one final value, this is ndays+1.

- **Initialize the variable**. If the variable is a state variable, or something depends on the value of this variable during the previous timestep, you will need to initialize the values of the variable. You do that in data_alloc() by looking for the right initialization loop (for years, for days, for number of soil layers...) and adding this line:

  ```
  structvar->newvar[m]=0;
  ```

  where m is the loop index.

- **Free space after the end**. To avoid running out of memory, the last thing MAIDENiso does is to liberate the space allocated to the variables. Go to the function data_free() (in "Maiden.cpp") and add this line:

  ```
  free(structvar->newvar);
  ```

5.4.2.6 Adding a function

To add a function, you must first declare it in "Maiden_funct.h", specifying the arguments that the function will take, separated by commas. To specify an argument, you must write the type of argument it is, followed by the internal name (a local variable will be created responding to that name). For instance,

```
double My_min(double value1, double value2); // PAD 161008
```

declares that a function called My_min will take two arguments of type double, which will be treated inside the function as local variables called value1 and value2. You must also declare which type of number the function will return (int, double, void...) in front of the name of the function.

Then you must define the function inside one of the .cpp files included in the compilation instructions of MAIDENiso. If you define it in a new file, it will be necessary to change the instructions to the compiler to include this file. Instead, try to avoid this by adding the function to whichever of the existing files that has the most appropriate

name (for instance, a new root function should be added to "root.cpp"), using "Myfunctions.cpp" for miscellaneous functions that do not fit elsewhere. This definition must start the same way that the declaration, then followed by what the function actually does, and close with the return function giving something back (it can be the result of a simple calculation, or a "0" indicating that the function has finished successfully). In our example:

```
double My_min(double value1, double value2) // test ok 271008
{ return ( (value1 < value2) ? value1 : value2); }
```

Then calling the function in the code, make sure that the arguments are given in the right order the function expects them to be in, that the type of the arguments match the type of the variables used as arguments, and that the type of the variable fits the argument to which it is assigned:

```
double a=1.2;
double b=1.5;
double c;
c = My_min(a,b)
```

Note that when using pointers as arguments, these must appear in the declaration and definition of the function with the "*" symbol:

```
int read_init(site_struct *site, param_struct *par);
```

However when calling the function in the code, the pointer argument must be passed using the "&" symbol:

```
read_init(&site,&par);
```

Chapter 6: Troubleshooting

This chapter addresses the known and most common issues encountered when running MAIDENiso. If your trouble does not fit into any of those described here, please report to the developers so this guide can be updated.

6.1 Segmentation fault

A segmentation fault is the least informative failure that can happen, because typically no other information is given other than a segmentation fault has happened. However, these are the main causes that can be identified:

- Not enough memory could be allocated to the job: This can happen when submitting jobs in a cluster, where you have to specify the memory allocated to a job or go with a default that can be too low. If the run is long, the program may have to allocate more memory than it is available. The failure will therefore happen in the function data_alloc() before the run even starts.
- Trying to write out of allocated space: This can happen if a variable was not defined or allocated properly (therefore being too short), or if it was but bad indexing tried to write outside of the allocated space (for instance, trying to write in position 1001 of an array of length 1000). This kind of error is typical after modifying the code, and will be due to sloppy coding. Another possibility is that the parameter "ndays" was to low (it should be 365 times the number of years in the run, which is "yearn" - "year0" + 1).

6.2 NaN or strange values in the output files

Sometimes MAIDENiso will run without apparent error, but looking at the outputs you will find some that have NaN (not a number) values or completely absurd numerical values. This kind of error is relatively easy to trace, as observing the daily outputs can pinpoint the exact moment where the issue happens. It is also the most tedious error

to trace, as it will likely require a lot of investigation to point out the exact point in the code where the error happened. Notice that the fields "year" and "day" should not be affected by this, and their values should not show anything out of the ordinary. If they are, the problem is of a different nature (see "Messed-up time fields in output files").

In this kind of error, it is typical for the program to behave normally until a given day, which should be reflected in the output files. If a period of training is included in the run, it is possible that the output files show the strange values from the first recorded day, meaning the error happened during the training. Change the parameter "year0_sim" to be equal to "year0", thus integrating all the training in the simulation, but most importantly including it in the outputs, which will allow you to see which was the first day when strange values started to occur.

If you have made any change to the code, this is extremely likely to be the cause of the error. Double-check these changes to make sure no divide-by-zero or other easy mistakes were made. Otherwise, the best solution is to execute the program line-by-line at the day where the error was located (or as tight as you could locate the error). Some developer programs allow to do this easily, but most developers will have to fill the code with pauses and prints.

Another possibility is that the error is caused by the original code, but the model was never run before under some particular circumstances that cause the error, and was therefore never debugged. An example of this happened when using the model for the first time at a latitude high enough that some winter days had no daylight, producing a divide-by-zero error when dividing by the variable daylength. In this case, your must contact the developer team identifying the error, so that the debugging of this error will be included in subsequent versions of MAIDENiso.

6.3 Messed-up time fields in output files

Because the time fields (year and day) in the output files are exactly the values that were read from the inmet file, these fields being wrong in the output indicates that the problem lies with the inmet file. Most likely mistakes are:

- Wrong line terminators. This can happen when creating an inmet file in a DOS system and trying to use it in a UNIX environment. You can check this with the command "file inmet.txt" in Unix, which should show "ASCII text". If instead it shows "ASCII text, with CRLF line terminators", then you must use the command "dos2unix inmet.txt" to correct the line terminators.

- The number of meteorological fields does not correspond to the expectation. For instance, if the program expects 7 fields (including year and day) but the inmet contains 6, the 7th field of the first day will be mistakenly read from the 1st field of the second day, and so on.

- The header is wrong. The first line of the inmet file is the header. If it is not one of the accepted values, or if the header is missing, the program will not read the inmet file. Take into account that inmet files produced by some machines (like Windows) can automatically create end-of-line (EOL) characters that will produce trouble when using other machines (like Linux). Because these EOL characters do not show (unless you specify it), what in appearance is a perfectly fine inmet file will produce trouble. You can check the EOL easily, for instance with advanced text editors like Notepad++, or using the "file inmet.txt" command in Linux.

Part II

Technical Description

Chapter 7: Introduction

7.1 Model history

The original version of the model, MAIDEN [Misson, 2004], was specifically designed to improve the interpretation of tree-ring proxies based on our knowledge about eco-physiological processes and relationships between climate and tree growth. The isotope-enabled version, MAIDENiso [Danis et al., 2012], incorporates calculations of the stable isotopic composition of oxygen (δ^{18}O) and carbon (δ^{13}C) in the different components of the tree. MAIDEN was originally created for tree species in Mediterranean climates, and it has been optimized for *Quercus petraea* (Matt.) Liebl. and 12 Mediterranean species [Misson, 2004, Gaucherel et al., 2008, Boucher et al., 2014, Gea-Izquierdo et al., 2015]. Since then, the phenology and physiological processes have been adapted to simulate tree radial growth in boreal northeastern American forests [Gennaretti et al., 2017] and used to simulate tree-ring cellulose δ^{18}O in boreal and temperate forests of eastern Canada and southern South America [Lavergne et al., 2017]. The latest version of the model [Hermoso de Mendoza et al., 2022] incorporates a thermal module and an update to the hydrological module to incorporate a snow layer and ice layers for the soil and the canopy.

While this is the first technical description of MAIDENiso, a number of versions have been developed and published during the years. The corresponding papers can be consulted for information on those versions:

- MAIDEN [Misson, 2004]. The original model.
- MAIDENiso v1 [Danis et al., 2012]. Isotoped-enabled version of MAIDEN.
- MAIDENiso v2 [Gea-Izquierdo et al., 2015]. This version developed the phenology and allocation modules in relation to climatic drivers.
- MAIDENiso v3 [Gennaretti et al., 2017]. This version adapts the model to boreal species, the most important evolution in this regard being the addition of the acclimation of photosynthesis to temperature.

- MAIDENiso v4 [Hermoso de Mendoza et al., 2022]. **The current version** described in this document. This version incorporates thermal calculations and updates the hydrology with snow and ice.

As a consequence of its development, MAIDENiso currently makes a difference between "boreal" (energy-limited) and "mediterranean" (water-limited) trees, a difference that has to be explicitly specified by the user, and results in the use of different equations and parameters throughout the model. Ideally, the model should be able to automatically identify the site as either energy-limited or water-limited, and use a single set of equations that automatically become different depending on the growth-limiting factor of the environment. However, this has not yet been implemented in the current version MAIDENiso v4. Therefore, be aware of the use of "boreal" and "mediterranean" in the technical description, which in a practical sense mean "energy-limited" and "water-limited", respectively.

7.1.1 New developments in MAIDENiso v4

The development of previous versions are described in the relevant paper. MAIDENiso v4 was developed by Hermoso de Mendoza et al. [2022] and here we describe the added features.

MAIDENiso v3 and previous versions used a simple hydrological model that partitions precipitation water into multiple fluxes (Fig. 7.1a). Precipitation is first divided into canopy interception and direct precipitation. Evaporation is applied to the canopy water and the remainder drips to the ground overnight, adding to direct precipitation into throughfall. This throughfall infiltrates into the soil up to a maximum infiltration (determined by a parameter) and the excess is added to runoff, which comes back the next day, adding to that day's throughfall (acting as a surface water storage, though not subject to evaporation). Infiltration is the source of water for the soil layers, while soil water is extracted by root absorption, evaporation from the top layer, and drainage from the bottom layer. The movement of water between soil layers is determined with Darcy's law, as the solution to a tridiagonal system of equations. Soil water movement is calculated hourly, while every other flux in the hydrology model and MAIDENiso is calculated daily.

MAIDENiso v3 includes a "snowpack" function that estimates the growth and shrink of snow as a function of precipitation and atmospheric temperature. This snowpack is used to calculate an albedo that is used in the calculations of radiation. This however is not a real snow model, and it does not interact with hydrology in any way. The precipitation used to simulate the growth of the snowpack is not subtracted from the throughfall in the hydrological model, and the melting of the snowpack does not release water on the ground. In fact, MAIDENiso v3 does not consider the possibility of snowfall or soil ice, and all hydrology works identically independently of atmospheric temperature.

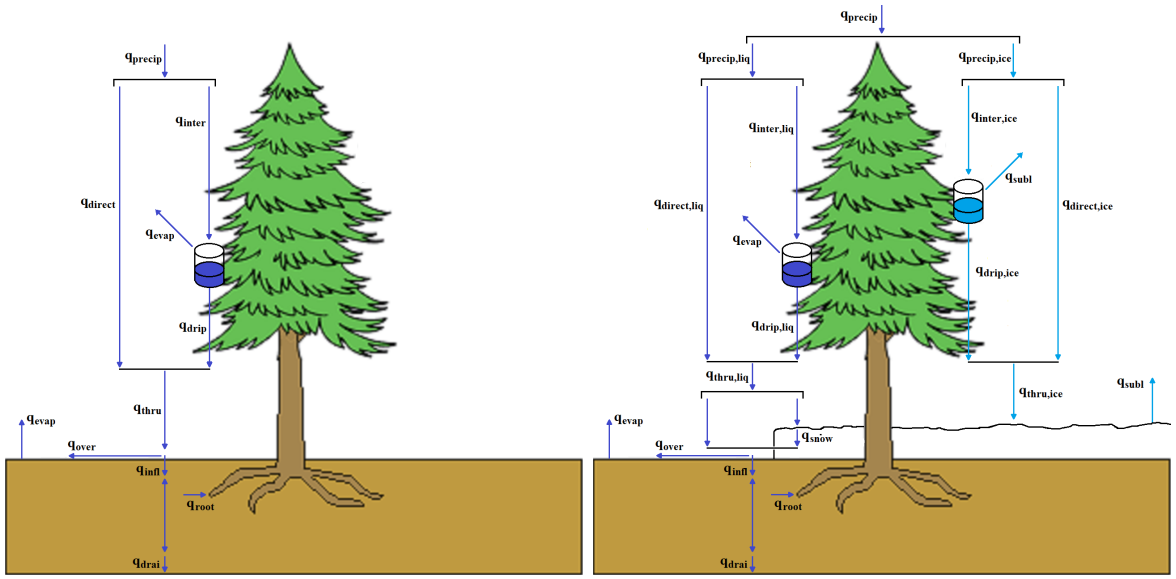

Figure 7.1: Main hydrological fluxes in MAIDENiso v3 (left) and MAIDENiso v4 (right).

The new MAIDENiso v4 expands the hydrological model of the previous versions by adding a snow and ice component (Fig. 7.1b). Precipitation is now divided into rain and snowfall, and the fluxes in which rainfall is divided before reaching the ground have been duplicated for snow. Snowfall is divided into canopy interception and direct snowfall. Sublimation is applied to canopy snow, which stays overnight in the canopy, but can start dripping to the ground if temperatures exceed a given threshold. Canopy snow drip and direct snowfall are added into snow throughfall, which is added directly to the new snow layer. This snow layer grows from snow throughfall and decreases from snow sublimation, but also can start thawing when temperatures rise above the freezing point, creating liquid water within the snow layer.

The fluxes of liquid water have also been modified to work with the new snow system. A portion of the liquid throughfall, determined by the fraction of the ground covered by snow, hits the snow layer instead of the soil, adding to the snow storage of liquid water. This water can percolate through the snow layer, adding to the portion of liquid throughfall that hits the ground directly. This water is used to calculate infiltration and runoff, but in contrast to the previous versions, infiltration is now calculated with the hydrological properties of the top soil layer instead of being a parameter, while runoff now leaves the system permanently. Snow water that exceeds the infiltration capacity can remain in the snow layer if there is space for it, otherwise the excess is added to runoff. The soil water model now accounts for ice content within each layer, which occupies part of the available porous space and decreases the hydraulic conductivity of the layer.

Finally, to allow the model to work with ice and snow, phase transitions between solid and liquid water are calculated for the snow layer and the soil layers. This in turn

required to implement a thermal conduction model to calculate the energy fluxes in these layers. This model is constrained at the top of the snow/soil column by the interaction with the atmosphere (which accounts for shortwave radiation, longwave radiation and sensible heat flux) and at the bottom by a constant heat flux (taken as the crustal heat flux value in the region). The thermal module then solves a tridiagonal system of equations to obtain the layer temperatures at any time-step, and uses the excess/deficit of energy above/below the fusion point to estimate melting/freezing of ice/water. In addition to soil water movement, snow water movement, thermal conduction and phase transitions are also calculated in an hourly basis.

The snowpack model used for the purposes of the radiation calculations is still used in MAIDENiso v4. Radiation is calculated for every day in precedence to the main loop of daily time-step. Therefore, it is not possible to substitute the snowpack with the new snow model unless the radiation module is moved inside the main loop. There is no technical reason why this could not be done, but it could potentially increase the execution time of MAIDENiso, and therefore this decision should not be taken lightly.

7.2 Time notation

The equations described in this document characterize a multitude of quantities used in MAIDENiso, many of which are defined for a particular time interval. To avoid confusion and a possible overload of subindexes, it is convenient to define the notation we follow to indicate that a particular quantity is defined as it is for a given time. In general, we use the angle brackets [] to denote the time for which the quantity is defined. As an example, the expression:

$$X[i] = X[i-1] + Y \quad ,$$

indicates that the quantity X at the time step i is defined as the quantity X at the previous time step $i-1$ plus another function Y.

There are different time intervals for which a quantity can be defined. In MAIDENiso, most quantities are defined daily, while a few are defined yearly. There exist as well quantities defined hourly (in the case of the soil/snow hydrology) and quantities that are defined only for specific days of the year (DOY). These different time intervals are denoted with different symbols, and are defined for different numerical intervals:

- Year of the simulation y. This is an integer that runs in the interval $[1, X]$, X being the number of years defined for the simulation.
- Day of the simulation i. This is an integer that runs in the interval $[1, 365 \cdot X]$.
- Day of the year DOY. This is an integer that runs in the interval $[1, 365]$, and is a function of i as:

$$\mathrm{DOY}[i] = i - 365 \cdot \mathrm{integer}(\frac{i}{365}) \quad .$$

To avoid making the notation tedious, most of the time **we will generally not write explicitly the time dependency of a quantity**, i.e. we will write X instead of $X[i]$. We will make the time dependency explicit, however, in cases when we want to remark this dependency, or when we want to indicate that some quantity defined at a time has a dependency on another quantity defined at a different time. Most notably, the dependency will be made explicit in the definition of a quantity, in the left side of the expression. Some examples:

- $X[i] = Y + C \cdot Z$: We define X, thus we explicitly write its dependence on i on the left. The quantities Y, C and Z are defined somewhere else, thus we do not write their dependencies here and it is implicit that it is the same as in their definition.
- $X[i] = Y[i-1] + C \cdot Z$: X depends on the Y from the previous time interval, therefore we write the dependency of Y explicitly.

We can also use the notation $X[i + \frac{1}{2}]$ to indicate an intermediate step in the calculation of the quantity X. However, this is just a way to facilitate the description. Quantities are only recorded at integer steps, and not in between.

Chapter 8: Atmosphere

8.1 Input meteorology

In MAIDENiso, we simulate the growth of a tree in a particular location, called the "site". The model demands a meteorological input, which should ideally correspond to meteorological measurements taken at the same site, but this is usually not the case. The model designates as the meteorological "base" the place where the input meteorology is taken from, and corrects it based on the differences between the coordinates of the site and the base.

8.1.1 Temperature

The temperature of the atmosphere is introduced as an input to MAIDENiso, via a daily minimum temperature $\mathrm{T}_{\mathrm{min,base}}[i]$ and a daily maximum temperature $\mathrm{T}_{\mathrm{max,base}}[i]$, corresponding to the meteorological base. These are corrected for the site, based on the difference between the elevations of the base and the site, H_{base} and H_{site} (km). To simplify the notation, temperatures at the site ($\mathrm{T}_{\mathrm{X,site}}$, X being whatever) are denoted without the "site" suffix (T_{X}).

$$\Delta H = H_{site} - H_{base} \quad , \tag{8.1}$$

$$\mathrm{T}_{\mathrm{min}}[i] \equiv \mathrm{T}_{\mathrm{min,site}}[i] = \mathrm{T}_{\mathrm{min,base}}[i] + \Delta H \cdot \mathrm{lr}_{\mathrm{min}} \quad , \tag{8.2}$$

$$\mathrm{T}_{\mathrm{max}}[i] \equiv \mathrm{T}_{\mathrm{max,site}}[i] = \mathrm{T}_{\mathrm{max,base}}[i] + \Delta H \cdot \mathrm{lr}_{\mathrm{max}} \quad , \tag{8.3}$$

where $\mathrm{lr}_{\mathrm{min}} = -3°\mathrm{C/km}$ and $\mathrm{lr}_{\mathrm{max}} = -6°\mathrm{C/km}$ are the lapse rate corrections for the minimum and maximum temperatures.

The average temperature of the atmosphere each day is calculated as:

$$\mathrm{T}_{\mathrm{avg}}[i] = \frac{\mathrm{T}_{\mathrm{max}}[i] + \mathrm{T}_{\mathrm{min}}[i]}{2} \quad , \tag{8.4}$$

while the daylight temperature (used for photosynthetic calculations) is:

$$\mathrm{T}_{\mathrm{day}}[i] = (\mathrm{T}_{\mathrm{max}}[i] - \mathrm{T}_{\mathrm{avg}}[i]) \cdot \mathrm{T}_{\mathrm{day,COEF}} + \mathrm{T}_{\mathrm{avg}}[i] \quad , \tag{8.5}$$

where $T_{day,COEF} = 0.45$ is the daylight temperature coefficient.

8.1.2 Precipitation

The precipitation from the meteorological base $q_{precip,base}$ is also corrected, using the isohyet of the base and the site $isoh_{base}$ and $isoh_{site}$:

$$q_{precip}[i] \equiv q_{precip,site}[i] = q_{precip,base}[i] \frac{isoh_{site}}{isoh_{base}} \quad . \tag{8.6}$$

From here, precipitation follows a complex path. It is first divided in solid (snowfall) an liquid (rainfall) fractions depending on atmospheric temperature. Then, each of these is divided into fractions reaching the ground directly or intercepted by the canopy, which is further divided between fractions that evaporate, drip to the ground, or stay in the canopy (in the case of snow). These processes are fully described in Chapter 10.

8.2 Atmospheric pressure

The atmospheric pressure [Iribarne and Godson, 1981, Berberan-Santos et al., 1997] is calculated as:

$$P_{atm} = P_{STD} \cdot P_{ratio} \quad , \tag{8.7}$$

$$P_{ratio} = (1 - \frac{LR_{STD} \cdot h_{site}}{T_{STD}})^{\frac{G_{STD}}{LR_{STD} \cdot \frac{R}{M_A}}} \quad , \tag{8.8}$$

where $P_{STD} = 101325$ Pa is the standard pressure at sea level, $G_{STD} = 9.80665$ m s^{-2} is Earth's gravity at sea level, $T_{STD} = 288.15$ K is the standard temperature at sea level, $LR_{STD} = 0.0065$ K m^{-1} standard temperature lapse rate, h_{site} is the height of a site over sea level, $M_A = 0.0289644$ kg mol^{-1} is the molecular mass of air, and $R = 8.3143$ m^3 Pa mol^{-1} K^{-1} is the ideal gas constant.

8.3 Potential evapotranspiration

The calculation of potential evapotranspiration requires knowledge of the daily average incident shortwave radiation from Eq. (9.31) (rad[i], W m^{-2}). This is used to estimate the net absorbed radiation (rad_{net}, W m^{-2}) as a fraction of rad, This fraction is based on an assumption of an albedo of 0.2 and a ground heat flux of 10% of the absorbed radiation during daylight, which makes $(1 - 0.2) \cdot (1 - 0.1) = 0.72$.

$$rad_{net}[i] = rad \cdot 0.72 \quad . \tag{8.9}$$

On the other hand, independent of radiation, we calculate the latent heat of vaporization ($L*_{vap}$, J kg^{-1}) as a function of the daylight average air temperature $T_{day}[i]$ (in °C).

$$\lambda_{\mathrm{vap,Tday}}[i] = \lambda_{\mathrm{vap},0} - \alpha_{\lambda_{vap}} \cdot \mathrm{T_{day}} \quad , \tag{8.10}$$

where $\lambda_{\mathrm{vap},0} = 2.5023 \cdot 10^6$ J kg^{-1} is the latent heat of vaporization at 0 °C and $\alpha_{\lambda_{vap}} = 2430.54$ J kg^{-1} K^{-1} is the slope of the linear approximation for the dependence of the latent heat of vaporization on temperature.

We also calculate the psychometric parameter, dependent on atmospheric pressure P_{atm}:

$$\gamma[i] = \frac{\mathrm{C_P} \cdot \mathrm{P_{atm}}}{\lambda_{\mathrm{vap,Tday}} \cdot \epsilon_{wa}} \quad , \tag{8.11}$$

where $\epsilon_{wa} = 0.62196351$ is the ratio between the molecular weights of water (M_W = 18.0148 g mol^{-1}) and dry air (M_A = 28.9644 g mol^{-1}), and C_P = 1010.0 J kg^{-1} K^{-1} is the specific heat of air.

Now we estimate the slope of the saturation vapour pressure curve at T_{day}, using a temperature offset of 0.2 K for slope estimate, $T_1 = T_{day} + 0.2$ K and $T_2 = T_{day} - 0.2$ K. We calculate the saturation vapour pressures (PVS) at T_1 and T_2, PVS_1 and PVS_2, using formula from Abbott and Tabony [1985]:

$$\mathrm{PVS_1} = 610.7 \exp(\frac{17.38 \cdot \mathrm{T_1}}{239 + \mathrm{T_1}}) \quad , \tag{8.12}$$

$$\mathrm{PVS_2} = 610.7 \exp(\frac{17.38 \cdot \mathrm{T_2}}{239 + \mathrm{T_2}}) \quad , \tag{8.13}$$

and we use them to calculate the slope s of the PVS vs. temperature curve near T_{day}:

$$s = \frac{\mathrm{PVS_1} - \mathrm{PVS_2}}{\mathrm{T_1} - \mathrm{T_2}} \quad . \tag{8.14}$$

Finally we calculate PET (kg m^{-2} day^{-1}, equivalent to mmwater day^{-1}) using the Priestly-Taylor approximation, with coefficient set at 1.26:

$$\mathrm{PET}[i] = \frac{1.26 \cdot \frac{s}{s+\gamma} \cdot \mathrm{rad_{net}} \cdot \mathrm{dayL}}{\mathrm{L_{vap}}} \quad , \tag{8.15}$$

where dayL is the daytime length from Eq. (9.4).

8.4 Humidity

Atmospheric humidity is calculated from daylight temperature T_{day} and dew temperature T_{dew}. T_{dew} (see Chapter 9) is estimated first as T_{min} and can be corrected iteratively for an arid environment: T_{dew} determines atmospheric vapour pressure, which

determines radiation, which determines potential evapotranspiration, which can be used to correct T_{dew}. Regardless of which estimate of T_{dew} we keep in the end (the first estimate $T_{dew} = T_{min}$ or the correction), the final T_{dew} is used to calculate atmospheric vapour pressure (pva, Pa):

$$\text{pva}[i] = 610.7 \exp(17.38 \cdot \frac{T_{dew}}{239 + T_{dew}}) \quad , \tag{8.16}$$

Vapour pressure at saturation (pvs, Pa) is calculated from a similar formula, but using T_{day} instead of T_{dew}:

$$\text{pvs}[i] = 610.7 \cdot \exp(17.38 \cdot \frac{T_{day}}{239 + T_{day}}) \quad . \tag{8.17}$$

Relative humidity (RH, unitless) and vapour pressure deficit (vpd, Pa) are simply calculated as:

$$\text{RH}[i] = \frac{\text{pva}}{\text{pvs}} \quad , \tag{8.18}$$

$$\text{vpd}[i] = \text{pvs} - \text{pva} \quad , \tag{8.19}$$

8.5 Canopy layer conductance

We calculate the wind's friction velocity (u_*, m s^{-1}) using Monin-Obukhov similarity theory under neutral stability conditions:

$$u_*[i] = \kappa \frac{u}{\log(\frac{z_{ref} - d_0}{z0})} \quad , \tag{8.20}$$

where $\kappa = 0.4$ is the Von Karman constant, $z_{ref} = 17.5$ m is the reference height for the wind, $d_0 = 12.5$ m is the zero plane displacement, and $z_0 = 0.3$ m is the surface roughness.

The boundary layer conductance (g_b, m s^{-1}) [Monteith, 1965] is:

$$g_b[i] = \frac{1}{\frac{6.2}{u_*^{0.67}}} \quad , \tag{8.21}$$

with the boundary layer conductance for water vapour is:

$$g_{bw}[i] = g_b \cdot 1000./M_W \quad , \tag{8.22}$$

where $M_W = 18.0148$ is the molecular weight of water (g mol^{-1}). Similarly, the boundary layer conductance for CO_2 is:

$$g_{bc}[i] = \frac{g_{bw}}{1.37} \quad , \tag{8.23}$$

The canopy aerodynamic conductance (g_a, m s^{-1}), from Monteith [1965]:

$$g_a[i] = \frac{u_*^2}{u} \quad , \tag{8.24}$$

while the canopy aerodynamic conductance for water ($g_{a,w}$, mol m^{-2} s^{-1}) is:

$$g_{a,w}[i] = g_{a,w} \frac{1000\text{Kg m}^{-3}}{\text{M}_\text{W}} \quad . \tag{8.25}$$

It has to be noted that, while these quantities are in theory different in each day i as they ultimately depend on wind speed $u[i]$, wind speed data are in most cases not available daily. Therefore, in practice, these quantities are constant through time in most cases.

Chapter 9: Radiation

Before starting the iterative algorithm between humidity and radiation, all the variables that do not depend on humidity are calculated so they only get done once.

9.1 Transmitance

The initial transmittance tr_1 is corrected for elevation:

$$\mathrm{tr}_1 = {\mathrm{t}_{\mathrm{base}}}^{\mathrm{P}_{\mathrm{ratio}}} \quad , \tag{9.1}$$

where $\mathrm{tr}_{\mathrm{base}} = 0.870$ (unitless) is the maximum instantaneous transmittance at sea level and with dry atmosphere. This transmittance is later corrected for the optimal air mass above it, which requires first to calculate the celestial angles.

For every day of the year DOY (1 to 365, only calculated for one year), we aim to calculate maximum daily total transmittance, potential radiation, and length of day. To do this, we first calculate several quantities that depend on i.

First, the declination is:

$$\mathrm{DECL[DOY]} = \mathrm{DECL}_{\mathrm{min}} \cos((\mathrm{DOY} + \mathrm{DAYSOFF})\frac{2\pi}{365}); \quad , \tag{9.2}$$

where DAYSOFF = 11.25 is julian day offset of the winter solstice and $\mathrm{DECL}_{\mathrm{min}} = -0.4092797$ is the minimum declination in radians (-23.5 degrees).

The hour angle at sunset hss (radians) is calculated as:

$$\mathrm{hss[DOY]} = \arccos(-\frac{\sin(\mathrm{lat})}{\cos(\mathrm{lat})}\frac{\sin(\mathrm{DECL})}{\cos(\mathrm{DECL})}) \quad , \tag{9.3}$$

where lat (degrees North) is the latitude of the site.

This hss is used to calculate the daytime length or daylength (dayL, in seconds) as:

$$\mathrm{dayL[DOY]} = 2\mathrm{hss} \cdot \mathrm{SECPERRAD} \quad , \tag{9.4}$$

where SECPERRAD = 13750.9871 is the number of seconds per radian of hour angle.

The solar constant SC (W m^{-2}) is:

$$\mathrm{SC[DOY]} = 1368 + 45.5\sin((2\pi\frac{\mathrm{DOY}}{365.25}) + 1.7) \quad . \tag{9.5}$$

To calculate the maximum daily total transmittance and the potential radiation, we integrate over small periods of time adding to the whole day, using a sub-daily routine to calculate quantities that depend on the hour angle. Before starting this routine, we calculate several auxiliary quantities bsg1, bsg2 and bsg3 that depend on the day i but not on the hour angle:

$$\mathrm{bsg1[DOY]} = -\sin(\mathrm{slp})\sin(\mathrm{asp})\cos(\mathrm{DECL}) \quad , \tag{9.6}$$

$$\mathrm{bsg2[DOY]} = (-\cos(\mathrm{asp})\sin(\mathrm{slp})\sin(\mathrm{lat}) + \cos(\mathrm{slp})\cos(\mathrm{lat})\cos(\mathrm{DECL}) \quad , \tag{9.7}$$

$$\mathrm{bsg3[DOY]} = (\cos(\mathrm{asp})\sin(\mathrm{slp})\cos(\mathrm{lat}) + cos(\mathrm{slp})\sin(\mathrm{lat}))\sin(\mathrm{DECL}) \quad , \tag{9.8}$$

where asp and spl are respectively the aspect (orientation of the slope) and the inclination of the slope at the site, both of which are parameters of the site.

Sub-daily routine

After calculating these quantities for the day-of-year DOY, a sub-daily routine starts to calculate several quantities at equally time-spaced intervals. This interval is $\mathrm{dt_r} = 600$ s (10 minutes), which is transformed into an hour angle interval $\mathrm{dh} = \frac{\mathrm{dt_r}}{\mathrm{SECPERRAD}}$ to perform calculations as function of the solar hour angle h. This routine runs between sunrise (h=-hss) and sunset (h=+hhs) at increments of dh.

The previous auxiliary quantities independent of the angle hour h (bsg1, bsg1 and bsg1) are now used to calculate the beam-slope angle cbsa:

$$\mathrm{cbsa}[h] = \arccos(\sin(h) * \mathrm{bsg1} + cos(h) * \mathrm{bsg2} + \mathrm{bsg3}) \quad . \tag{9.9}$$

The total over the time-step $\mathrm{dt_r}$ of the extraterrestrial radiation perpendicular to beam, $\mathrm{rad_{per,top}}$ (J) is calculated as (note that is the same for every interval of length $\mathrm{dt_r}$, but dependent on DOY):

$$\mathrm{rad_{per,top}[DOY]} = \mathrm{dt_r} \cdot \mathrm{SC[DOY]} \quad . \tag{9.10}$$

The solar zenith angle depends on both the DOY and h:

$$\mathrm{Zen}[\mathrm{DOY}, h] = \arccos(\frac{\sin(\mathrm{DECL})}{\cos(\mathrm{DECL})}\cos(h) + \frac{\sin(\mathrm{lat})}{\cos(\mathrm{lat})}) \quad . \tag{9.11}$$

If $\cos \mathrm{Zen} > 0$, we calculate the potential radiation for this time interval, for a flat surface, at the top of atmosphere:

$$\mathrm{rad}_{\mathrm{flat,top}}[\mathrm{DOY}, h] = \mathrm{rad}_{\mathrm{per,top}} \cdot \cos \mathrm{Zen} \quad . \tag{9.12}$$

Now we calculate the optical air mass (am, unitless):

$$\mathrm{am}[\mathrm{DOY}, h] = \frac{1}{\cos \mathrm{Zen} + 10^{-7}} \quad , \tag{9.13}$$

which is used to correct the instantaneous transmittance:

$$\mathrm{tr}_2[\mathrm{DOY}, h] = \mathrm{tr}_1^{\mathrm{am}} \quad . \tag{9.14}$$

Note that Zen, am, $\mathrm{rad}_{,\mathrm{flat,top}}$ and tr_2 depend on h. To obtain the total daily transmittance, we sum the instantaneous transmittance for each 10 minutes interval, weighted by potential radiation for a flat surface at top of atmosphere:

$$\mathrm{tr}_{\mathrm{sum}}[\mathrm{DOY}] = \sum_h \mathrm{tr}_2 \cdot \mathrm{rad}_{\mathrm{flat,top}} \quad . \tag{9.15}$$

End of the sub-daily routine

After the sub-daily routine, we calculate maximum daily total transmittance $\mathrm{ttr}_{\mathrm{max}}$ and daylight average flux density for a flat surface $\mathrm{fluxdens}_{\mathrm{flat}}$ and for the slope $\mathrm{fluxdens}_{\mathrm{slope}}$.

$$\mathrm{ttr}_{\mathrm{max}}[\mathrm{DOY}] = \frac{\sum_h \mathrm{tr}_2 \cdot \mathrm{rad}_{\mathrm{flat,top}}}{\sum_h \mathrm{rad}_{\mathrm{flat,top}}} \quad , \tag{9.16}$$

$$\mathrm{fluxdens}_{\mathrm{flat}}[\mathrm{DOY}] = \frac{\sum_h \mathrm{rad}_{\mathrm{flat,top}}}{\sum_h \mathrm{dayL}} \quad , \tag{9.17}$$

$$\mathrm{fluxdens}_{\mathrm{slope}}[\mathrm{DOY}] = \frac{\sum_h \mathrm{rad}_{\mathrm{per,top}} \cdot \mathrm{cbsa}}{\sum_h \mathrm{dayL}} \quad . \tag{9.18}$$

9.2 Sky proportion

The next step is to calculate the sky proportion for diffuse radiation. We use the product of spherical cap defined by average horizon angle and the great-circle truncation of a hemisphere. This factor is independent of i.

First, the average horizon $\mathrm{Hor}_{\mathrm{avg}}$ is calculated as the average between the East and West horizons, $\mathrm{Hor}_{\mathrm{E}}$ and $\mathrm{Hor}_{\mathrm{W}}$, which are parameters of the site.

$$\mathrm{Hor}_{\mathrm{avg}} = \frac{\mathrm{Hor}_{\mathrm{E}} + \mathrm{Hor}_{\mathrm{W}}}{2} \quad . \tag{9.19}$$

If the inclination of the slope at the site exceeds $\mathrm{Hor_{avg}}$, the excess is calculated:

$$\mathrm{spl_{excess}} = \mathrm{spl} - \mathrm{Hor_{avg}} \quad . \tag{9.20}$$

The horizon scalar is then calculated as:

$$\mathrm{Hor_{scalar}} = 1 - \sin(\mathrm{Hor_{avg}}) \quad , \tag{9.21}$$

Now we calculate the slope scalar $\mathrm{spl_{scalar}}$. If $\mathrm{Hor_{avg}} > \pi$ (rad), then $\mathrm{spl_{scalar}} = 0$. Otherwise, it is calculated as:

$$\mathrm{spl_{scalar}} = 1 - (\frac{\mathrm{spl_{excess}}}{\pi - 2\mathrm{Hor_{avg}}}) \geq 0 \quad . \tag{9.22}$$

Finally the sky proportion for diffuse radiation is:

$$\mathrm{sky_{prop}} = \mathrm{Hor_{scalar}} \cdot \mathrm{spl_{scalar}} \quad . \tag{9.23}$$

9.3 Daily Radiation

A series of calculations is used to calculate the daily radiation and the fraction of it that is photosynthetically active. In a nutshell (we will now explain it in detail), the daily radiation is sum of the direct and diffuse radiations plus a snow correction. These are all dependent on daily transmittance, which is affected by three factors: Daily difference between maximum and minimum temperatures (known, as these are direct inputs to the model), whether there has been some precipitation in that day (independent on the quantity), and atmospheric vapour pressure (pva, Pa), which depends directly on dew temperature $\mathrm{T_{dew}}$ (which is not known a priori).

Let us for now suppose we know $\mathrm{T_{dew}}$. Then, we can calculate daily pva using Eq. (8.16). Vapour pressure has an effect on the maximum daily total transmittance, which is corrected to:

$$\mathrm{ttr'_{max}}[i] = \mathrm{ttr_{max}} + \mathrm{A_{base}} \cdot \mathrm{pva} \quad , \tag{9.24}$$

where $\mathrm{A_{base}} = -6.1 \cdot 10^{-5}$ $\mathrm{Pa^{-1}}$ is the vapour pressure effect on transmittance. Note that $\mathrm{ttr_{max}}$ depends on DOY while pva depends on i, and because DOY is a function of i, we say that $\mathrm{ttr'_{max}}$ depends only on i. This allows us to calculate the final daily total transmittance:

$$\mathrm{ttr_{fin}}[i] = \mathrm{ttr'_{max}} \cdot \mathrm{ttr_{frac}} \quad , \tag{9.25}$$

where $\mathrm{ttr_{frac}}$ is the proportion of daily maximum transmittance, calculated as:

$$\mathrm{ttr_{frac}}[i] = \mathrm{R_{scalar}}(1 - 0.9\exp(-b \cdot \mathrm{DTR}^C) \cdot \mathrm{ttr_{fmax}}) \quad , \tag{9.26}$$

$$b[i] = B_0 - B_1 \exp(-B_2 \cdot \mathrm{DTR}_{\mathrm{smooth}}) \quad , \tag{9.27}$$

$$\mathrm{R}_{\mathrm{scalar}}[i] = \begin{cases} 1 & \text{if } q_{precip}[i] = 0 \\ 0.75 & \text{if } q_{precip}[i] > 0 \end{cases} \quad . \tag{9.28}$$

where DTR is the diurnal temperature range, $\mathrm{DTR}[i] = \mathrm{T}_{\max} - \mathrm{T}_{\min}$, and $\mathrm{DTR}_{\mathrm{smooth}}[i]$ is DTR smoothed over a 30 day window. $B_0 = 0.013$, $B_1 = 0.201$, $B_2 = 0.185$ and $C = 1.5$ are dimensionless radiation parameters. The rain scalar $\mathrm{R}_{\mathrm{scalar}}$ is the transmitance correction for rainy days, 0.75 if there is rain in that day, 1 otherwise.

Now we estimate the fraction of radiation that is diffuse $\mathrm{p}_{\mathrm{dif}}$, on an instantaneous basis, from relationship with daily total transmittance in Jones [1993] Fig 2.8, p. 25, and Gates [1980] Fig 6.14, p. 122.

$$\mathrm{p}_{\mathrm{dif}}[i] = -1.25 \cdot \mathrm{ttr}_{\mathrm{fin}} + 1.25 \quad , \tag{9.29}$$

with $\mathrm{p}_{\mathrm{dif}} \in [0, 1]$. The fraction of radiation that is direct is simply:

$$\mathrm{p}_{\mathrm{dir}}[i] = 1 - \mathrm{p}_{\mathrm{dif}} \quad . \tag{9.30}$$

The daily total radiation is estimated as the sum of three components:

$$\mathrm{rad}[i] = \mathrm{rad}_{\mathrm{dir}}[i] + \mathrm{rad}_{\mathrm{dif}}[i] + \mathrm{S}_{\mathrm{cor}}[i] \quad , \tag{9.31}$$

where these components are:

1. $\mathrm{rad}_{\mathrm{dir}}$: The direct radiation arriving during the part of the day when there is direct beam on the slope.
2. $\mathrm{rad}_{\mathrm{dif}}$: The diffuse radiation arriving over the entire dayL (when sun is above ideal horizon).
3. $\mathrm{S}_{\mathrm{cor}}$: The snow correction ($\mathrm{S}_{\mathrm{cor}}$), when snow is present.

The direct radiation is calculated as the product of the average flux density for a slope, the transmittance, and the fraction of direct radiation:

$$\mathrm{rad}_{\mathrm{dir}}[i] = \mathrm{fluxdens}_{\mathrm{slope}} \cdot \mathrm{ttr}_{\mathrm{fin}} \cdot \mathrm{p}_{\mathrm{dir}} \quad . \tag{9.32}$$

The diffuse radiation includes the effect of surface albedo in raising the diffuse radiation for obstructed horizons:

$$\mathrm{rad}_{\mathrm{dif}}[i] = \mathrm{fluxdens}_{\mathrm{flat}} \cdot \mathrm{ttr}_{\mathrm{fin}} \cdot \mathrm{p}_{\mathrm{dif}} \cdot (\mathrm{sky}_{\mathrm{prop}} + \mathrm{DIF}_{\mathrm{ALB}}(1 - \mathrm{sky}_{\mathrm{prop}})) \quad , \tag{9.33}$$

where $\mathrm{DIF}_{\mathrm{ALB}} = 0.6$ (dimensionless) is the diffuse albedo for horizon correction.

The snow correction $\mathrm{S_{cor}}$ (W m^{-2}), with a maximum allowed value of 100 W m^{-2}, is:

$$\mathrm{S_{cor}}[i] = (1.32 + 0.096 \cdot \Delta z_{snow} 10^4) \frac{1}{\mathrm{dayL}} \quad , \tag{9.34}$$

where Δz_{snow} cm is the thickness of the snow layer. However, it is important to note that this is not the same snow layer thickness from section 11.1, which is calculated each day using precipitation and temperature-driven phase transitions. Instead, it is a pre-run estimate of the snow layer thickness based on daily precipitation and temperature:

$$\Delta z_{snow}[i] = q_{precip,snow}[i] - q_{melt}[i] \quad , \tag{9.35}$$

$$q_{melt}[i] = \begin{cases} \mathrm{r_{melt}} \cdot \mathrm{T_{avg}} & \text{if } \mathrm{T_{avg}} \geq 0°\mathrm{C} \\ 0 & \text{if } \mathrm{T_{avg}} \leq 0\ °\mathrm{C} \end{cases} \quad . \tag{9.36}$$

where $\mathrm{r_{melt}} = 0.042$ cm °C d^{-1} snowmelt rate, and of course Δz_{snow} cannot go below 0.

The photosynthetically active radiation (PAR) is calculated as a fraction of the radiation, following Ross [1975]:

$$\mathrm{PAR}[i] = \mathrm{rad} \cdot \left(0.60 + 0.42 \cdot \frac{\frac{\mathrm{p_{dir}}}{\mathrm{p_{dif}}}}{1 + \frac{\mathrm{p_{dir}}}{\mathrm{p_{dif}}}} \right) \quad . \tag{9.37}$$

9.4 Dew temperature

Now, let us remember that we do not know $\mathrm{T_{dew}}$, which we need to input in Eq. (8.16) to obtain all subsequent equations until Eq. (9.37). The way we proceed is to make a first estimate for $\mathrm{T_{dew}}$, calculate Eq. (8.16) and Eqs. (9.24)-(9.37), use the resulting radiation to correct $\mathrm{T_{dew}}$, and calculate Eq. (8.16) and Eqs. (9.24)-(9.37) again.

The first estimate is to assume dew temperature to be the minimum temperature for each day, $\mathrm{T_{dew}}[i] = \mathrm{T_{min}}[i]$, which allows us to obtain a first estimate of the daily average shortwave incident radiation. Using this, we can make a first estimate of daily potential evapotranspiration (PET, section 8.3). The daily PET and the daily precipitation are added to calculate the average annual precipitation ($\overline{q}_{precip,yr}$) and average annual PET ($\overline{\mathrm{PET}}_{\mathrm{yr}}$) at the site, which are compared to decide whether to keep our first estimates or make a correction. If $\overline{\mathrm{PET}}_{\mathrm{yr}}/\overline{q}_{precip,yr} < 2.5$, no correction is applied and our initial assumption $\mathrm{T_{dew}} = \mathrm{T_{min}}$ is considered valid.

However, if $\overline{\mathrm{PET}}_{\mathrm{yr}}/\overline{q}_{precip,yr} \geq 2.5$ the arid correction is applied. For it, we make a second estimate of $\mathrm{T_{dew}}$ using the first estimate of PET:

$$\mathrm{T_{dew}}[i] = \mathrm{T_{min}} \cdot (-0.127 + 1.121 \cdot (1.003 - 1.444r + 12.312r^2 - 32.766r^3) + 0.0006\ \mathrm{DTR}) \quad , \tag{9.38}$$

where $r[i] = \frac{\mathrm{PET}}{q_{precip}}$ is the ratio between PET and precipitation, on a daily basis.

Now we make a second estimate of radiation using the second estimate of T_{dew}, repeating the calculations of Eqs. (8.16) to (9.37). This allows us to obtain a second estimate of PET, which are then used to calculate a third estimate of T_{dew} using Eq. (9.38) again.

Chapter 10: Throughfall

10.1 Precipitation

Precipitation in the model is partitioned into several fluxes before reaching the soil surface. First, it is partitioned into rainfall and snowfall, as a function of atmospheric temperature. Each of these is partitioned into the amount that reaches the soil directly, or direct precipitation, and the amount that is intercepted by the canopy. The intercepted precipitation is further divided into evaporated canopy water, canopy water that drips to the soil (adding to direct precipitation), or in the case of snow, snow that remains in the tree for the next day. A diagram of these fluxes can be seen in Fig. 10.1.

10.1.1 Solid/Liquid precipitation

Daily precipitation $q_{precip}[i]$ is divided into liquid (rainfall) $q_{precip,liq}$ and solid (snowfall) $q_{precip,snow}$ precipitation for each day. Rainfall and snowfall are calculated as a portion of the daily total precipitation:

$$q_{precip,liq}[i] = c_{liq} \cdot q_{precip} \quad , \tag{10.1}$$

$$q_{precip,snow}[i] = (1 - c_{liq}) \cdot q_{precip} \quad , \tag{10.2}$$

where $c_{liq} \in [0, 1]$ is calculated using the linear transition method [McCabe and Wolock, 2009]. This method uses the average daily temperature T_{avg} to interpolate c_{liq} between the temperatures $T_{rain} = 4°\text{C}$ above which all precipitation is rainfall, and $T_{snow} = -2°\text{C}$ below which all precipitation is snowfall:

$$c_{liq} = \begin{cases} 1 & \text{if } T_{avg} \geq T_{rain} \\ \frac{T_{avg} - T_{snow}}{T_{rain} - T_{snow}} & \text{if } T_{rain} > T_{avg} > T_{snow} \\ 0 & \text{if } T_{avg} \leq T_{snow} \end{cases} \quad . \tag{10.3}$$

There are alternative methods to calculate the proportion between liquid from solid precipitations based on surface-level observations [Harpold et al., 2017]. These include the use of a static threshold, using the minimum and maximum daily temperatures

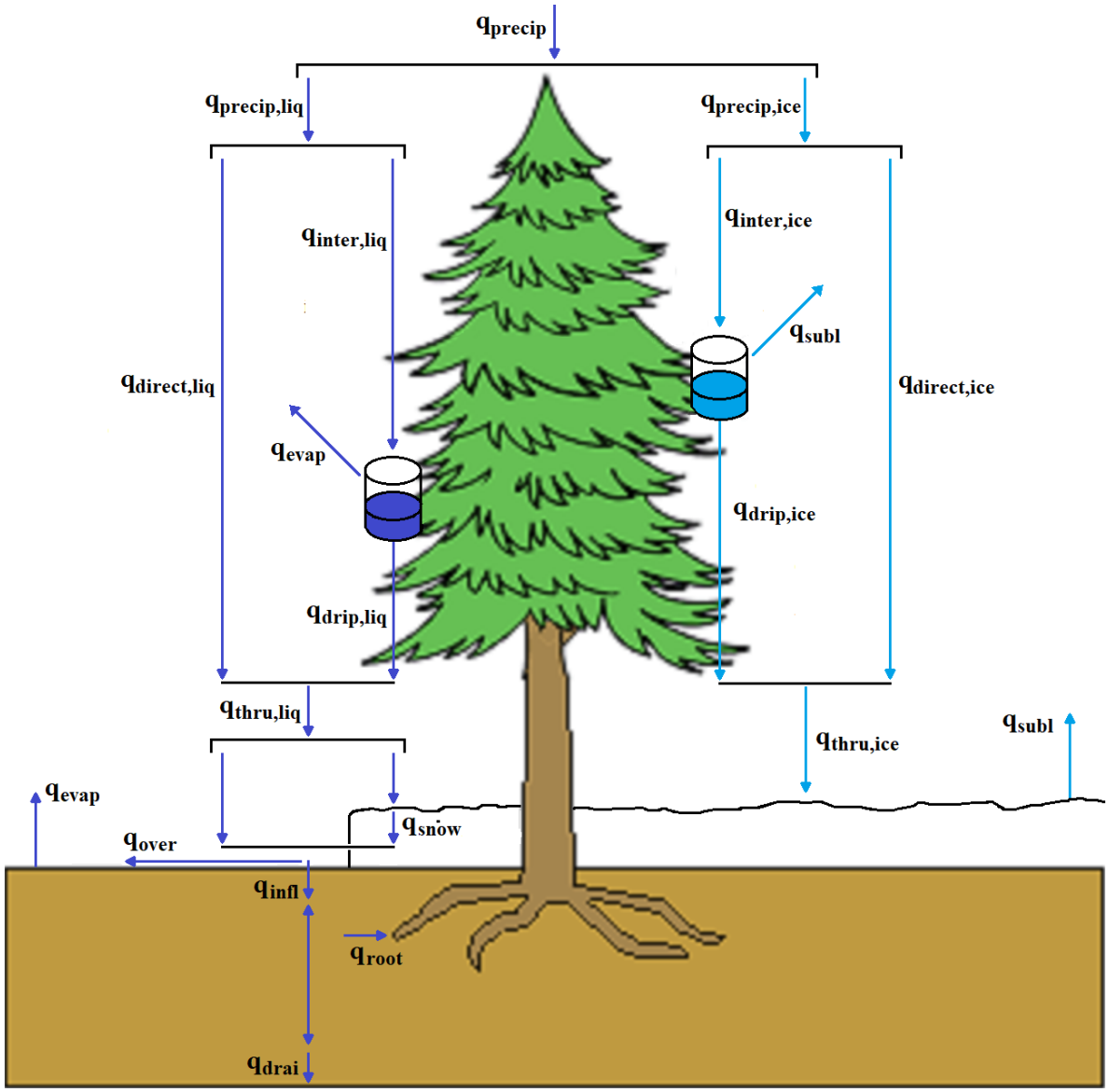

Figure 10.1: Main hydrological fluxes.

[Leavesley et al., 1996], or a sigmoidal curve [Dai, 2008]. However, the linear transition method was able to best reproduce this proportion on tested sites where rainfall, snowfall and temperature data was available.

10.1.2 Snow blow

Testing of the snow module in coastal sites showed an sizable discrepancy between snowfall and accumulated snow on the ground, where the size of the snow pack was excessively small to match the cumulative snowfall, even when accounting for very high

sublimation. This pointed towards snow blow as an important factor in certain sites, where snow blowing out of the site is not compensated by snow blowing in from the neighbourhood.

The net blowing of snow cannot be estimated easily, because it requires daily wind data to estimate the amount of blown snow and a model of the local topography to estimate the relation between blow-in and blow-out. This is out of the scope of a local-scaled model such as MAIDEN. Therefore, we instead introduce parameter for the proportion of blown snow s_{blow}, that has to be calibrated locally. Typically, non-coastal sites in flat grounds (a symmetric topography) will have a $s_{blow} \approx 0$, as the blow-in and blow-out of snow can be expected to compensate each other. This parameter modifies Eq. (10.2) into:

$$q_{precip,snow}[i] = (1 - c_{liq})(1 - s_{blow}) \cdot q_{precip} \quad . \tag{10.4}$$

10.1.3 Direct precipitation

Direct precipitation, the portion of precipitation that reaches the ground without striking the canopy, is calculated as a function of the plant area index (PAI):

$$q_{pdirect,liq}[i] = q_{precip,liq} \cdot \exp(-0.18 \cdot \text{clumping} \cdot \text{PAI}) \quad , \tag{10.5}$$

$$q_{pdirect,snow}[i] = q_{precip,snow} \cdot \exp(-0.18 \cdot \text{clumping} \cdot \text{PAI}) \quad , \tag{10.6}$$

where vegetation clumping is an input parameter of the model and:

$$\text{PAI}[i] = \text{SAI} + \text{LAI} \quad , \tag{10.7}$$

where $LAI[i]$ is the leaf area index (Eq. (15.13)) and SAI is the stem area index (which can also be called WAI, wood area index) a model parameter with a value of $SAI = 0.17$ for black spruce.

10.2 Canopy water

10.2.1 Canopy interception

The amount of rainfall and snowfall intercepted by the canopy is calculated by subtracting direct precipitation from precipitation:

$$q_{inter,liq}[i] = q_{precip,liq} - q_{pdirect,liq} \tag{10.8}$$

$$q_{inter,snow}[i] = q_{precip,snow} - q_{pdirect,snow} \tag{10.9}$$

This is the added to the canopy water (snow) storage cws (css), up to a maximum capacity. This capacity is proportional to the PAI:

$$\mathrm{cws}_{\max}[i] = \mathrm{cws}_{\mathrm{coef}} \cdot \mathrm{PAI} \quad , \tag{10.10}$$

$$\mathrm{css}_{\max}[i] = \mathrm{css}_{\mathrm{coef}} \cdot \mathrm{PAI} \quad . \tag{10.11}$$

The canopy water storage coefficient is $\mathrm{cws}_{\mathrm{coef}} = 0.34$ mm for black spruce. The canopy snow storage coefficient was measured for Engelmann spruce as $\mathrm{css}_{coef} = 5.9$ mm [Pomeroy et al., 1998], which we assume to be the same for black spruce. The reservoirs of canopy water and canopy snow are currently independent, a simplification that can hold because of the reduced amount of situations where there would be a conflict between the two.

The storage of canopy water is always 0 before adding the intercepted rainfall, because the remaining water at the end of the day is supposed to drip overnight. However, the snow stays from the previous day. The amounts of canopy water/snow after interception are:

$$\mathrm{PrecipOnLeaves}[i] = \min(q_{inter,liq}, \mathrm{cws}_{\max}) \quad , \tag{10.12}$$

$$\mathrm{SnowOnLeaves}[i] = \min(\mathrm{css}[i] + q_{inter,snow}, \mathrm{css}_{\max}) \quad , \tag{10.13}$$

where $\mathrm{css}[i]$ is the canopy snow storage at the beginning of day i. The excess rain/snow are added to the direct rainfall/snowfall:

$$q'_{pdirect,liq}[i] = q_{pdirect,liq} + \max(q_{inter,liq} - \mathrm{PrecipOnLeaves}, 0) \quad , \tag{10.14}$$

$$q'_{pdirect,snow}[i] = q_{pdirect,snow} + \max(\mathrm{css}[i] + q_{inter,snow} - \mathrm{SnowOnLeaves}, 0) \quad , \tag{10.15}$$

10.2.2 Canopy evaporation

Potential evaporation is calculated using the Penman-Monteith equation. For liquid water, the equation was originally implemented as:

$$\lambda E_{pot,liq}[i] = \frac{\Delta \cdot R + \rho_{air} \cdot C_{air} \cdot \delta_{atm}/r_{hr}}{\Delta + \gamma(1 + r_s/r_{hr})} \quad , \tag{10.16}$$

where Δ (Pa/K) is the change rate of the specific humidities of air to temperature, R (W/m^2) is the incoming radiation, $\rho_{air} \approx 1.292$ kg/m^3 is the density of air, $C_{air} = 1010$ J kg^{-1}K^{-1} is the specific heat of air, δ_{atm} (Pa) is the vapour pressure deficit, $r_s = 0$ s/m is the stomatal resistance, $r_{hr} = \frac{r_h r_r}{r_h + r_r}$ (s/m) is the combined resistance (in parallel) of convective and radiative heat transfer, and γ (Pa/kg) is the psychometric parameter:

$$\gamma[i] = \frac{\mathrm{C_P} \cdot \mathrm{P_{atm}}}{\lambda_{\mathrm{vap,Tday}} \cdot \epsilon_{wa}} \quad , \tag{10.17}$$

where $\lambda_{\text{vap,Tday}} = 2.5 \cdot 10^6 - 2430.54 \cdot T_{day}$ J/kg is the latent heat of vaporization, $\epsilon_{wa} = 0.62196351$ is the unitless ratio of molecular weight of water vapour to dry air, P_{atm} is the atmospheric pressure, and $\text{C}_\text{P} = 1010.0$ J kg^{-1} K^{-1} is the specific heat of air.

Eq. (10.16) gives the potential evaporation in units of W/m^2. This energy flux rate can be transformed into a volume flux rate in mm/day:

$$E_{pot,liq}[i] = \lambda E_{pot,liq} \cdot \frac{dayL}{\rho_{liq} \cdot \lambda_{vap}} \quad . \tag{10.18}$$

with $dayL$ (s/day) being the length of the day (time between sunrise and sunset) and ρ_{liq} is the density of liquid water. Then, actual evaporation is calculated as the minimum between the potential evaporation and the canopy water storage.

$$E_{liq}[i] = \min(E_{pot,liq}, cws_{liq}) \quad , \tag{10.19}$$

which can be transformed again to W/m^2:

$$\lambda E_{liq}[i] = E_{liq} \cdot \frac{\rho_{liq} \cdot \lambda_{vap}}{dayL} \quad . \tag{10.20}$$

To represent the sublimation of snow, we use the same equation as Eq. (10.16) but using the latent heat of sublimation $\lambda_{sub} = \lambda_{vap} + L_f$, where $L_f = 3.337 \cdot 10^5$ J kg^{-1} is the latent heat of fusion, and modifying the radiation-dependent term with the snow albedo $\alpha_s = 0.85$ [Greuell and Konzelmann, 1994]:

$$\lambda E_{pot,snow}[i] = \frac{(1-\alpha_s)\Delta \cdot R + \rho_{air} \cdot C_{air} \cdot \delta_{atm}/r_{hr}}{\Delta + \gamma(1 + r_s/r_{hr})} \quad . \tag{10.21}$$

Snow is sublimated from the remaining energy after all canopy water has been evaporated. Before the actual sublimation of snow is calculated, the actual evaporation of water λE_{liq} is subtracted from $\lambda E_{pot,snow}$ (in W/m^2 units).

$$\lambda E'_{pot,snow}[i] = \cdot(\lambda E_{pot,snow} - \lambda E_{liq}) \quad , \tag{10.22}$$

The potential sublimation is then transformed from W/m^2 to mm/day by multiplying it by $dayL/(\rho_{liq} \cdot \lambda_{sub})$, where we use ρ_{liq} instead of ρ_{snow} because ice and snow are measured in water-equivalent units (mm). Actual sublimation of snow is calculated as the minimum between $E_{pot,snow}$ and cws_{snow}.

$$E_{snow}[i] = \min(E'_{pot,snow}, cws_{snow}) \quad , \tag{10.23}$$

The water/snow amounts in the canopy after evaporation/sublimation but before canopy drip are:

$$\text{cws}[i + \frac{1}{2}] = \text{PrecipOnLeaves} - E_{liq} \quad , \tag{10.24}$$

$$\text{css}[i + \frac{1}{2}] = \text{PrecipOnLeaves} - E_{snow} \quad . \tag{10.25}$$

10.2.3 Canopy drip

In the case of liquid water, the amount of water that remains in the canopy after evaporation drips to the ground at the end of the day, adding to liquid throughfall:

$$q_{thru,liq}[i] = q'_{direct,liq} + \mathrm{cws}[i + \frac{1}{2}] \quad , \tag{10.26}$$

$$\mathrm{cws}[i+1] = 0 \quad . \tag{10.27}$$

In the case of canopy snow, a drip model taken from CLM5 [Lawrence et al., 2019] is implemented to represent snow drip from above freezing temperatures:

$$q_{drip,snow}[i] = \frac{\mathrm{css}[i + \frac{1}{2}] \cdot (T_{avg} - 270\ \mathrm{K})}{1.87 \cdot 10^5\ \mathrm{K \cdot s}} \cdot 86400\ \mathrm{s/day} \geqslant 0 \quad . \tag{10.28}$$

This equation implies that if the average temperature $T_{avg} > -1$ C, the snow drip is larger than the canopy snow, and therefore all snow falls during the night. CLM5 implements time steps of 20-30 minutes, therefore it is likely that this result is was not intended. Also, the snow drip has another term for wind unloading, which has not been currently implemented because of lack of wind data:

$$q_{unl,wind}[i] = \frac{u \cdot \mathrm{css}[i + \frac{1}{2}]}{1.56 \cdot 10^5\ \mathrm{m}} \quad , \tag{10.29}$$

where u (m/s) is the wind speed.

The amount of snow reaching the ground, or snow throughfall, is:

$$q_{thru,snow}[i] = q'_{direct,snow} + q_{drip,snow} \quad , \tag{10.30}$$

$$\mathrm{css}[i+1] = \mathrm{css}[i + \frac{1}{2}] - q_{drip,snow} \quad . \tag{10.31}$$

Chapter 11: Surface hydrology

Soil and snow calculations in MAIDENiso are made within a sub-daily time interval, in order to improve the accuracy of the calculations. This time interval Δt is henceforth referred to as "hourly" and quantities that are calculated hourly have their time dependence indicated with the letter h, as opposition to the daily quantities labelled by i. It is to be noted that Δt does not need to be an exact hour even though is called "hourly" (we call it so to keep notation simple), and can be changed to other sub-daily time intervals if a user so desires. For simplicity reasons too, the "hourly" time interval has been chosen to be one exact hour, $\Delta t = 3600$ s. Note that the hydrological calculations are integrated with the thermal calculations. Thus, at the beginning of each day i, both the hydrological and thermal calculations for $h = 1$ will be calculated before moving to $h = 2$.

The daily fluxes are simply calculated by adding the hourly fluxes for that day. The model does not store the hourly values from past days, only these daily fluxes which are the values written in the outputs of the model.

11.1 Snow

11.1.1 Snow pack dynamics

The snow pack is implemented as a dynamic layer on top of the usual soil layers. This layer is special as it can either cover the soil completely, be non-existing, or cover the soil partially. In the latter case, the fluxes between the ground and the surface or the atmosphere are proportionally distributed between the snow layer and the uppermost soil layer.

The snow layer has a thickness $\Delta z_{snow}[h]$ (m), an ice content $\omega_{ice,0}[h]$ (kg), and a water mass content $\omega_{liq,0}[h]$ (kg). Note that we use the suffix 0 for the snow layer as we later use the suffix $j = 1..nslay$ for the soil layers below. Because the thickness of the snow layer changes dynamically, it is easier to track absolute masses of ice and water than to keep track of volumetric contents as it is done for the soil layers.

The calculation of thermal conduction (Chapter 12) can produce numerical diver-

gence if the time-step Δt is too large compared to layer thickness Δz. For the time-step $\Delta t = 3600$ used in the temperature calculations, numerical divergences can happen if the snow layer has a thickness $\Delta z < 0.05$ m. To ensure numerical convergence, the snow layer has a minimum thickness $\Delta z_{snow,min} = 0.1$ m. If the thickness of the snow layer is reduced below this value, we simulate a situation where the snow only covers the ground partially, in patches of thickness $\Delta z_{snow,min}$. The fraction of snow cover is calculated as:

$$f_{snow}[h] = \frac{\Delta z_{snow}}{\Delta z_{snow,min}} \leq 1 \quad . \tag{11.1}$$

In this situation, we still keep track of Δz_{snow} as the depth the snow pack would have if spread evenly, which is used to calculate snow density. Thermal calculations instead use $\Delta z_{snow,min}$ for the snow-covered portion f_{snow} of the ground, while ignoring snow cover in the snow-free portion. f_{snow} is also used to partition the calculations performed at the surface for incoming liquid throughfall and evaporation.

The snow layer grows from incoming solid precipitation at the surface, $q_{thru,snow}[i]$ (defined as a flux, mm/s, not the daily precipitation). The mass of snow in the top snow layer grows by $\Delta\omega_{ice,0}[h] = q_{thru,snow}[i]\Delta t$. The thickness of the snow layer grows accordingly, given the density of newly fallen snow ρ_{nsnow} (kg/m^3). This is dependent on a temperature-driven term ρ_T and a wind-driven compaction term ρ_u. The temperature-driven term ρ_T is:

$$\rho_{nsnow}[h] = \rho_T + \rho_u \quad , \tag{11.2}$$

$$\rho_T[h] = \begin{cases} 50 + 1.7 \cdot (17)^{1.5} & \text{if } \mathrm{T_{atm}} > \mathrm{T_F} + 2 \\ 50 + 1.7 \cdot (\mathrm{T_{atm}} - \mathrm{T_F} + 15)^{1.5} & \text{if } \mathrm{T_F} - 15 \leq \mathrm{T_{atm}} < \mathrm{T_F} + 2 \\ 50 & \text{if } \mathrm{T_{atm}} \leq \mathrm{T_F} - 15 \end{cases} \quad , \tag{11.3}$$

where $\mathrm{T_{atm}}$ is the atmospheric temperature from Eq. (12.39), which is calculated hourly.

The wind-driven compaction term [van Kampenhout et al., 2017] is:

$$\rho_u[i] = 266.816 \left(\frac{1 + \tanh(u/5)}{2} \right)^{8.8} \quad . \tag{11.4}$$

We typically do not have daily wind data available for MAIDEN. Therefore, the wind-driven compaction is applied using the average wind speed u at the site. Given this density, the thickness of the snow layer increases by $\Delta\omega_{ice,0}/\rho_{nsnow}$.

11.1.2 Snow water

Each time-step h, the water content $\omega_{liq,0}$ of the snow layer is first updated with the inflow $q_{liq,in}$ and the outflow $q_{liq,out}$.

$$\omega_{liq,0}[h + \frac{1}{2}] = \omega_{liq,0}[h] + (q_{liq,in} - q_{liq,out})\Delta t \quad . \tag{11.5}$$

In a second step, after the thermal state of the snow and soil layers is calculated, the water content is updated for freezing/thawing (calculated in section 12.3):

$$\omega_{liq,0}[h+1] = \omega_{liq,0}[h+\frac{1}{2}] + \Delta\omega_{liq,0} \quad . \tag{11.6}$$

The outflow $q_{liq,out}$ comes from the percolation of liquid water through the snow, and it is limited by the infiltration capacity of the soil $q_{infl,max}$ from Eq. (11.15).

$$q_{liq,out}[h] = \frac{\rho_{liq}[\theta_{liq} - S_r(1-\theta_{ice})]\Delta z_{snow}}{\Delta t} \leqslant q_{infl,max} \quad , \tag{11.7}$$

where $S_r = 0.033$ is the irreductible water saturation and $\rho_{liq} = 1000$ kg m^{-3} is the density of liquid water (and the density of ice is $\rho_{ice} = 917$ kg m^{-3}). The volumetric contents of ice θ_{ice} and water θ_{liq} are:

$$\theta_{ice}[h] = \frac{\omega_{ice,0}}{\Delta z_{snow}\rho_{ice}} \quad , \tag{11.8}$$

$$\theta_{liq}[h] = \frac{\omega_{liq,0}}{\Delta z_{snow}\rho_{liq}} \quad . \tag{11.9}$$

The inflow of liquid water $q_{liq,in}$ into the snow layer is calculated as incoming liquid precipitation minus the flux of snow water evaporation, and can therefore be negative.

$$q_{liq,in}[h] = f_{snow}(q_{thru,liq} - q_{evap,snow}) \quad . \tag{11.10}$$

The evaporative flux of snow water $q_{evap,snow}$ (mm s) is calculated from the potential evaporative flux $\lambda E_{pot,snow}[i]$ (W/m^2) in Eq. (11.69). The evaporated water from the snow layer in a time-step h, $q_{evap,snow}[h]\Delta t$, cannot exceed the quantity of water in the snow layer in that interval, $\omega_{liq,0}[h]$. Therefore:

$$q_{evap,snow}[h] = \min\left(\frac{\omega_{liq,0}}{\Delta t}, \frac{\lambda E_{pot,snow}}{\lambda_{vap}}\right) \quad . \tag{11.11}$$

If water evaporation from snow $q_{evap,snow}$ is not enough to meet the atmospheric demand $\frac{\lambda E_{pot,snow}}{\lambda_{vap}}$, the unused energy is then used to sublimate snow. The sublimation flux of snow $q_{sub,snow}$ is:

$$q_{sub,snow}[h] = \frac{\lambda_{vap}}{\lambda_{sub}}\left(\frac{\lambda E_{pot,snow}}{\lambda_{vap}} - q_{evap,snow}\right) \quad , \tag{11.12}$$

where λ_{vap} is the latent heat of vaporization and $\lambda_{sub} = \lambda_{vap} + L_f$ is the latent heat of sublimation, L_f being the latent heat of fusion (Table 12.1).

At any given time, the snow layer has a maximum capacity to store water, determined by the thickness and density of the snow:

$$\omega_{liq,0,max} = \Delta z_{snow}\rho_{liq} - \omega_{ice,0}\frac{\rho_{liq}}{\rho_{ice}} \quad . \tag{11.13}$$

If the liquid water in the snow layer $\omega_{liq,0}$, after snow water movement has been calculated, exceeds $\omega_{liq,0,max}$, the excess of water will be added to the runoff q_{over} calculated in Eq. (11.17).

11.2 Infiltration and runoff

Infiltration into the soil uses a simple model, where all the liquid water reaching the soil surface, $q_{in,surface}$, is able to infiltrate up to a maximum capacity:

$$q_{infl}[h] = \min(q_{in,surface}, q_{infl,max}) \quad , \tag{11.14}$$

This maximum infiltration capacity $q_{infl,max}$ is determined as:

$$q_{infl,max}[h] = \Theta_{ice,1}\kappa_{sat,1} \quad , \tag{11.15}$$

where $\Theta_{ice,1}$ and $\kappa_{sat,1}$ are respectively the ice impedance (Eq. (11.22)) and the saturated hydraulic conductivity (Eq. (11.20)) of the upper soil layer.

On the other hand, the water that reaches the soil surface is:

$$q_{in,surface}[h] = (1 - f_{snow})q_{thru,liq} + q_{liq,N_{snowlay}} \quad , \tag{11.16}$$

where $q_{liq,N_{snowlay}}$ is the outflux of water from the bottom snow layer. If there is enough snow to consistently cover the ground, $f_{snow} = 1$ and therefore the only water reaching the soil surface comes from the snow layer. Otherwise, throughfall reaching the ground is proportional to the surface free of snow. The fraction that hits the snow, $f_{snow}q_{thru,liq}$, is added to the snow layer instead. Note that snow throughfall $q_{thru,snow}$ is added entirely to the snow regardless of the value of f_{snow}.

Runoff q_{over} is then simply calculated as:

$$q_{over}[h] = q_{in,surface} - q_{infl} \quad . \tag{11.17}$$

11.3 Soil water

11.3.1 Hydrological properties

Soil layers are denoted by $j = 1..nslay$, where the number of soil layers is $nslay = 4$ (though the model can be configured to have $nslay = 5$ layers). The soil can contain

ice in addition to liquid water. For a layer j we denote by $\theta_{liq,j}$ its volumetric content of liquid water and by $\theta_{ice,j}$ its volumetric content of ice. For hydrological calculations, it is preferable to work in terms of volumetric content, but for thermal calculations (see Chapter 12) it is preferable to work in terms of masses of liquid water $\omega_{liq,j}[h]$ and ice $\omega_{ice,j}[h]$. At any time-step, the conversion between the volumetric content and mass is straightforward given the thickness of the layer Δz_j and the liquid/ice densities:

$$\theta_{liq,j} = \frac{\omega_{liq,j}}{\rho_{liq}\Delta z_j} \quad , \tag{11.18}$$

$$\theta_{ice,j} = \frac{\omega_{liq,j}}{\rho_{ice}\Delta z_j} \quad . \tag{11.19}$$

Note that $\omega_{liq,j}$ and $\omega_{ice,j}$ are masses in kg, but we can also work interchangeably with them in equivalent units of "mm of liquid water" (height that such mass per unit of area would reach, with 1 kg/mm for water), which is usual in hydrology. For ice, this is understood as the height that it would reach if melted.

Ice competes with liquid water for available porous space, which decreases the hydraulic conductivity κ_j (calculated hourly) of a soil layer:

$$\kappa_j[h] = \Theta_{ice,j}\kappa_{sat,j}\left(\frac{\theta_{liq,j}}{\theta_{sat,j}}\right)^{2B_j+3} \quad , \tag{11.20}$$

where $\theta_{liq,j}[h]$ is the volumetric (liquid) water content of the layer, $\theta_{sat,j}$ is the volumetric content at saturation, B_j is the "Clapp and Hornberger B exponent" (which accounts for organic matter), and $\Theta_{ice,j}$ is the impedance factor due to ice occupying porous space:

$$B_j = (1 - f_{om,j}) \cdot (2.91 + 0.159 \cdot (\%\text{clay})_j) + f_{om,j} \cdot 2.7 \quad , \tag{11.21}$$

$$\Theta_{ice,j}[h] = 10^{-6 \cdot \frac{\theta_{ice,j}}{\theta_{sat,j}}} \quad , \tag{11.22}$$

where $\theta_{ice,j}$ is the volumetric ice content of the layer j. When $\theta_{ice,j} = 0$, then $\Theta_{ice,j} = 1$, and Eq. (11.20) reads as it did in the versions previous to the incorporation of ice and snow with MAIDENiso v4.

The saturated hydraulic conductivity κ_{sat} (mm/s) is defined as:

$$\kappa_{sat,j} = \left(\frac{1 - f_{om,j}}{\kappa_{sat,min,j}} + \frac{f_{om,j}}{\kappa_{sat,om}}\right)^{-1} \quad , \tag{11.23}$$

where $\kappa_{sat,om} = 0.1$ mm/s is the hydraulic conductivity for organic matter, $f_{om,j}$ is the fraction of organic matter composition for layer j, and $\kappa_{sat,min,j}$ (mm/s) is the hydraulic conductivity for mineral soil:

$$\kappa_{sat,min,j} = 0.0070556 \cdot 10^{-0.884+0.0153(\%\text{sand})_j} \quad . \tag{11.24}$$

11.3.1.1 Darcy's law

The flow of water q is calculated through Darcy's law:

$$q = -\kappa \frac{\partial(\psi + z)}{\partial z} \quad , \tag{11.25}$$

where ψ is the soil matric potential. In MAIDENiso, the flow of water q_j between layers j and $j+1$ is implemented numerically as:

$$q_j[h] = -\frac{2(\psi_j - \psi_{j+1}) + (\Delta z_j + \Delta z_{j+1})}{\frac{\Delta z_j}{\kappa_j} + \frac{\Delta z_{j+1}}{\kappa_{j+1}}} \quad . \tag{11.26}$$

Soil matric potential is calculated as:

$$\psi_j[h] = \psi_{sat,j} \left(\frac{\theta_{liq,j}}{\theta_{sat,j}} \right)^{-B_j} < 0 \quad , \tag{11.27}$$

where $B_j = 2.91 + 0.159(\%\text{clay})$ and $\psi_{sat,j} = -10 \cdot 10^{1.88 - 0.0131(\%\text{sand})_j}$. In MAIDENiso, low values of $\theta_{liq,j}$ (typical for freezing temperatures) produce very large absolute values of ψ_j. This produces numerical problems in the calculations of hydrology, which can lead to numerical overflow. In imitation of the approach taken by Lawrence et al. [2019], ψ_j has been capped at a minimum value of $\psi_j > -10^8$. This ensures the convergence of the equations.

11.3.1.2 Field capacity

Field capacity can be derived assuming a hydraulic conductivity of 0.1 mm/d, then inverting Eq. (11.20):

$$\theta_{fc,j} = \theta_{sat,j} \left(\frac{0.1 \text{ mm/d}}{86400 \text{ s/d} \kappa_{sat,j}} \right)^{\frac{1}{2B_j+3}} \tag{11.28}$$

11.3.1.3 Soil water stress

Photosynthesis is directly influenced by soil water, through a soil water stress function $\theta_g[i]$, calculated on a daily basis.

Previous to MAIDENiso v4, the soil water stress was a function of the total water content of the ground. Two parameters soil_b and soil_ip controlled the slope and inflection point of this function:

$$\theta_g[i] = \frac{1}{1 + \exp(\text{soil}_\text{b} \cdot (\text{SWC[i]} - \text{soil}_\text{ip}))} \quad . \tag{11.29}$$

This function presents some weaknesses. First, it is not mechanistic and its form is arbitrary. Second, it requires the calibration of 2 parameters. Third, it takes into account the water content of the whole soil, regardless of soil composition and even of the presence of roots. This constrains the soil module to the layers accessible to the tree, or the tree would take unavailable water into consideration. This is very restrictive for the hydrology of the model, which could otherwise benefit from additional lower layers.

In MAIDENiso v4, the soil water stress $\theta_g[h]$ is defined without the need of parameters, being ultimately controlled by the humidity of the soil layers and physiological constants of the tree (the osmotic potential of the fully open/closed stomata):

$$\theta_g[h] = \sum_j \mathrm{wilt} r_j \quad , \tag{11.30}$$

where r_j is the root fraction in the layer j (each r_j being a parameter of the model) and wilt_j is the wilting factor of the layer j:

$$\mathrm{wilt}_j[h] = \begin{cases} \frac{\psi_c - \psi_j}{\psi_c - \psi_o} \left(\frac{\theta_{sat,j} - \theta_{ice,j}}{\theta_{sat,j}} \right) \leq 1 & \text{if } T_j > \mathrm{T_F} - 2 \ \& \ \theta_{liq,j} > 0 \\ 0 & \text{if } T_j \leq \mathrm{T_F} - 2 \text{ or } \theta_{liq,j} \leq 0 \end{cases} \quad , \tag{11.31}$$

where ψ_c and ψ_o are the soil water potentials (mm) when the stomata are, respectively, fully closed or fully opened, and T_j is the temperature of the layer j. For needleleaf or for broadleaf evergreen trees, $\psi_c = 255000$ mm and $\psi_o = 65000$ mm, while for broadleaf deciduous trees $\psi_c = 224000$ mm and $\psi_o = 35000$ mm.

$\theta_g[h]$ is calculated hourly, however it is applied in the photosynthesis module (section 16.2) with the daily time-step. The daily $\theta_g[i]$ is calculated as an average of all the hourly $\theta_g[h]$ within the same day.

In addition, a yearly hydraulic stress factor $\mathrm{H_{stress}}$ is calculated at the end of the year. This is done by averaging the daily $\theta_g[i]$ values during the growth season, which is used by the tree to adjust the targeted size of the canopy for the next year (see section 15.3):

$$\mathrm{H_{stress}} = \frac{\sum d_i \cdot \theta_{g,i}}{\sum d_i} \quad , \tag{11.32}$$

where $d_i = 1$ during Summer and Autumn and $d_i = 0$ during Spring and Winter.

11.3.2 Numerical solution

The conservation of water for each layer gives the following equation:

$$\Delta z_j \frac{\partial \theta_{liq,j}}{\partial t} = -q_{j-1} + q_j - e_j \quad , \tag{11.33}$$

where Δz_j (mm) is the thickness of layer j, q_j is the flux of water from layer j to layer $j+1$, and e_j is the sink of soil moisture via evapotranspiration, defined positive for flow out of the layer (mm/s). In the soil module the fluxes are evaluated at discrete intervals of $\Delta t = 3600$ s (using h as the integer for this subdaily time-step), giving us:

$$\Delta z_j[h]\frac{\Delta\theta_{liq,j}}{\Delta t} = -q_{j-1}[h+1] + q_j[h+1] - e_j \quad , \tag{11.34}$$

where $\Delta\theta_{liq,j} = \theta^{n+1}_{liq,j} - \theta^{n}_{liq,j}$ is the change in volumetric soil liquid water of the layer j in time Δt.

Soil evaporation can happen only to the upper soil layer $j = 1$. It is calculated as the potential evaporative flux $E_{pot,soil}[i]$ (kg m^{-2} s^{-1}) from Eq. (11.66) (see section 11.4), limited by the available water in the layer:

$$q_{evap,soil}[h] = \min\left(\frac{\omega_{liq,1}}{\Delta t}, E_{pot,soil}[i]\right) \quad . \tag{11.35}$$

The layer water removed by evapotranspiration is given by evaporation ($q_{evap,soil}$, only for upper soil layer $j = 1$) and transpiration (q_{tran}, see section 16.4):

$$e_j[h] = \begin{cases} (1 - f_{snow})q_{evap,soil} + q_{tran}r_{e,j} & \text{if } j = 1 \\ q_{tran}r_{e,j} & \text{if } j > 1 \end{cases} \quad , \tag{11.36}$$

where f_{snow} is the portion of the soil covered in snow (as evaporation only affects the snow-free portion). $r_{e,j}$ is the effective root fraction for layer j, which can be defined in several ways. The original formulation is simply:

$$r_{e,j} = r_j \quad . \tag{11.37}$$

Another possibility, is to use the wilting factor wilt_j (subsection 11.3.1.3) to weight the root fraction r_j:

$$r_{e,j}[h] = \frac{\text{wilt}_j \cdot r_j}{\sum_j \text{wilt}_j \cdot r_j} \quad . \tag{11.38}$$

A third possibility is to avoid the root fraction entirely, which it has to be arbitrarily defined by the user. This can be done under the assumption that the tree absorbs water from each layer in proportion to the wilting in that layer and to its water content at saturation. This is equivalent to the previous model, assuming root fractions proportional to the maximum capacity of the layers.

$$r_{e,j}[h] = \frac{\text{wilt}_j \Delta z_j \theta_{sat,j}}{\sum_j \text{wilt}_j \Delta z_j \theta_{sat,j}} \quad . \tag{11.39}$$

The soil water fluxes from Eq. (11.34) can be linearised with a Taylor series expansion.

$$q_j[h+1] = q_j[h] + \frac{\partial q_j}{\partial \theta_{liq,j}}\Delta\theta_{liq,j} + \frac{\partial q_j}{\partial \theta_{liq,j+1}}\Delta\theta_{liq,j+1} \quad , \tag{11.40}$$

$$q_{j-1}[h+1] = q_{j-1}[h] + \frac{\partial q_{j-1}}{\partial \theta_{liq,j-1}} \Delta\theta_{liq,j-1} + \frac{\partial q_{j-1}}{\partial \theta_{liq,j}} \Delta\theta_{liq,j} \quad . \tag{11.41}$$

When these equations are substituted in Eq. (11.34), we obtain a general tridiagonal equation set:

$$\Lambda_{r,j} = \Lambda_{a,j}\Delta\theta_{j-1} + \Lambda_{b,j}\Delta\theta_j + \Lambda_{c,j}\Delta\theta_{j+1} \quad , \tag{11.42}$$

where $\Lambda_{a,j}$, $\Lambda_{b,j}$ and $\Lambda_{c,j}$ are the subdiagonal, diagonal and superdiagonal elements of the tridiagonal matrix, respectively, and $\Lambda_{r,j}$ is a column vector of constants. Note that we have used this notation instead of the simpler a_j, b_j, c_j and r_j to avoid confusion with other quantities. These elements are:

$$\Lambda_{a,j} = -\frac{\partial q_{j-1}}{\partial \theta_{liq,j-1}} \quad , \tag{11.43}$$

$$\Lambda_{b,j} = \frac{\partial q_j}{\partial \theta_{liq,j}} - \frac{\partial q_{j-1}}{\partial \theta_{liq,j}} - \frac{\Delta z_j}{\Delta t} \quad , \tag{11.44}$$

$$\Lambda_{c,j} = \frac{\partial q_j}{\partial \theta_{liq,j+1}} \quad , \tag{11.45}$$

$$\Lambda_{r,j} = q_{j-1}[h] - q_j[h] + e_j \quad . \tag{11.46}$$

The fluxes and their partial derivatives are computed with a finite difference form, as:

$$q_j[h] = \frac{-2\cdot(\psi_j - \psi_{j+1}) - (\Delta z_j + \Delta z_{j+1})}{(\frac{\Delta z_j}{\kappa_j} + \frac{\Delta z_{j+1}}{\kappa_{j+1}})} \quad , \tag{11.47}$$

$$\frac{\partial q_j}{\partial \theta_{liq,j}} = \frac{-2(\frac{\Delta z_j}{\kappa_j} + \frac{\Delta z_{j+1}}{\kappa_{j+1}})\frac{\partial \psi_j}{\partial \theta_{liq,j}} + (-2\cdot(\psi_j - \psi_{j+1}) - (\Delta z_j + \Delta z_{j+1}))\frac{\Delta z_j}{\kappa_j^2}\frac{\partial \kappa_j}{\partial \theta_{liq,j}}}{(\frac{\Delta z_j}{\kappa_j} + \frac{\Delta z_{j+1}}{\kappa_{j+1}})^2} \quad , \tag{11.48}$$

where:

$$\frac{\partial \psi_j}{\partial \theta_{liq,j}} = -B_j \frac{\psi_j}{\theta_{liq,j}} \quad , \tag{11.49}$$

$$\frac{\partial \kappa_j}{\partial \theta_{liq,j}} = (2B_j + 3)\frac{\kappa_j}{\theta_{liq,j}} \quad . \tag{11.50}$$

For the upper layer ($j = 1$), the coefficients of the tridiagonal set of equations are:

$$\Lambda_{a,j} = 0 \quad , \tag{11.51}$$

$$\Lambda_{b,j} = \frac{\partial q_j}{\partial \theta_{liq,j}} - \frac{\Delta z_j}{\Delta t} \quad , \tag{11.52}$$

$$\Lambda_{c,j} = \frac{\partial q_j}{\partial \theta_{liq,j+1}} \quad , \tag{11.53}$$

$$\Lambda_{r,j} = q_{infl}[h] - q_j[h] + (1 - f_{snow})q_{evap,soil} + q_{tran}r_{e,j} \quad . \tag{11.54}$$

For the intermediate layers $j = 2, 3 \ ... \ nslay - 1$:

$$\Lambda_{a,j} = -\frac{\partial q_{j-1}}{\partial \theta_{liq,j-1}} \quad , \tag{11.55}$$

$$\Lambda_{b,j} = \frac{\partial q_j}{\partial \theta_{liq,j}} - \frac{\partial q_{j-1}}{\partial \theta_{liq,j}} - \frac{\Delta z_j}{\Delta t} \quad , \tag{11.56}$$

$$\Lambda_{c,j} = \frac{\partial q_j}{\partial \theta_{liq,j+1}} \quad , \tag{11.57}$$

$$\Lambda_{r,j} = q_{j-1}[h] - q_j[h] + q_{tran}r_{e,j} \quad . \tag{11.58}$$

And for the bottom layer $j = nslay$:

$$\Lambda_{a,j} = -\frac{\partial q_{j-1}}{\partial \theta_{liq,j-1}} \quad , \tag{11.59}$$

$$\Lambda_{b,j} = \frac{\partial q_j}{\partial \theta_{liq,j}} - \frac{\partial q_{j-1}}{\partial \theta_{liq,j}} - \frac{\Delta z_j}{\Delta t} \quad , \tag{11.60}$$

$$\Lambda_{c,j} = 0 \quad , \tag{11.61}$$

$$\Lambda_{r,j} = q_{j-1}[h] + \kappa_j + q_{tran}r_{e,j} \quad . \tag{11.62}$$

Solving the tridiagonal equations yields $\Delta\theta_j$ for each time-step. The moisture of each layer is updated as:

$$\theta_{liq,j}[h + \frac{1}{2}] = \theta_{liq,j}[h] + \Delta\theta_j \quad , \tag{11.63}$$

where the final moisture of each layer is obtained after adding the change in water content after melting/freezing (which is obtained in terms of mass $\Delta\omega_{liq,j}$, see section 12.3):

$$\theta_{liq,j}[h + 1] = \theta_{liq,j}[h + \frac{1}{2}] + \frac{\Delta\omega_{liq,j}}{\rho_{liq}\Delta z_j} \quad . \tag{11.64}$$

And the soil drainage is:

$$q_{drai}[h] = (\kappa_{nslay} + \frac{\partial \kappa_{nslay}}{\partial \theta_{liq,nslay}}\Delta\theta_{nslay})\Delta t \tag{11.65}$$

11.4 Soil evaporation

Soil evaporation uses a different implementation for the Penman-Monteith equation than the canopy water, calculating potential evaporation from the soil $E_{pot,soil}$ (kg m^{-2} s^{-1}) as:

$$E_{pot,soil}[i] = \frac{\rho_{air} \mathrm{MR}_{sat}(\mathrm{RH}_{soil} - \mathrm{RH}_{atm})}{r_{soil}} \quad , \tag{11.66}$$

where MR_{sat} is the mixing ratio at saturation and RH_{soil} and RH_{atm} are the relative humidities of the soil and the atmosphere. The relative humidity of the soil is calculated as:

$$\mathrm{RH}_{soil}[i] = \exp(G\frac{\psi_1}{R\mathrm{T}_{\mathrm{avg}}}) \quad , \tag{11.67}$$

where G is the gravitational constant, R is the gas constant, ψ_1 is the soil water potential of the upper soil layer and $\mathrm{T}_{\mathrm{avg}}$ is the atmospheric temperature averaged over the last 30 days (likely because there is no calculation of soil temperature).

The implementation in Eq. (11.66) uses the soil resistance to water vapour movement, calculated as $r_{soil} = 3.8113 \cdot 10^6 \exp(-13.515\frac{\theta_{liq,1}}{\theta_{sat,1}})$. [1]

If the formula in Eq. (11.66) is translated to the formulation used in Eqs. (10.16) and (10.21) for better comparison, it would read as:

$$\lambda E_{pot,soil}[i] = \frac{\rho_{air} \cdot C_{air} \cdot (e_{sat}\mathrm{RH}_{soil} - e_{atm})/r_{soil}}{0.622\gamma} \quad . \tag{11.68}$$

Therefore this formula does not account for the variation of saturation vapour pressure e_{sat} with temperature ($\Delta = 0$), and a factor 0.622 that was not present in Eq. (10.16) appears in the denominator. This formula might be revised in future versions of MAIDENiso.

Note on previous versions

Versions previous to MAIDENiso v4 performed a unit transformation for soil water potential from mm to m by dividing this quantity by 100 instead of 1000, thus overestimating the soil water potential tenfold.

[1] These quantities were not justified in the original code by L. Misson, who only pointed out that this formula came from Baldocchi. Digging into it, Baldocchi and Meyers [1998] got the formula from Mahfouf and Noilhan [1991], who compared several formulations of evaporation. This specific formula was taken from Passerat de Silans [1986], a thesis that only exists in paper in the university of Grenoble.

11.5 Snow evaporation/sublimation

Evaporation from snow is calculated in parallel to soil evaporation, and each is applied proportionally to the fraction of the soil covered by snow. The formula for snow evaporation [Stigter et al., 2018] is similar to that Eq. (10.21), but it can be applied either to sublimate snow or to evaporate liquid water in the snow layer, adopting the appropriate latent heat λ for either sublimation or evaporation:

$$\lambda E_{pot,snow}[i] = \frac{(1-\alpha_s)\Delta\ R + \rho_{air}\ \mathrm{C_P}\ \delta_{atm}/r_a}{\Delta+\gamma} \quad . \tag{11.69}$$

Note that $\lambda E_{pot,snow}$ has units W m^{-2}. Here, r_a (s/m) is the aerodynamic resistance to water vapour transfer. There are a multitude of alternative formulas to calculate r_a, some extremely complicated and dependent on many parameters [Lawrence et al., 2019], others have a simpler dependency on the wind speed u [Blanken and Black, 2004]. Others use a constant value for r_a, like the value $r_a = 400$ s/m given by Stigter et al. [2018]. Because wind inputs are not always available, we use the constant value approach for r_a, however this value has to be calibrated for the model. Typical values tend to reside in the order of 5 s/m to 20 s/m.

The radiative term in Eq. (11.69) is weighted with the albedo α_s, as only the absorbed portion of the radiation counts towards evaporation/sublimation of snow. Our formulation for albedo is adapted from Greuell and Konzelmann [1994]:

$$\alpha_s[i] = \alpha_{ice} + (\rho_s - \rho_{ice})\frac{(\alpha_{nsnow} - \alpha_{ice})}{(\rho_{nsnow} - \rho_{ice})} \quad , \tag{11.70}$$

where the subscripts *ice* and *nsnow* refer to ice and new snow, respectively, and ρ_s is the density of the snow layer. We use the values $\rho_{nsnow} = 316.861$ kg m^{-1}, $\rho_{ice} = 917$ kg m^{-1}, $\alpha_{nsnow} = 0.85$ and $\alpha_{ice} = 0.58$. This albedo is used as well for the canopy snow in Eq. 10.21, but considering that the density of the canopy snow is always that of newly deposited snow, $\rho_s = \rho_{nsnow}$, which means that for the canopy snow $\alpha = \alpha_{nsnow} = 0.85$.

Because liquid water can be present in the upper snow layer, $\lambda_{vap}E_{pot,snow}$ (W m^{-2}) is used first to evaporate this water, then the used amount is subtracted from $\lambda_{sub}E_{pot,snow}$ and the difference is used to sublimate snow.

Snow evaporation/sublimation can be calculated with alternative formulas to the Penman-Monteith equation. A popular alternative is the Monin-Obukhov similarity theory, used for instance in CLM [Lawrence et al., 2019]:

$$E = -\rho_{atm}\frac{q_{atm} - q_S}{r_a} \quad , \tag{11.71}$$

where q_{atm} and q_S are the specific humidities of the atmosphere and the surface (in the case of snow, equivalent to a saturated surface). Note that in this formula there is no radiative term, and therefore it tends to underestimate the sublimation of snow.

11.6 Thawed root threshold

MAIDENiso v4 introduced a new condition used for the phenological transitions between late Winter and Spring (see chapter 14), that is that a sufficient portion of the roots must be free of ice (therefore allowing for water absorption of transpiration) before allowing the growth phase to start. For this, at the end of each day the fraction of thawed (ice-free) roots is calculated as:

$$\text{thawedroot}[i] = \sum_j \left(r_j \frac{\theta_{liq,j}}{\theta_{liq,j} - \theta_{ice,j}} \right) \quad . \tag{11.72}$$

With this, thawedroot varies within the interval [0,1]. Then the flag $\text{thaw}_{\text{flag}} = 1$ is set, indicating that the roots are "sufficiently thawed", if thawedroot $>$ $\text{thawedroot}_{\text{thres}}$, where $\text{thawedroot}_{\text{thres}} = 0.2$ is a model parameter.

Chapter 12: Soil and snow temperatures

The implementation of snow layers and ice content in the soil requires a precise thermal model, which has been adapted from CLM5 [Lawrence et al., 2019]. To be able to model the daily cycles of thawing and freezing at the beginning and at the end of winter, temperature calculations are performed in an hourly basis, indicated with the time-step integer h. Note that the thermal calculations are integrated with the hydrological calculations. Thus, at the beginning of each day i, both the hydrological and thermal calculations for $h = 1$ will be calculated before moving to $h = 2$.

Similar to surface hydrology, the model only records and writes in the outputs the history of daily values. While the daily fluxes were calculated by adding the hourly fluxes, temperatures are recorded as the final temperatures of the day (the last sub-daily time-step).

12.1 Thermal properties of soil and snow

The thermal properties of the soil are a weighted combination of their mineral and organic properties. We have no data available for the site for the fraction of soil organic matter, so a provisional value of $f_{om} = 0.0$ has been adopted.

12.1.1 Thermal conductivity

Soil thermal conductivity λ_j (W m^{-1} K^{-1}) for layer j, with $j = 1..nslay$ where $nslay = 4$ is the number of soil layers, is:

$$\lambda_j[h] = \begin{cases} K_{e,j}\lambda_{sat,j} + (1 - K_{e,j})\lambda_{dry,j} & \text{if } S_{r,j} > 10^{-7} \\ \lambda_{dry,j} & \text{if } S_{r,j} \leq 10^{-7} \end{cases}, \tag{12.1}$$

Description	Symbol	Value	Units
Thermal conductivity of solid organic matter	$\lambda_{s,om}$	0.25	W m^{-1} K^{-1}
Thermal conductivity of dry organic matter	$\lambda_{dry,om}$	0.05	W m^{-1} K^{-1}
Thermal conductivity of water	λ_{liq}	0.57	W m^{-1} K^{-1}
Thermal conductivity of ice	λ_{ice}	2.29	W m^{-1} K^{-1}
Thermal conductivity of air	λ_{air}	0.023	W m^{-1} K^{-1}
Specific heat capacity of water	C_{liq}	4188	J kg^{-1} K^{-1}
Specific heat capacity of ice	C_{ice}	2117.27	J kg^{-1} K^{-1}
Specific heat capacity of air	C_{air}	1004.64	J kg^{-1} K^{-1}
Volumetric heat capacity of organic matter	$c_{s,om}$	$2.5 \cdot 10^6$	J m^{-3} K^{-1}
Latent heat of vaporization at 0 °C	$\lambda_{vap,0}$	$2.5023 \cdot 10^6$	J kg^{-1}
Latent heat of vaporization at T	$\lambda_{vap,T}$	$\lambda_{vap,0} - \alpha_{\lambda_{vap}}$T	J kg^{-1}
Slope of $\lambda_{vap,T}$ linear dependence on T	$\alpha_{\lambda_{vap}}$	2430.54	J kg^{-1} K^{-1}
Latent heat of fusion	L_f	$3.337 \cdot 10^5$	J kg^{-1}
Latent heat of sublimation	λ_{sub}	$\lambda_{vap} + L_f$	J kg^{-1}

Table 12.1: Physical constants for thermal properties.

where $K_{e,j}$ is the Kersten number for layer j:

$$K_{e,j}[h] = \begin{cases} \log(S_{r,j}) + 1 & \text{if } T_j \geq \mathrm{T_F} \\ S_{r,j} & \text{if } T_j < \mathrm{T_F} \end{cases} \quad . \tag{12.2}$$

$\mathrm{T_F} = 0$ °C is the fusion temperature of water and $S_{r,j}$ is the degree of saturation of layer j, given by the volumetric contents of water ($\theta_{liq,j}$) and ice ($\theta_{ice,j}$) of the layer, relative to its saturation value $\theta_{sat,j}$ (evaluated at $h+\frac{1}{2}$, after solving water conduction):

$$S_{r,j}[h] = \frac{\theta_{liq,j}[h+\frac{1}{2}] + \theta_{ice,j}[h+\frac{1}{2}]}{\theta_{sat,j}} \quad . \tag{12.3}$$

The dry thermal conductivity $\lambda_{dry,j}$ is given by:

$$\lambda_{dry,j} = (1 - f_{om,j})\lambda_{dry,min,j} + f_{om,j}\lambda_{dry,om} \quad , \tag{12.4}$$

where $\lambda_{dry,om}$ is the thermal conductivity of dry organic matter (Table 12.1) and $\lambda_{dry,min,j}$ is the thermal conductivity of dry mineral soils, dependent on the bulk density $\rho_{d,j} = 2700(1 - \theta_{sat,j})$ (kg m^{-3}) as:

$$\lambda_{dry,min,j} = \frac{0.135\rho_{d,j} + 64.7}{2700 - 0.947\rho_{d,j}} \quad . \tag{12.5}$$

The saturated thermal conductivity $\lambda_{sat,j}$ is:

$$\lambda_{sat,j} = \lambda_{s,j}^{1-\theta_{sat,j}} \lambda_{liq}^{\frac{\theta_{liq,j}}{\theta_{liq,j}+\theta_{ice,j}}\theta_{sat,j}} \lambda_{ice}^{\frac{\theta_{ice,j}}{\theta_{liq,j}+\theta_{ice,j}}\theta_{sat,j}} \quad , \tag{12.6}$$

where the λ_{liq} and λ_{ice} are the thermal conductivities of water and ice (Table 12.1) and $\lambda_{s,j}$ is the thermal conductivity of soil solids:

$$\lambda_{s,j} = (1 - f_{om,j})\lambda_{s,min,j} + f_{om,j}\lambda_{s,om} \quad , \tag{12.7}$$

where $\lambda_{s,om}$ is the thermal conductivity of solid organic matter (Table 12.1) and $\lambda_{s,min,j}$ is the mineral soil solid thermal conductivity, dependent on the sand-clay composition of the soil layer:

$$\lambda_{s,min,j} = \frac{8.80(\%\text{sand})_j + 2.92(\%\text{clay})_j}{(\%\text{sand})_j + (\%\text{clay})_j} \quad . \tag{12.8}$$

The thermal conductivity of the snow layer λ_{snow} is given by:

$$\lambda_{snow}[h] = \lambda_{air} + (7.75 \cdot 10^{-5}\rho_{snow}[h+\frac{1}{2}] + 1.105 \cdot 10^{-6}\rho^2_{snow}[h+\frac{1}{2}])(\lambda_{ice} - \lambda_{air}) \quad , \tag{12.9}$$

where λ_{air} is the thermal conductivity of air (Table 12.1) and $\rho_{snow}[h + \frac{1}{2}]$ is the bulk density of snow (kg m^{-3}) evaluated at $h + \frac{1}{2}$:

$$\rho_{snow}[h] = \frac{\omega_{ice}[h + \frac{1}{2}] + \omega_{liq}[h + \frac{1}{2}]}{\Delta z_{snow}[h + \frac{1}{2}]} \quad , \tag{12.10}$$

where ω_{ice} and ω_{liq} are the solid and liquid water quantities (kg or mm of liquid water) in the snow layer, evaluated at $h + \frac{1}{2}$ (see section 11.1).

12.1.2 Heat capacity

The volumetric heat capacity c_j (J m^{-3} K^{-1}) for the soil layers is given as:

$$c_j[h] = c_{s,j}(1 - \theta_{sat,j}) + \frac{\omega_{ice,j}}{\Delta z_j}C_{ice} + \frac{\omega_{liq,j}}{\Delta z_j}C_{liq} \quad . \tag{12.11}$$

Here, C_{liq} and C_{ice} are the specific heat capacities of water and ice (Table 12.1), and $c_{s,j}$ is the volumetric heat capacity of soil solids:

$$c_{s,j} = (1 - f_{om})c_{s,min,j} + f_{om}c_{s,om} \quad , \tag{12.12}$$

where $c_{s,om}$ is the volumetric heat capacity of organic matter (Table 12.1), and $c_{s,min,j}$ is the volumetric heat capacity of mineral soil solids:

$$c_{s,min,j} = 10^{-6}\left(\frac{2.218(\%\text{sand})_j + 2.385(\%\text{clay})_j}{(\%\text{sand})_j + (\%\text{clay})_j}\right) \quad . \tag{12.13}$$

12.2 Thermal conduction

The numerical solution of thermal conduction uses a similar model to that of section 11.3 for soil hydrology. The updated temperatures of all layers are obtained simultaneously as the solution to a tridiagonal system of equations. These equations are obtained from the heat flux between layers, which are calculated from the thermal properties of the layer at time h (including the temperature $T_j[h]$.

In contrast with the numerical solution for soil hydrology, where the soil layers $j = 1..nslay$ were the only ones solved in a tridiagonal system and the snow layer was treated separately, the numerical solution for thermal conduction includes both the snow layer and the soil layers. Therefore, we now use $j = 0, 1..nslay$ with $j = 0$ referring to the snow layer, and it should be understood that quantities like λ_j, T_j and c_j with $j = 0$ are the same as λ_{snow}, T_{snow} and c_{snow}.

The first law of heat conduction is:

$$F = -\lambda \nabla T \quad . \tag{12.14}$$

This law is applied to the combined set of snow and soil layers, giving us F_j, the heat flow between layers j and $j + 1$, defined as positive upwards:

$$F_j = -\lambda_{h,j} \frac{T_j - T_{j+1}}{z_{j+1} - z_i} \quad , \tag{12.15}$$

where $j + 1$ is the layer below the layer j, and $\lambda_{h,j}$ is the thermal conductivity at the interface (mid-point) between the layers, calculated as:

$$\lambda_{h,j} = \frac{\lambda_j \lambda_{j+1}(z_{j+1} - z_j)}{\lambda_j(z_{j+1} - z_{h,j}) + \lambda_{j+1}(z_{h,j} - z_j)} \quad , \tag{12.16}$$

where $z_{h,j}$ is the depth of the interface between layers. The energy balance for layer j is:

$$\frac{c_j \Delta z_j}{\Delta t}(T_j[h + 1] - T_j[h]) = -F_{j-1} + F_j \quad , \tag{12.17}$$

where h and $h+1$ are the beginning and end of the time-step (same as the beginning of the next time-step), respectively. This equation is solved with the Crank-Nicholson method, that combines the explicit method (where the fluxes are evaluated at h, using $F_{j-1}[h]$ and $F_j[h]$) and the implicit method (where the fluxes are evaluated at $h + 1$, with $F_{j-1}[h + 1]$ and $F_j[h + 1]$):

$$\frac{c_j \Delta z_j}{\Delta t}(T_j[h+1] - T_j[h]) = \alpha(-F_{j-1}[h] + F_j[h]) + (1-\alpha)(-F_{j-1}[h+1] + F_j[h+1]) \ , \tag{12.18}$$

where $\alpha = 0.5$. This results in a tridiagonal system of equations:

$$\Lambda_{r,j} = \Lambda_{a,j} T_{j-1}[h + 1] + \Lambda_{b,j} T_j[h + 1] + \Lambda_{c,j} T_{j+1}[h + 1] \quad , \tag{12.19}$$

where $\Lambda_{a,j}$, $\Lambda_{b,j}$ and $\Lambda_{c,j}$ are the subdiagonal, diagonal and superdiagonal elements of the tridiagonal matrix, respectively, and $\Lambda_{r,j}$ is a column vector of constants. Note that we have used this notation instead of the simpler a_j, b_j, c_j and r_j to avoid confusion with other quantities like the volumetric heat capacity.

The elements of the tridiagonal matrix and the column vector are determined for each individual layer. For a generic soil layer (all except the top and bottom soil layers, $j = 2, 3 \; ... \; nslay - 1$):

$$\Lambda_{a,j} = -(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j-1}}{z_j - z_{j-1}} \quad , \tag{12.20}$$

$$\Lambda_{b,j} = 1 + (1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\left(\frac{\lambda_{h,j-1}}{z_j - z_{j-1}} + \frac{\lambda_{h,j}}{z_{j+1} - z_j}\right) \quad , \tag{12.21}$$

$$\Lambda_{c,j} = -(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j}}{z_{j+1} - z_j} \quad , \tag{12.22}$$

$$\Lambda_{r,j} = T_i[h] + \alpha\frac{\Delta t}{c_j \Delta z_j}(F_j - F_{j-1}) \quad . \tag{12.23}$$

For the bottom soil layer, the superdiagonal element is $\Lambda_{c,j} = 0$ and the bottom heat flux F_j is taken from the mean crustal heat flux in the region F_{crust} (site parameter, which is taken from a global map by Jaupart and Mareschal [2010]):

$$\Lambda_{a,j} = -(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j-1}}{z_j - z_{j-1}} \quad , \tag{12.24}$$

$$\Lambda_{b,j} = 1 + (1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j-1}}{z_j - z_{j-1}} \quad , \tag{12.25}$$

$$\Lambda_{c,j} = 0 \quad , \tag{12.26}$$

$$\Lambda_{r,j} = T_i[h] + \alpha\frac{\Delta t}{c_j \Delta z_j}(F_{crust} - F_{j-1}) \quad . \tag{12.27}$$

The top soil layer $j = 1$, can be completely or partially covered by snow or be completely uncovered, which turns it effectively into the upper layer. Thus, the elements are dynamically adapted to the situation by using f_{snow}:

$$\Lambda_{a,j} = -f_{snow}(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j-1}}{z_j - z_{j-1}} \quad , \tag{12.28}$$

$$\Lambda_{b,j} = 1 + \frac{\Delta t}{c_j \Delta z_j}\left((1-\alpha)\frac{\lambda_{h,j}}{z_{j+1} - z_j} - \frac{\partial h}{\partial T_j}\right) \quad , \tag{12.29}$$

$$\Lambda_{c,j} = -(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j}}{z_{j+1} - z_j} \quad , \tag{12.30}$$

$$\Lambda_{r,j} = T_j[h] + \frac{\Delta t}{c_j \Delta z_j}\left((1-f_{snow})(F_{atm}[h] - \frac{\partial F_{atm}}{\partial T_j}T_j[h]) + \alpha(F_j - f_{snow}F_{j-1})\right) \quad . \tag{12.31}$$

For the snow layer ($j = 0$), note that the corresponding elements are not influenced by f_{snow}, as this layer is always completely covered by the atmosphere above (thus $\Lambda_{a,j} = 0$) and while it does not necessarily fully cover the upper soil layer, all of the bottom surface of the snow layer is in contact with the upper soil layer. Thus:

$$\Lambda_{a,j} = 0 \quad , \tag{12.32}$$

$$\Lambda_{b,j} = 1 + \frac{\Delta t}{c_j \Delta z_j}\left((1-\alpha)\frac{\lambda_{h,j}}{z_{j+1} - z_j} - \frac{\partial h}{\partial T_j}\right) \quad , \tag{12.33}$$

$$\Lambda_{c,j} = -(1-\alpha)\frac{\Delta t}{c_j \Delta z_j}\frac{\lambda_{h,j}}{z_{j+1} - z_j} \quad , \tag{12.34}$$

$$\Lambda_{r,j} = T_j[h] + \frac{\Delta t}{c_j \Delta z_j}\left(F_{atm}[h] - \frac{\partial F_{atm}}{\partial T_j}T_j[h] + \alpha(F_j - F_{j-1})\right) \quad . \tag{12.35}$$

Here, F_{atm} (W m^{-2}) is the heat flux from the atmosphere, defined positive downwards:

$$F_{atm} = S - L - H - \lambda E \quad , \tag{12.36}$$

where $S[i]$ is the incoming short-wave radiation (rad$[i]$ from Eq. (9.31), see section 9.3), L is the outgoing long-wave radiation, H is the sensible heat flux, and $-\lambda E$ is the latent heat flux. L is calculated as black body radiation with the Stefan-Boltzmann law:

$$L[h] = \epsilon\sigma(T_1)^4 \quad . \tag{12.37}$$

Here, ϵ is the emissivity of the surface ($\epsilon_{soil} = 0.96$ for soil and $\epsilon_{snow} = 0.97$ for snow), $\sigma = 5.67 \cdot 10^{-8}$ W m^{-2} K^{-4} is the Stefan-Boltzmann constant, and T_1 is the temperature of the upper layer (of snow or soil). L should include a negative term accounting for the incoming longwave radiation diffused by the atmosphere, however we lack data to calculate it.

The sensible heat flux is calculated as:

$$H[h] = -\rho_{atm} C_{atm}\frac{\mathrm{T_{atm}} - T_1}{r_a} \quad , \tag{12.38}$$

where r_a (s/m) is the resistivity of the atmosphere to heat flux transfer, which we assume to be the same as the resistivity of the atmosphere to water vapour transfer (parameter of the model). The atmospheric temperature is calculated in an hourly basis using the minimum and maximum temperatures $\mathrm{T_{min}}$ and $\mathrm{T_{max}}$ of the day:

$$\mathrm{T_{atm}}[h] = \frac{\mathrm{T_{min}} + \mathrm{T_{max}}}{2} + (\mathrm{T_{max}} - \mathrm{T_{min}}) \cdot \sin\left(\pi + 2\pi\frac{t_n}{24}\right) \quad , \tag{12.39}$$

with h being the current hour of the day. This trigonometric approach makes the minimum temperature happen at 6am and the maximum temperature happen at 6pm, but could be further refined to account for the length of the day within the year in future versions of MAIDENiso.

The derivative of F_{atm} to surface temperature is:

$$\frac{\partial F_{atm}}{\partial T_1} = -\frac{\partial L}{\partial T_1} - \frac{\partial H}{\partial T_1} - \frac{\partial \lambda E}{\partial T_1} \quad , \tag{12.40}$$

$$\frac{\partial L}{\partial T_1} = 4\epsilon\sigma(T_1)^3 \quad , \tag{12.41}$$

$$\frac{\partial H}{\partial T_1} = \frac{\rho_{atm} C_{atm}}{r_a} \quad . \tag{12.42}$$

Our calculation of latent heat flux does not include a dependence on surface temperature, therefore $\frac{\partial \lambda E}{\partial T_1} = 0$.

The solution to the tridiagonal equations allows us to obtain the updated temperatures of each layer, which we denote $T_j[h + \frac{1}{2}]$ as these are still not (necessarily) the final temperatures. The last step is to apply phase changes for the water in the layers that might result from the changes in temperature.

12.3 Phase change

After the layer temperatures after thermal conduction $T_j[h + \frac{1}{2}]$ have been calculated, we can now assess the excess energy for melting/freezing of water:

$$H_j = -\frac{c_j \Delta z_j}{\Delta t}(\mathrm{T_F} - T_j[h + \frac{1}{2}]) \quad . \tag{12.43}$$

The potential mass $\Delta\omega_{pot,liq,j}$ (kg) of melted ice ($\Delta\omega_{pot,liq} > 0$) or frozen water ($\Delta\omega_{pot,liq} < 0$) is:

$$\Delta\omega_{pot,liq,j} = \frac{H_j \Delta t}{L_f} \quad . \tag{12.44}$$

Melting happens if $\omega_{ice,j} > 0$ and $H_j > 0$, in which case the ice mass that is melted $\Delta\omega_{ice,j}$ and the corresponding liquid water mass $\Delta\omega_{liq,j}$ that is added are:

$$\Delta\omega_{ice,j} = -\min(\Delta\omega_{pot,liq,j}, \omega_{ice,j}) \quad , \tag{12.45}$$

$$\Delta\omega_{liq,j} = -\Delta\omega_{ice,j} \quad . \tag{12.46}$$

While freezing requires $\omega_{liq,j} > 0$ and $H_j < 0$, in which case the liquid water mass that is frozen $\Delta\omega_{liq,j}$ and the corresponding ice mass that is added $\Delta\omega_{ice,j}$ are:

$$\Delta\omega_{liq,j} = -\min(|\Delta\omega_{pot,liq,j}|, \omega_{liq,j}) \quad , \tag{12.47}$$

$$\Delta\omega_{ice,j} = -\Delta\omega_{liq,j} \quad . \tag{12.48}$$

Regardless of whether there was melting ($\Delta\omega_{liq,j} > 0$) or freezing ($\Delta\omega_{liq,j} < 0$), the same formula is used to obtain the amount of used energy:

$$H_{used,j} = \frac{\Delta\omega_{liq,j} L_f}{\Delta t} \quad , \tag{12.49}$$

where $H_{used,j} > 0$ for melting and $H_{used,j} < 0$ for freezing. The temperature of the layer is finally updated to:

$$T_j[h+1] = \mathrm{T_F} + H_{used,j} \frac{\Delta t}{c_j \Delta z_j} \quad . \tag{12.50}$$

Note that if $H_{used,j} = H_j$, i.e. we used all the energy we had available for phase change (meaning that we were not limited by not having enough water to freeze or ice to melt) then the $T_j[h+1]$ obtained from the previous equation will inevitably be $\mathrm{T_F}$, the freezing/melting temperature. In boreal environments, this typically results in the snow/soil layers staying at $\mathrm{T_F}$ for several days in spring, as melting happens, and at fall, when freezing happens.

Additionally, when melting (not freezing) occurs for the snow layer, the thickness of this layer decreases. Before melting is applied, the density of ice in the layer is calculated as $\rho_{ice,0} = \frac{\omega_{ice,0}[h]}{\Delta z_{snow}}$ ($\omega_{ice,0}[h]$ being the ice before the melting), and the new thickness is calculated so that this density of the snow layer remains constant (as we suppose the part of the snow layer that melts does it without affecting the rest):

$$\Delta z_{snow}[h+1] = \Delta z_{snow}[h] + \frac{\Delta\omega_{ice,0}}{\rho_{ice,0}} \quad . \tag{12.51}$$

Chapter 13: Isotopes

13.1 Carbon isotopes

13.1.1 Discrimination against C isotopes

The fractionation for $\delta^{13}C$ at the leaves, $\delta^{13}C_{leaf}$, is calculated according to a discrimination model with photo-respiratory effect [Lavergne et al., 2020]:

$$\delta^{13}C_{leaf}[i] = as + (b - as) \cdot \frac{CCi_{CO2}}{CCa_{CO2}} - \frac{f\Gamma^*}{CCi_{CO2}} \quad , \tag{13.1}$$

where $as = 4.4$ stands for isotopic fractionations associated with diffusion in air, $b = 28$ is the isotopic fractionation due to carboxylation by Rubisco ($28 \pm 2‰$)), f=12 is the isotopic fractionation during photorespiration ($12 \pm 4‰$), Γ^* (Pa, see Eq. (16.6)) is the CO_2 compensation point in the absence of dark respiration. $CCi_{CO2}[i]$ (ppmV) and $CCa_{CO2}[i]$ (ppmV) are the concentrations of CO_2 in the stomata and the atmosphere, respectively. The latter is an input to MAIDENiso, that has to be given daily, while the former is calculated as:

$$CCi_{CO2}[i] = C_{i,sun} \frac{10^6 \text{Pa ppmV}^{-1}}{P_{atm}} \quad , \tag{13.2}$$

13.1.2 Isotopic composition of carbon stored

If carbon was stored during the time-step i, i.e. $\Delta CStore[i] > 0$, then the $\delta^{13}C$ for the early photosynthetates, $\delta^{13}C_{photo}$ is:

$$\delta^{13}C_{photo}[i] = \frac{\delta^{13}C_{CO2} - \delta^{13}C_{leaf}}{1 + \frac{\delta^{13}C_{leaf}}{1000}} \quad , \tag{13.3}$$

where $\delta^{13}C_{CO2}$ is the $\delta^{13}C$ of the atmospheric CO_2, which is an input to the model. With this, the $\delta^{13}C$ of the carbon in the store $\delta^{13}C_{store}$ is updated as:

$$\delta^{13}C_{store}[i] = \frac{\delta^{13}C_{store}[i-1]() + \delta^{13}C_{photo} CStem_{in}[i-1]}{C_{store}} \quad . \tag{13.4}$$

If no carbon was stored during the time-step i, i.e. $\Delta\text{CStore}[i] = 0$, then $\delta^{13}\text{C}_{\text{store}}$ simply is carried over from the previous time-step:

$$\delta^{13}\text{C}_{\text{store}}[i] = \delta^{13}\text{C}_{\text{store}}[i-1] \quad . \tag{13.5}$$

13.1.3 Isotopic composition of tree rings

In the general case, this is if $\text{CStore}_{\text{out}}[i-1] > 0$ and $\text{NPP}[i-1] > 0$, then we have two sources of carbon to the tree-rings. In this case, $\delta^{13}\text{C}_{\text{TRC}}$ is calculated as the weighted sum of the δ^{13}C of these two sources, with the C quantities of each source being the weights:

$$\delta^{13}\text{C}_{\text{TRC}}[i] = \frac{\text{NPP}[i-1](\delta^{13}\text{C}_{\text{photo}} + \text{Dc}_{\text{p}}) + \text{CStore}_{\text{out}}[i-1](\delta^{13}\text{C}_{\text{store}}[i] + \text{Dst}_{\text{dest}} + \text{Dc}_{\text{p}})}{\text{NPP}[i-1] + \text{CStore}_{\text{out}}[i-1]} \quad , \tag{13.6}$$

where Dst_{dest} (‰) is the ^{13}C fractionation induced by destocking of storage carbon as starch and Dc_{p} (‰) is the difference between δ^{13}C in cellulose and in early photosynthetates.

If $\text{CStore}_{\text{out}}[i-1] = 0$, then we simply add the post-photosynthetic fractionation between leaf and tree-ring wood (cellulose or bulk wood):

$$\delta^{13}\text{C}_{\text{TRC}}[i] = \delta^{13}\text{C}_{\text{photo}} + \text{Dc}_{\text{p}} \quad . \tag{13.7}$$

On the other hand, if $\text{NPP}[i-1] = 0$, then the general expression is reduced to:

$$\delta^{13}\text{C}_{\text{TRC}}[i] = \delta^{13}\text{C}_{\text{photo}} + \text{Dst}_{\text{dest}} + \text{Dc}_{\text{p}} \quad . \tag{13.8}$$

The daily δ^{13}C is then added for the whole year, weighted by the daily increments of stem carbon, to obtain the δ^{13}C of the tree ring produced during that year, $\delta^{13}\text{C}_{\text{TRC,yr}}$

$$\delta^{13}\text{C}_{\text{TRC,yr}}[y] = \frac{\sum_i \delta^{13}\text{C}_{\text{TRC}}[i]\text{CStem}_{\text{in}}[i-1]}{\sum_i \text{CStem}_{\text{in}}[i]} \quad . \tag{13.9}$$

13.2 Water isotopes

Water isotopes are tracked through the hydrological cycle, until they are absorbed by the roots and incorporated to the wood structure. The introduction of snow in MAIDENiso means introducing new parametrizations and processes.

13.2.1 Isotopic composition of precipitation

Danis et al. [2012] developed a statistical model to estimate daily $\delta^{18}O$ (‰) in precipitation ($\delta^{18}O_P$) based on daily values of mean air temperature T_{air} (Celsius) and precipitation P (mm).

$$\delta^{18}O_P = a\ T_{air} + b\ P + c \quad , \tag{13.10}$$

where a (‰ K^{-1}), b (‰ mm) and c (‰) are the coefficients of the regression model, which are site parameters. To obtain these coefficients, the regression model in Eq. (13.10) has to be applied first to daily data of $\delta^{18}O_P$, T_{air} and P in the site were the model is to be run, or (under reasonable assumption) in a nearby place.

After the addition of snow hydrology with MAIDENiso v4, the study of regression curves at boreal sites showed that there was a clear differentiation between the regression models in Eq. (13.10) when discriminating rainfall and snowfall events. The isotopic composition of snowfall is distinct from that of rainfall because, unlike liquid rainfall, the solid precipitation does not exchange isotopes with the atmosphere and conserves the isotopic composition it had when formed in the cloud [Gat, 1996]. This motivated the modification of the original formulation into two different regression models:

$$\delta^{18}O_{rain} = a_{rain}\ T_{air} + b_{rain}\ P + c_{rain} \quad , \tag{13.11}$$

$$\delta^{18}O_{snow} = a_{snow}\ T_{air} + b_{snow}\ P + c_{snow} \quad , \tag{13.12}$$

The new model therefore requires 6 instead of 3 regression parameters, but the method to obtain them is almost identical to that of the previous versions, using daily data of $\delta^{18}O_P$. Simply, the coefficients a_{rain}, b_{rain} and c_{rain} have to be obtained from precipitation events of pure rainfall (identifiable as such by the air temperature) while a_{snow}, b_{snow} and c_{snow} have to be obtained from precipitation events of pure snowfall.

Once the regression coefficients in Eqs. (13.11), (13.12) are obtained for the study site, the model estimates the deuterium composition of precipitation, δ^2H_P, from $\delta^{18}O_P$ using a meteoric water line:

$$\delta^2H_{rain} = 6.8959\ \delta^{18}O_{rain} + 0.6995 \quad . \tag{13.13}$$

$$\delta^2H_{snow} = 6.8959\ \delta^{18}O_{snow} + 0.6995 \quad . \tag{13.14}$$

However, these values are from a local meteoric water line somewhere in France, and were never changed since the first version of the model as δ^2H outputs were never really used. Future versions of MAIDENiso can be expected to substitute these fixed values by configurable parameters.

13.2.2 Isotopic mixing

In MAIDENiso, there are two processes that can change the isotopic composition of water pools (e.g. the canopy ice, the liquid water in layer 1, the ice in layer 1...) in

the model: mixing and fractionation. Mixing is calculated in parallel to hydrological calculations. The way it works is simple. Let us denote by X a generic pool of water (in mm), and denote by $q_{X,in}$ and $q_{X,out}$ the influx to and the outflux from the layer, respectively (both in mm/s). Let us also call $\delta^{18}O_X$ to the isotopic composition of oxygen-18 of the pool X (the same goes for deuterium as we do for oxygen-18), and call $\delta^{18}O_{q_{X,in}}$ and $\delta^{18}O_{q_{X,out}}$ to the isotopic composition of the water carried in the fluxes $q_{X,in}$ and $q_{X,out}$. In this way, we have:

$$X[n+1] = X[n] + (q_{X,in}[n] - q_{X,out}[n])\Delta t \quad , \tag{13.15}$$

where n is an integer to indicate a generic time-step of Δt (s), as this is done for quantities that are calculated daily and hourly.

By definition, $q_{X,out}$ is a flux that originates from X, so independently to its destination, the associated oxygen-18 isotopic composition of this water flux is $\delta^{18}O_{q_{X,out}} = \delta^{18}O_X$.

Meanwhile, the $q_{X,in}$ can have a variety of sources, which can either be the outflux $q_{Z,out}$ from another pool Z or being defined as influxes to the model q_I (thought currently this is the case solely for precipitation):

$$q_{X,in}[n] = \sum_Z q_{Z,out}[n] + \sum_I q_I[n] \quad , \tag{13.16}$$

The isotopic composition of $q_{Z,out}$ is known as it is the isotopic composition of the source, while the isotopic composition of q_I, $\delta^{18}O_{q_I}$, has to be given outside of the model (as it is the case for precipitation, its isotopic composition has to be given either as a model that allows to calculate $\delta^{18}O_P$ daily, or the values of $\delta^{18}O_P$ have to be given directly as an input). For the sake of simplicity, let us rewrite the previous equation with the different fluxes that make up the influx being called generically q_Z:

$$q_{X,in}[n] = \sum_Z q_Z[n] \quad . \tag{13.17}$$

Therefore, the isotopic composition after mixing ($X[n+1]$) will be obtained as the weighted sum of the isotopic compositions $\delta^{18}O_X$ and $\delta^{18}O_{q_Z}$ ($\forall\ Z$), with the masses of water $X - q_{X,out}\Delta t$ and $q_Z\Delta t$ ($\forall\ Z$) being the weights:

$$\delta^{18}O_X[n+1] = \frac{\delta^{18}O_X(X - q_{X,out}\Delta t) + \sum_Z(\delta^{18}O_{q_Z} q_Z \Delta t)}{X - q_{X,out}\Delta t + \sum_Z(q_Z\Delta t)} \quad , \tag{13.18}$$

with all quantities (pools, fluxes and isotopic compositions) at the right side of this equation being evaluated at time-step n. While this expression looks complex, in practice q_I only exist for precipitation, and most water pools receive water directly from a very low number of sources.

The same logic applies in the situation when two or more fluxes q_Z join together into a flux q_J, rather than a pool. The equation reads the same as Eq. (13.18) when making $X - q_{X,out}\Delta t = 0$:

$$\delta^{18}O_{q_J} = \frac{\sum_Z(\delta^{18}O_{q_Z} q_Z \Delta t)}{\sum_Z(q_Z \Delta t)} \quad , \tag{13.19}$$

Another possible situation is when a flux is divided without being deposited into a water pool first, as it is the case for precipitation being separated into intercepted precipitation and direct precipitation (see section 10.2). In this case, the branching fluxes have the same isotopic composition as the original flux.

13.2.3 Fractionation processes

13.2.3.1 Evaporative fractionation

Evaporation of water in the soil and the canopy produces fractionation, as lighter isotopes are evaporated preferentially, leading to higher concentrations of heavy isotopes in the remaining water. In MAIDENiso, the evaporative fractionation is described with the Craig-Gordon model [Craig and Gordon, 1965] implemented following the formulation of Gibson et al. [2008], where the isotopic composition of the evaporative flux δ_E is given as:

$$\delta_E = \frac{1}{1 - h + \epsilon_K}\left(\frac{\delta_L - \epsilon^+}{\alpha^+} - h\delta_A - \epsilon_K\right) \quad , \tag{13.20}$$

where δ_L and δ_A are respectively the isotopic composition of the remaining water and of the atmospheric vapour, h is the relative humidity of the atmosphere, $\epsilon_K = nC_K^0\theta(1-h)$, $\theta = (1-h')/(1-h)$ being h' the adjusted atmospheric humidity following admixture of evaporating moisture, C_K^0 is 25‰ for deuterium and 28.6‰ for oxygen-18, $n = 1/2$ for open water bodies and $n = 1$ for soil water and canopy, and $\epsilon^+ = \alpha^+ - 1$ is the equilibrium isotopic separation between liquid and vapour. α^+, the liquid-vapour equilibrium isotopic fractionation, was estimated from empirical relations by Horita and Wesolowski [1994]:

$$10^3 \ln \alpha^+_{^{18}O} = -7.685 + 6.7123(10^3/T) - 1.6664(10^6/T^2) + 0.35041(10^9/T^3) \tag{13.21}$$

$$10^3 \ln \alpha^+_{^{2}H} = 1158.8(T^3/10^9) - 1620.1(T^2/10^6) + 794.84(T/10^3) - 161.04 + 2.9992(10^3/T) \tag{13.22}$$

Fractionation from evaporation in the canopy and the ground, however, was not active in versions previous to MAIDENiso v4. Danis et al. [2012] mentions that direct observations of oxygen-18 isotopic composition $\delta^{18}O_{SW}$ in their work site revealed no that it was not significantly affected by soil evaporation, therefore they considered that

evaporation happened to water pockets and puddles isolated from the soil water, and no fractionation occurred as these evaporated completely. Activation of fractionation resulted in numerical errors, because of an error in the implementation of Eq. (13.20), where instead of the isotopic composition of the remaining water after evaporation δ_L, it was used that of the water source before evaporation δ_W. The problem to correctly implement Eq. (13.20) is that MAIDENiso does not know δ_L before it is calculated from δ_W and δ_E:

$$L\delta_L = W\delta_W - E\delta_E \quad , \tag{13.23}$$

where E is the evaporated water and W and $L = W - E$ are the water quantities of the evaporation source before and after the evaporation has taken place, respectively. We can join Eqs. (13.23) and (13.20) to obtain a system of two equations with two unknown variables δ_L and δ_E. Substitution of δ_E in Eq. (13.20) yields δ_L as a function of δ_W and the water quantities W and E as:

$$\delta_L = \frac{1}{W - E + \frac{E}{(1-h+\epsilon_K)\alpha^+}} \left[\delta_W W - \frac{E}{(1-h+\epsilon_K)} \left(-\frac{\epsilon^+}{\alpha^+} - h\delta_A - \epsilon_K \right) \right] \quad . \tag{13.24}$$

Once δ_L is obtained, we can use either Eq. (13.20) or Eq. (13.23) to obtain δ_E.

13.2.3.2 Sublimative fractionation

The incorporation of snow into the model implies that sublimative fractionation has to be implemented as well. As snow remains exposed to the atmosphere for long periods of time, sublimation is bound to have a much more significant effect on isotopic concentration than evaporation. The isotopic composition of the sublimative flux is calculated through the Craig-Gordon equation (13.20), changing some parameters respect to evaporation. While we still use $n = 1$ for snow in the canopy, the snow over the ground is fully exposed to the air, and therefore we use $n = 1/2$ for it (evaporative flux from liquid water in the snow layer also uses $n = 1/2$). More importantly, the solid-vapour equilibrium isotopic fractionation α^+ is different than that for liquid-vapour fractionation. Ellehoj et al. [2013] measured α^+ between 0°C and −40°C and extrapolated the following dependencies on temperature:

$$\ln \alpha^+_{^{18}O} = 0.0831 - \frac{49.192}{T} + \frac{8312.5}{T^2} \quad , \tag{13.25}$$

$$\ln \alpha^+_{^{2}H} = 0.2133 - \frac{203.10}{T} + \frac{48888}{T^2} \quad . \tag{13.26}$$

Other than α^+ and n, the Craig-Gordon Equation remains the same for sublimation as for evaporation.

13.2.3.3 Melting and freezing fractionation

Following the original work by O'Neil [1968], the fractionation factor α^+_{s-l} between ice and water is defined by the isotopic ratios between the newly formed ice R_{ice} and the bulk water R_{water}:

$$\alpha^+_{s-l} = \frac{R_{ice}}{R_{water}} = \frac{1 + \delta_{ice}/1000}{1 + \delta_{water}/1000} \quad . \tag{13.27}$$

O'Neil [1968] measured the ice-water fractionation factor for oxygen-18 and deuterium from two samples, obtaining $\alpha^+_{s-l,^{18}O} = 1.0029$ and $\alpha^+_{s-l,^{2}H} = 1.0178$ for the first sample and $\alpha^+_{s-l,^{18}O} = 1.0031$ and $\alpha^+_{s-l,^{2}H} = 1.0195$ for the second one. A more recent measurement by Lehmann and Siegenthaler [1991] yields $\alpha^+_{s-l,^{18}O} = 1.00291 \pm 0.00003$ and $\alpha^+_{s-l,^{2}H} = 1.0212 \pm 0.0004$. From Eq. (13.27), we obtain the isotopic composition of the newly formed ice as:

$$\delta_{ice} = \alpha^+_{s-l}(1000 + \delta_{water}) - 1000 \quad . \tag{13.28}$$

However, we have to take into account that this is a Rayleigh process. As the source water depletes, its isotopic composition changes and therefore it changes the composition of the new ice:

$$\delta_{ice} = \alpha^+_{s-l}(1000 + \delta_{water}) f^{\alpha^+_{s-l}-1} - 1000 \quad , \tag{13.29}$$

where $f = \frac{W-Y}{W}$ is the remaining fraction of the source water (whose isotopic composition was δ_{water}), W being the quantity of source water and Y being the quantity of this water that freezes. If we integrate this equation, we will obtain the composition of the newly formed ice as a function of the new ice:

$$\delta_{ice} = \frac{W}{Y}\left(\delta_{water} - (1000 + \delta_{water})\left(\frac{W-Y}{W}\right)^{\alpha^+_{s-l}} + 1000\frac{W-Y}{W}\right) \quad . \tag{13.30}$$

Freezing produces fractionation, because heavier water molecules freeze easier. However, no fractionation occurs during melting, because melting happens layer by layer and individual ice crystals melt completely. Therefore, the isotopic composition of the melted water is considered identical to that of the source ice/snow.

13.2.4 Tree-ring cellulose

The δ^{18}O that is incorporated to the tree-ring cellulose in one particular day depends directly on the δ^{18}O of the xylem water, $\delta^{18}O_{XW}$. This is taken from the δ^{18}O composition of each soil layer ($\delta^{18}O_{soil,j}$), proportionally to their root fractions (r_j):

$$\delta^{18}O_{XW}[i] = \sum_j + r_j \delta^{18}O_{soil,j} \quad . \tag{13.31}$$

The oxygen-18 composition for tree-ring cellulose during day i is calculated as:

$$\delta^{18}\mathrm{O}_{\mathrm{TRC}}[i] = (1 - f_0) \cdot \delta^{18}\mathrm{O}_{\mathrm{leaf}} + f_0 \cdot \delta^{18}\mathrm{O}_{\mathrm{XW}} + \epsilon_0 \quad , \tag{13.32}$$

with $\delta^{18}\mathrm{O}_{\mathrm{leaf}}$ being the δ^{18}O at leaf level:

$$\delta^{18}\mathrm{O}_{\mathrm{leaf}}[i] = \epsilon^* + \epsilon_k \cdot (1 - \mathrm{RH}) + \mathrm{RH} \cdot \delta^{18}\mathrm{O}_{\mathrm{V}} + (1 - \mathrm{RH}) \cdot \delta^{18}\mathrm{O}_{\mathrm{XW}} \quad . \tag{13.33}$$

Here, f_0 (unitless) is the dampening factor reflecting the exchange of the oxygen atoms between sucrose and xylem water during the synthesis of cellulose in the xylem cells of the tree rings, typically within a range of 0.4-0.5 [Roden and Ehleringer, 2000, Saurer et al., 1997, Sternberg et al., 1986, Yakir, 1992]. ϵ_0 is the biochemical fractionation due to oxygen exchange between water and the carbonyl groups (C=O) in the organic molecules, undetermined but expected in a range of 24-30 ‰ [DeNiro and Epstein, 1979, Farquhar et al., 1998]. ϵ_k is the kinetic fractionation due to the diffusion of vapour into unsaturated air through the stomata and the leaf boundary layer, set to 26.5 ‰ in Farquhar et al. [1989], but we consider it undetermined as it can vary over larger ranges [Buhay et al., 1996]. ϵ^* is the equilibrium fractionation due to the change of phase of water from liquid to vapour at leaf temperature (fixed at 21.4 °C, which is the temperature threshold for maximum carbon assimilation), with a value of 9.65 ‰ [Helliker and Richter, 2008]:

$$\epsilon^* = 10^3 \exp\left(\frac{-7.685 + \frac{6.7123 \cdot 10^3}{\mathrm{T}_{\mathrm{leaf}}} - \frac{1.6664 \cdot 10^6}{{\mathrm{T}_{\mathrm{leaf}}}^2} + \frac{0.35041 \cdot 10^9}{{\mathrm{T}_{\mathrm{leaf}}}^3}}{10^3} - 1\right) \quad . \tag{13.34}$$

Following Linacre [1964], and in the absence of a species-specific equation, leaf temperature is estimated as a linear function of air temperature:

$$\mathrm{T}_{\mathrm{leaf}} = \frac{23}{33}\mathrm{T}_{\mathrm{air}} + 10 \quad , \tag{13.35}$$

where constants are empirically-determined [Evans, 2007]. However, Helliker and Richter [2008] showed a remarkably constant leaf temperature of $\mathrm{T}_{\mathrm{leaf}} = 21.4 \pm 2.2$°C across 50° of latitude, from subtropical to boreal biomes. Inputing this temperature in Eq. (13.34) gives $\epsilon^* = 9.65$ ‰.

Back to Eq. (13.33), RH is the relative humidity (see section 8.4). $\delta^{18}\mathrm{O}_{\mathrm{V}}$ is the δ^{18}O of vapour, which is calculated from $\delta^{18}\mathrm{O}_{\mathrm{P}}$ and the fractionation due to the phase change from liquid water to vapour at mean air temperature, $\epsilon^*_{\mathrm{T}_{\mathrm{air}}}$ [Horita and Wesolowski, 1994]:

$$\delta^{18}\mathrm{O}_{\mathrm{V}}[i] = \delta^{18}\mathrm{O}_{\mathrm{P}} - \epsilon^*_{\mathrm{T}_{\mathrm{air}}} \quad . \tag{13.36}$$

The $\delta^{18}\mathrm{O}_{\mathrm{TRC}}$ time series produced through Eq. (13.32) are daily, while the $\delta^{18}\mathrm{O}_{\mathrm{TRC,yr}}$ measured from tree rings is commonly annually resolved, or occasionally with intra-annual resolution (e.g. Szejner et al. [2018]). To produce a yearly record

comparable with observations, the daily series have to be weighted. We have two possibilities to weight the daily $\delta^{18}O_{TRC}$. The first, used in Lavergne et al. [2017] and Hermoso de Mendoza et al. [2022] assumes that allocation of carbon to the trunk is proportional to daily GPP:

$$\delta^{18}O_{TRC,yr}[y] = \frac{\sum_i \delta^{18}O_{TRC}[i] \cdot GPP[i]}{\sum_i GPP[i]} \quad . \tag{13.37}$$

The second possibility, used in Gennaretti et al. [2017], is to weight daily $\delta^{18}O_{TRC}$ with the daily allocation of C to the stem $CStem_{in}[i]$ (gC $m^{-2}d^{-1}$, see section 15.2):

$$\delta^{18}O_{TRC,yr}[y] = \frac{\sum_i \delta^{18}O_{TRC}[i] \cdot CStem_{in}[i]}{\sum_i CStem_{in}[i]} \quad . \tag{13.38}$$

It should be obvious that Eq. (13.38) is the right way to weight daily $\delta^{18}O_{TRC}$. However, the allocation of carbon to the different parts of the tree (see Chapter 15 requires to calibrate the allocation model at the site. If it is not possible to calibrate the allocation model reliably, it is more convenient to use Eq. (13.37). Therefore, the use of Eq. (13.37) or Eq. (13.38) is subject to the circumstances.

Chapter 14: Phenology

14.1 Phenology phases and allocation periods

There are two types of phases or chronological periods in MAIDENiso that should not be confused: phenology and carbon allocation. To avoid a confusion between the two, we use the term "phase" when talking about phenology, and the term "period" when talking about carbon allocation. Phenological phases and carbon allocation periods tend to be coincidental, since the transitions between carbon allocation periods (except for the transition between Spring and Summer) is triggered by the transitions of phenological phases. A schematic of these two chronologies is shown in Fig. 14.1.

	Early Winter	Late Winter	Spring / budburst	Summer	Fall	Early Winter
Phenology phase	1	2	3		4	1
Allocation period	0		1	2	3	0

Figure 14.1: Phenology phases and carbon allocation periods

14.2 Phenological phases

The phenological phases in MAIDENiso are the following [Gea-Izquierdo et al., 2015]:

- Phase 1: **Early Winter phase**. This is the phase entered after fall, without growth and without the accumulation of growth degree days (gdd).
- Phase 2: **Late Winter phase**. In this winter phase the accumulation of growth degree days is active. This is a cumulative quantity that triggers the growth phase after reaching a threshold.
- Phase 3: **Growth phase or Spring-Summer phase**. This phase has both Spring and Summer, though these are treated as separate carbon allocation periods.

- Phase 4: **Fall**. A short phase during which growth stops but photosynthesis doesn't. It is coincidental with the carbon allocation Fall period.

14.2.1 Phase transitions

The transitions between these phenological phases is determined mainly by meteorological and radiative triggers (length of day), but influenced by a set of model parameters. These are:

- vegphase12 : number of days that must be elapsed since the beginning of the current year to change from Phase 1 to Phase 2.
- vegphase23 : threshold of days that Phase 2 can last until Phase 3.
- gdd_thres: number of growing degree days that must be reached to trigger the Spring phenological phase.
- $\text{thawedroot}_{\text{thres}}$: threshold of proportion of roots that must be thawed to trigger the Spring phenological phase (see section 11.6).
- phenolcount: number of days that must be elapsed since the beginning of the fall phenophase to trigger the winter phenophases.

14.2.1.1 From Early Winter (phase 1) to Late Winter (phase 2)

To transition from Early to Late Winter, therefore activating the accumulation of gdd, the current day of the year (DOY) must overtake the value of the parameter vegphase12. There are no other triggers or conditions.

14.2.1.2 From Late Winter (phase 2) to Spring-Summer (phase 3)

The Spring-Summer phase comes in MAIDEN when both of these conditions are met:

- The DOY has reached the smoothed estimated transition day day23_sp.
- The roots are considered to be sufficiently ice-free, i.e. the flag $\text{thaw}_{\text{flag}}$ has a value of 1 (see section 11.6).

The yearly values day23_sp[y] are the spline-smoothed version of the yearly values day23[y], which are calculated previous to the main MAIDENiso loop. Each year, day23 is calculated as the first DOY in which:

- The DOY reaches the threshold vegphase23, OR

- The daily accumulation of growing degree days reaches the threshold gdd_thres.

This daily accumulation of growing degree days (gdd, °C) is the sum of the average daily air temperatures T_{avg} above a temperature threshold parameter $T_{thres} = 3°C$, this is:

$$\text{gdd}[i] = \begin{cases} \text{gdd}[i-1] + T_{avg} - T_{thres} & \text{if } T_{avg} > T_{thres} \ \& \ \text{vegphase} = 2 \\ \text{gdd}[i-1] & \text{otherwise} \end{cases}, \quad (14.1)$$

14.2.1.3 From Spring-Summer (phase 3) to Fall (phase 4)

The fall comes in MAIDEN when one of the following conditions is met:

- The daily photoperiod (daylight hours) falls below the photoperiod threshold $\text{photoperiod}_{thres}$ AND the photoperiod is decreasing ($\text{photoperiod}[i] < \text{photoperiod}[i-1]$), OR
- The DOY has reached day34_{min} AND the $T_{min} < 0°C$, OR
- The DOY has reached day34_{max}.

With $\text{day34}_{min} = 273$ (September 30th) for Mediterranean and $\text{day34}_{min} = 244$ (September 1st) for Boreal forests, while $\text{day34}_{max} = 350$ for both.

The additional condition that the photoperiod must be currently decreasing was added in MAIDENiso v4. This solved the unintended effect by which the Fall phase would trigger immediately after the start of the Spring-Summer phase, if the Spring-Summer phase started so early in the year that the photoperiod was still below $\text{photoperiod}_{thres}$.

14.2.1.4 From Fall (phase 4) to Early Winter (phase 1)

Once the Fall phase has started, a countdown starts for the start of the Early Winter phase. This countdown is controlled by the parameter phenolcount and it typically has a value of 9 days. This means that the Fall period always lasts the same number of days, every year.

Chapter 15: Carbon allocation

15.1 Autotrophic respiration and NPP

Gross Primary Production (GPP, occasionally compared to Gross Ecosystem Production or GEP) is determined in the photosynthesis module. GPP is first divided into autotrophic respiration AR and net primary production (NPP). In MAIDEN, this partition is determined by a fixed parameter $\mathrm{npp}_{\mathrm{frac}}$.

$$\mathrm{NPP}[i] = \mathrm{npp}_{\mathrm{frac}}\ \mathrm{GPP}[i] \quad . \tag{15.1}$$

$$\mathrm{AR}[i] = (1 - \mathrm{npp}_{\mathrm{frac}})\ \mathrm{GPP}[i] \quad . \tag{15.2}$$

This parameter is fixed at a value $\mathrm{npp}_{\mathrm{frac}} = 0.47$. This means that NPP, daily or yearly, is always 47% of the GPP.

15.2 Tree carbon pools

There are 4 carbon reservoirs defined for the main parts of the tree. These 4 reservoirs are:

- Leaf.
- Stem (defined as trunk wood + branch + bark).
- Root.
- Store.

For each reservoir, variables are defined daily for input (e.g. $\mathrm{CStore}_{\mathrm{in}}[i]$), output ($\mathrm{CStore}_{\mathrm{out}}[i]$), and net change ($\Delta\mathrm{CStore}[i] = \mathrm{CStore}_{\mathrm{in}}[i] - \mathrm{CStore}_{\mathrm{out}}[i]$). The stem does not decrease over time, however the variables for this reservoir are still defined:

$$\mathrm{CStem}_{\mathrm{out}}[i] = 0 \quad . \tag{15.3}$$

Each day, the C contents of the reservoirs are updated as:

$$\mathrm{CLeaf}[i+1] = \mathrm{CLeaf}[i] + \Delta\mathrm{CLeaf}[i] \quad , \tag{15.4}$$

$$\mathrm{CStem}[i+1] = \mathrm{CStem}[i] + \Delta\mathrm{CStem}[i] \quad , \tag{15.5}$$

$$\mathrm{CRoot}[i+1] = \mathrm{CRoot}[i] + \Delta\mathrm{CRoot}[i] \quad , \tag{15.6}$$

$$\mathrm{CStore}[i+1] = \mathrm{CStore}[i] + \Delta\mathrm{CStore}[i] \quad , \tag{15.7}$$

with:

$$\Delta\mathrm{CLeaf}[i] = \mathrm{CLeaf}_{\mathrm{in}}[i] - \mathrm{CLeaf}_{\mathrm{out}}[i] \quad , \tag{15.8}$$

$$\Delta\mathrm{CStem}[i] = \mathrm{CStem}_{\mathrm{in}}[i] - \mathrm{CStem}_{\mathrm{out}}[i] \quad , \tag{15.9}$$

$$\Delta\mathrm{CRoot}[i] = \mathrm{CRoot}_{\mathrm{in}}[i] - \mathrm{CRoot}_{\mathrm{out}}[i] \quad , \tag{15.10}$$

$$\Delta\mathrm{CStore}[i] = \mathrm{CStore}_{\mathrm{in}}[i] - \mathrm{CStore}_{\mathrm{out}}[i] \quad , \tag{15.11}$$

15.2.1 Leaf Area Index

Every day, the leaf area index (LAI) is calculated, in the case of the mediterranean forest, as:

$$\mathrm{LAI}[i] = \frac{\mathrm{CLeaf}[i]}{\frac{1}{\mathrm{SLA}}} \cdot \mathrm{SLA} \quad , \tag{15.12}$$

and in the case of the boreal forest, as:

$$\mathrm{LAI}[i] = \frac{\mathrm{CLeaf}[i]}{\frac{\mathrm{percCLeaf}}{\mathrm{SLA}}} \cdot \mathrm{SLA} \quad , \tag{15.13}$$

where SLA is the specific leaf area (leaf area per unit of mass), and percCLeaf is the fraction of carbon content in the leaves, both of them model parameters (we use $\mathrm{SLA} = 0.004234$ m^2 g^{-1} and $\mathrm{percCLeaf} = 0.49$ for black spruce).

15.3 Canopy target

At the beginning of the year, the tree estimates the how much carbon it will need in the new year, based on a target of leaves and roots. The excess carbon will be later used in storage and growing the stem. The maximum amount of carbon that can be contained in the canopy, MaxCcanopy, is:

$$\mathrm{MaxCcanopy} = \frac{\mathrm{LAI}_{\mathrm{max}}}{\mathrm{SLA}} \mathrm{percCLeaf} \quad , \tag{15.14}$$

where SLA is the specific leaf area (leaf area per unit of mass) and $\mathrm{LAI_{max}}$ is the maximum LAI, all of which are parameters determined by the tree species (for boreal, we use SLA = 0.004234, percCLeaf = 0.49 and $\mathrm{LAI_{max}} = 3.3$). MaxCcanopy is directly proportional to $\mathrm{LAI_{max}}$, however the LAI will not always be the maximum possible. The tree loses leaves during the year and has to produce new ones. In general, the amount of leaf carbon targeted by the tree or canopy target, Cmax, is not going to be equal to MaxCcanopy. The Cmax for each year is calculated the first day of that year, but different models can be used to estimate Cmax.

Mediterranean trees use one model to calculate the canopy target Cmax [Gea-Izquierdo et al., 2015], while there are three different models available for boreal trees. The original model for boreal trees was designed in Gennaretti et al. [2017] and use in the latter publications [Lavergne et al., 2017, Hermoso de Mendoza et al., 2022], however this model depends on meteorological statistics over the entire simulation period to set the canopy target for any given year, which does not allow to use this model in some circumstances. Two alternative models have been designed in order to provide an alternative method of calculating a canopy target that is independent of future years.

15.3.1 Mediterranean model

For Mediterranean trees, we use the model developed by Gea-Izquierdo et al. [2015]. In this model, the canopy target for the new year y is calculated as:

$$\mathrm{Cmax}[y] = (1 - \mathrm{Cmax_{slope}} \frac{\mathrm{LAI_{perc}} - \mathrm{H_{stress}}}{\mathrm{LAI_{perc}}}) \cdot \mathrm{MaxCcanopy} \quad , \tag{15.15}$$

where $\mathrm{Cmax_{slope}}$ is a parameter that determines the slope of Cmax, $\mathrm{LAI_{perc}} = 250$ is a parameter to compute the yearly carbon that will be allocated to leaf and root as a function of the previous year soil moisture. The hydraulic stress of the previous year, $\mathrm{H_{stress}}$, is the average of the soil water content (SWC) of the last 250 days of the previous year ($\mathrm{H_{stress}} = 0$ for the first year):

$$\mathrm{H_{stress}} = \frac{1}{250} \sum_{j=1}^{j=250} SWC[i-j] \quad , \tag{15.16}$$

being i the first day of the corresponding year.

Cmax is only allowed to vary within the interval $\mathrm{Cmax} = [0.7, 1]\mathrm{MaxCcanopy}$. An addition posterior to Gea-Izquierdo et al. [2015] is that if Cmax is lower than the actual amount of carbon in the canopy at the time it is calculated (CLeaf at the first day of the year), Cmax is increased by 30 gC.

15.3.2 Boreal model

In the allocation model for boreal forest created by Gennaretti et al. [2017] the amount of canopy carbon targeted by the tree, Cmax, is set to vary within 70%-100% of the maximum:

$$\mathrm{Cmax}[y] = \mathrm{MaxCcanopy} \cdot (0.7 + 0.3 \cdot \mathrm{CanopyMult}) \quad . \tag{15.17}$$

The 30% that Cmax can vary is controlled by a factor CanopyMult that represents the overall climate dependence:

$$\mathrm{CanopyMult}[y] = \frac{1}{1 + \exp(\mathrm{CanopyT} \cdot \mathrm{Temp}_{\mathrm{yr}-1})} \cdot \frac{1}{1 + \exp(\mathrm{CanopyP} \cdot \mathrm{P}_{\mathrm{spring}})} \quad , \tag{15.18}$$

where $\mathrm{Temp}_{\mathrm{yr}-1}$ is the previous-year mean July–August temperature (detrended and transformed to z scores), $\mathrm{P}_{\mathrm{spring}}$ is the previous-year April precipitation (detrended and transformed to z scores), and CanopyT and CanopyP are two parameters to be optimized and that represent the slopes of the relationships between CanopyMult and $\mathrm{Temp}_{\mathrm{yr}-1}$ or $\mathrm{P}_{\mathrm{spring}}$, respectively.

This boreal allocation model has been applied with great success to boreal forests [Gennaretti et al., 2017, Lavergne et al., 2017, Hermoso de Mendoza et al., 2022]. However, a weakness arises from the need to detrend $\mathrm{Temp}_{\mathrm{yr}-1}$ and $\mathrm{P}_{\mathrm{spring}}$, as we compare these quantities from the previous year with those of the whole simulation period, including those in the future. This does not make sense mechanistically, and therefore this allocation model is likely to be replaced in future versions of MAIDENiso.

15.4 Yearly Carbon demand

Once calculated the canopy target for this year Cmax$[y]$, the model calculates the amount of carbon that needs to be allocated to the canopy to reach the target this year, $\mathrm{CLeaf}_{\mathrm{growth}}[y]$. This is calculated from the difference between Cmax$[y]$ and the carbon content of the canopy at the start of the year CLeaf$[365(y-1)+1]$ (we count the first year as $y = 1$, not as $y = 0$):

$$\mathrm{CLeaf}_{\mathrm{growth}}[y] = \mathrm{Cmax}[y] - \mathrm{CLeaf}[365(y-1)+1] \quad . \tag{15.19}$$

The carbon content of the roots is supposed proportional to that of the canopy, therefore the amount of carbon that needs to be allocated to the roots this year is:

$$\mathrm{CRoot}_{\mathrm{growth}}[y] = \mathrm{p}_{\mathrm{root-leaf}}\ \mathrm{Cmax}[y] - \mathrm{CRoot}[365(y-1)+1] \quad , \tag{15.20}$$

where $\mathrm{p}_{\mathrm{root-leaf}}$ is the ratio of leaf carbon reservoir to root carbon reservoir, and CRoot$[365(y-1)+1]$ is the carbon content of the roots at the start of the year y.

The yearly demand of carbon is therefore:

$$\text{YearCdemand}[y] = \text{CRoot}_{\text{growth}} + \text{CLeaf}_{\text{growth}} \quad . \tag{15.21}$$

And the ratio between the carbon in the leaves and the roots r_{leaf}, which is used to partition the Spring carbon supply between leaves and roots, is calculated as:

$$\text{r}_{\text{leaf}}[y] = \frac{\text{CLeaf}_{\text{growth}}}{\text{YearCdemand}} \tag{15.22}$$

15.5 Leaf losses

Depending on the tree being deciduous or evergreen, it is defined to lose its leaves differently. For evergreen trees, the canopy is defined to lose leaves at a constant rate, during any phenological phase:

$$\text{CLeaf}_{\text{out}}[i] = \text{PercentFall} \cdot \text{AlloCCanopy} \cdot \left(\exp(-0.5 \left(\frac{\text{DOY}[i-1]}{\text{OutMax}} \right)^{\text{OutLength}}) - \exp(-0.5 \left(\frac{\text{DOY}[i]}{\text{OutMax}} \right)^{\text{OutLength}}) \right) \quad , \tag{15.23}$$

where PercentFall is the yearly canopy turnover rate, OutMax is the approximate day of the year with maximum losses, OutLength is the index proportional to the length of the period with losses, $\text{DOY}[i-1]$ is the day of the year corresponding to the simulation day i, and AlloCCanopy is defined as AlloCCanopy = Cmax in the case of Picea Mariana or as AlloCCanopy = MaxCcanopy for Mediterranean species.

For deciduous trees, $\text{Leaf}_{\text{out}i} = 0$ during Winter, Spring and Summer. During Fall:

$$\text{CLeaf}_{\text{out}}[i] = \frac{1}{1 + \text{phenolcount}} \cdot \text{MaxCcanopy} \quad , \tag{15.24}$$

where phenolcount is the length of the senescence period (in days).

15.6 Carbon allocation periods

Carbon allocation periods are distinct from phenological phases, even though the two tend to coincide. This was explained in section 14.1. In this section we explain the rules that regulate the transitions between the four distinct carbon allocation periods:

- Period 1: **Winter**. Carbon from photosynthesis (which only happens under mild conditions for evergreens) is allocated to the storage.

- Period 2: **Spring**. Carbon from photosynthesis, plus a contribution from the tree's carbon storage, is used for the growth of leafs and roots in order to reach the canopy target.
- Period 3: **Summer**. Carbon from photosynthesis is used for stem growth and replenishment of the carbon storage, split between the two.
- Period 4: **Fall**. Carbon from photosynthesis is allocated to the storage. Root mortality happens only in this period

15.6.1 Period transitions

With the exception of Spring to Summer, all the carbon allocation periods are triggered by the corresponding changes in phenological phase. Therefore, the carbon allocation periods are coincidental with the phenological phases, except for the Winter carbon allocation period which corresponds to two phenological phases (early Winter and late Winter), and the Growth or Spring-Summer phenological phase which corresponds to two carbon allocation periods (Spring and Summer).

From Winter to Spring

The transition from the Winter to the Spring carbon allocation periods is triggered by the transition from the Late Winter to the Spring-Summer phenological phases.

From Spring to Summer

The transition from the Spring to the Summer carbon allocation periods is given when one of these conditions is true:

- The amount of carbon in the canopy CLeaf has reached the canopy target (Cmax), OR
- The amount of carbon in the storage CStore has fallen below the minimum (parameter glumin), OR
- The Spring C allocation period has lasted more than the limit (50 days for boreal, 365 days for mediterranean).

From Summer to Fall

The transition from the Summer to the Fall carbon allocation periods is triggered by the transition from the Spring-Summer to the Fall phenological phases.

From Fall to Winter

The transition from the Fall to the Winter carbon allocation periods is triggered by the transition from the Fall to the Early Winter phenological phases.

15.7 Carbon allocation rules

In this section we explain the rules that are active during the carbon allocation periods.

15.7.1 Winter allocation

No growth happens during winter. For evergreens, when there is photosynthesis in winter (e.g. under mild conditions like in Mediterranean ecosystems) this readily available carbon is directed to the store reservoir in the model.

$$\mathrm{CLeaf_{in}}[i] = 0 \quad , \tag{15.25}$$

$$\mathrm{CRoot_{in}}[i] = 0 \quad , \tag{15.26}$$

$$\mathrm{CStem_{in}}[i] = 0 \quad , \tag{15.27}$$

$$\mathrm{CStore_{in}}[i] = \mathrm{NPP}[i] \quad . \tag{15.28}$$

The reservoirs do not decrease during winter, with the exception of canopy losses for evergreen trees.

$$\mathrm{CRoot_{out}}[i] = 0 \quad , \tag{15.29}$$

$$\mathrm{CStore_{out}}[i] = 0 \quad . \tag{15.30}$$

15.7.2 Spring allocation

Spring is the only period when leaves and roots increase. Carbon is pulled from the store to aid the growth of leaves and roots, and some carbon is also allocated to the stem. A daily carbon offer dayCoffer[i] is defined by the NPP of that day and the parameter StorBud (gC $\mathrm{m^{-2}d^{-1}}$), that gives the speed at which the tree can access carbon from its storage:

$$\mathrm{dayCoffer}[i] = \mathrm{NPP}[i] + \mathrm{StorBud} \quad . \tag{15.31}$$

Using this daily carbon offer, the model allocates carbon to its parts. For the leaves, the roots and the stem, the daily amount is calculated as:

$$\mathrm{CLeaf_{in}}[i] = \mathrm{BudToLeaf}[i] \cdot \mathrm{dayCoffer}[i] \cdot \mathrm{r_{leaf}} \quad , \tag{15.32}$$

$$\mathrm{CRoot_{in}}[i] = \mathrm{BudToLeaf}[i] \cdot \mathrm{dayCoffer}[i] \cdot (1 - \mathrm{r_{leaf}}) \quad , \tag{15.33}$$

$$\mathrm{CStem_{in}}[i] = (1 - \mathrm{BudToLeaf}) \cdot \mathrm{dayCoffer}[i] \quad , \tag{15.34}$$

$$\mathrm{CStore_{in}}[i] = 0 \quad . \tag{15.35}$$

In these equations, BudToLeaf$[i]$ determines the partition of the daily available carbon between the stem and the leaves-roots system:

$$\mathrm{BudToLeaf}[i] = \mathrm{StorMoistLeaf1} \cdot \exp\left(-0.5\left(\frac{\mathrm{T_{max}}[i] - \mathrm{BudLeafTmax_{ip}}}{\mathrm{BudLeafTmax_b}}\right)^2\right) \quad , \tag{15.36}$$

where $\mathrm{T_{max}}[i]$ is the maximum temperature of the day i, $\mathrm{BudLeafTmax_{ip}}$ and $\mathrm{BudLeafTmax_b}$ are parameters controlling respectively the inflection point and slope for the temperature dependence of BudToLeaf, and StorMoistLeaf1 is the portion allocated to canopy and roots when $\mathrm{T_{max}}[i] = \mathrm{BudLeafTmax_{ip}}$.

The store is depleted during this time, as it contributes to the daily offer:

$$\mathrm{CRoot_{out}}[i] = 0 \quad , \tag{15.37}$$

$$\mathrm{CStore_{out}}[i] = \mathrm{StorBud} \quad . \tag{15.38}$$

15.7.3 Summer allocation

In summer, allocation is directed only towards the stem and the store, with no growth in the canopy or the roots. The partition of carbon between the two is given by $\mathrm{npp_{stor}}$, the fraction of NPP that goes to the store:

$$\mathrm{npp_{stor}}[i] = 0.8 \exp\left(-0.5\left(\frac{\mathrm{T_{max}}[i]}{\mathrm{par_{nppstor}}}\right)^2\right) \quad , \tag{15.39}$$

where $\mathrm{par_{nppstor}}$ is the inflection point of the temperature dependence. The daily increases of the reservoirs during this period are:

$$\mathrm{CLeaf_{in}}[i] = 0 \quad , \tag{15.40}$$

$$\mathrm{CRoot_{in}}[i] = 0 \quad , \tag{15.41}$$

$$\mathrm{CStem_{in}}[i] = (1 - \mathrm{npp_{stor}[i]}) \cdot \mathrm{NPP}[i] \quad , \tag{15.42}$$

$$\mathrm{CStore_{in}}[i] = \mathrm{npp_{stor}[i]} \cdot \mathrm{NPP}[i] \quad . \tag{15.43}$$

Other than leaves (for evergreen trees), no reservoirs decrease during Summer.

$$\mathrm{CRoot_{out}}[i] = 0 \quad , \tag{15.44}$$

$$\mathrm{CStore_{out}}[i] = 0 \quad . \tag{15.45}$$

15.7.4 Fall allocation

During Fall, no growth is allowed in canopy, roots, or stem: all the positive NPP goes to the storage:

$$\mathrm{CLeaf}_{\mathrm{in}}[i] = 0 \quad , \tag{15.46}$$

$$\mathrm{CRoot}_{\mathrm{in}}[i] = 0 \quad , \tag{15.47}$$

$$\mathrm{CStem}_{\mathrm{in}}[i] = 0 \quad , \tag{15.48}$$

$$\mathrm{CStore}_{\mathrm{in}}[i] = \mathrm{NPP}[i] \quad . \tag{15.49}$$

Meanwhile, the roots decrease during this season:

$$\mathrm{CRoot}_{\mathrm{out}}[i] = \frac{\mathrm{RootTurn}}{1 + \mathrm{phenolcount}} \cdot \mathrm{CRoot} \quad , \tag{15.50}$$

where RootTurn is the fraction of root turnover and rootperleaf.

No carbon is extracted from the store in this season:

$$\mathrm{CStore}_{\mathrm{out}}[i] = 0 \quad , \tag{15.51}$$

In the case of deciduous trees, the canopy decreases during Fall. For evergreen trees, there is decrease of the canopy during the other seasons as well (see section 15.5).

Chapter 16: Photosynthesis

Photosynthesis is calculated on a daily basis, thus all quantities in this section are dependent on day i (unless they are constants). To avoid making tedious, we do not explicitly write the dependency on the time step on the right side of the equations as we do in other sections.

16.1 Photosynthesis model

The net CO_2 assimilation rate (A_n, μmol CO_2 m^{-2} s^{-1}) in MAIDENiso is calculated based on the biochemical model of Farquhar for C3 plants [Farquhar et al., 1980], where it is the difference between the potential assimilation rate A_p (μmol CO_2 m^{-2} s^{-1}) and the mitochondrial (or dark) respiration R_d (μmol CO_2 m^{-2} s^{-1}):

$$\mathrm{A_n}[i] = \mathrm{A_p} - \mathrm{R_d} \quad , \tag{16.1}$$

$$\mathrm{A_p}[i] = \min(\mathrm{Wc}, \mathrm{Wj}) \quad , \tag{16.2}$$

where W_c (μmol CO_2 m^{-2} s^{-1}) and W_j (μmol CO_2 m^{-2} s^{-1}) are the rate of photosynthesis limited by Rubisco activity and by the rate of RuBP regeneration through electron transport, respectively. W_c, W_j and R_d are estimated following:

$$\mathrm{W_c}[i] = \frac{\mathrm{Vcmax} \cdot (\mathrm{C_i} - \Gamma^*)}{\mathrm{C_i} + \mathrm{K_m}} \quad , \tag{16.3}$$

$$\mathrm{W_j}[i] = \frac{\mathrm{J} \cdot (\mathrm{C_i} - \Gamma^*)}{4\mathrm{C_i} + 8\Gamma^*} \quad , \tag{16.4}$$

$$\mathrm{R_d}[i] = f\mathrm{A_p} \quad , \tag{16.5}$$

where C_i (Pa) is the leaf-internal partial pressure of CO_2, Γ^* (Pa) is the CO_2 compensation point in the absence of dark respiration, K_m (Pa) is the effective Michaelis-Menten constant for Rubisco-limited photosynthesis at ambient partial pressure of O_2, Vcmax (μmol C m^{-2} s^{-1}) is the maximum carboxylation rate, J (μmol C m^{-2} s^{-1}) is the electron transport rate and $f = 0.0256270$ is the dark respiration coefficient.

Γ^* is calculated following Bernacchi et al. [2001]:

$$\Gamma^*[i] = 42.75 \cdot 10^{-6}\ \mathrm{P_{atm}} \exp\left(\frac{37830\ (\mathrm{Tday} - 25)}{298.15\ \mathrm{R}\ (\mathrm{Tday} + 273.15)}\right) \geq 0 \quad , \tag{16.6}$$

with $\mathrm{P_{atm}}$ (Pa) being the atmospheric pressure (Eq. (8.7)) and Tday (K) the daily air temperature (Eq. (8.5)).

$\mathrm{K_m}$ is calculated following Bernacchi et al. [2001] as:

$$\mathrm{Km}[i] = \mathrm{Kc} \cdot \left(1 + \frac{\mathrm{P_{O2}}}{\mathrm{Ko}}\right) \quad , \tag{16.7}$$

where:

$$\mathrm{P_{O2}}[i] = 0.2095 \cdot \mathrm{P_{atm}} \quad . \tag{16.8}$$

$$\mathrm{Ko}[i] = 0.2784\ \mathrm{P_{atm}} \exp\left(\frac{36380\ (\mathrm{Tday} - 25)}{298.15\ \mathrm{R}\ (\mathrm{Tday} + 273.15)}\right) \quad . \tag{16.9}$$

$$\mathrm{Kc}[i] = 404.9 \cdot 10^{-6}\ \mathrm{P_{atm}} \exp\left(\frac{79430\ (\mathrm{Tday} - 25)}{298.15\ \mathrm{R}\ (\mathrm{Tday} + 273.15)}\right) \quad . \tag{16.10}$$

Ko (Pa) and Kc (Pa) are the Michaelis-Menten constants of Rubisco for oxygenation and carboxylation, respectively, and R is the universal gas constant (J $\mathrm{mol^{-1}}$ $\mathrm{K^{-1}}$).

Vcmax is defined as:

$$\mathrm{Vcmax}[i] = \frac{\mathrm{Vmax}}{1 + \exp(\mathrm{Vb} \cdot S - \mathrm{Vip})} \cdot \theta_\mathrm{p} \quad , \tag{16.11}$$

where $\theta_\mathrm{p} = 1 - \exp(p_{str} \cdot \mathrm{SWC_{180}}) > 0$ is an empirical water stress function mainly applied to species living in Mediterranean environments ($\theta_\mathrm{p} = 1$ for other species). Vmax, Vb and Vip are calibrated parameters controlling the maximum value, the slope and the inflexion point of the dependence of Vcmax on temperature. S (K) is a temperature response function describing the acclimation of photosynthesis to Tday:

$$S[i] = \frac{\mathrm{Tday}[i] - S[i-1]}{\tau} + S[i-1] \quad , \tag{16.12}$$

where τ is a parameter interpreted as the number of days needed by the photosynthetic apparatus to acclimate to changing temperature.

The electron transport rate (J, μmol $\mathrm{m^{-2}}$ $\mathrm{s^{-1}}$) is calculated as [De Pury and Farquhar, 1997]:

$$\mathrm{J}[i] = \frac{\alpha \cdot \mathrm{a} \cdot \mathrm{I} + \mathrm{Jmax} - \sqrt{(\alpha \cdot \mathrm{a} \cdot \mathrm{I} + \mathrm{Jmax})^2 - 4\alpha \cdot \mathrm{a} \cdot \mathrm{I} \cdot \mathrm{Jmax} \cdot \theta}}{2\theta} \geq 0 \quad , \tag{16.13}$$

with Jmax (μmol m^{-2} s^{-1}) being the maximum potential rate of electron transport, I (μmol m^{-2} s^{-1}) the absorbed photosynthetic photon flux density (see section 16.3), $a = 0.85$ the leaf absorptance to absorbed photosynthetic photon flux density, θ the curvature of the light response curve (model parameter, typically $\theta = 0.677$), and $\alpha = 0.21$ the quantum efficiency of electron transport (mol electron / mol photon).

Jmax is proportional to Vcmax, modulated by a parameter $p_{Jmax} = 2.7365$:

$$\mathrm{Jmax}[i] = \mathrm{p_{Jmax}} \cdot \mathrm{Vcmax} \quad . \tag{16.14}$$

16.2 Stomatal conductance model

The stomatal conductance for carbon (μmol m^{-2} s^{-1}) is calculated using a modified version of the Leuning model [Leuning et al., 1995]:

$$\mathrm{g_{sc}}[i] = \mathrm{g_0} + \mathrm{g_1} \frac{\mathrm{A_n}}{(\mathrm{C_a} - \Gamma^*)(1 + \mathrm{VPD}/\mathrm{VPD_0}) * 10^6/\mathrm{P_{atm}}} \theta_\mathrm{g} \quad , \tag{16.15}$$

where g_0 (μmol m^{-2} s^{-1}, assumed to be 0) is the residual stomatal conductance as A_n approaches zero, and g_1 (unitless, assumed equal to 10.00235) is a parameter controlling the slope of the function, related to intercellular CO_2 concentration at saturating irradiance by $1/g_1 = 1 - C_i/C_a$. $\mathrm{C_a} = \mathrm{s_{CO2}} \cdot \mathrm{P_{atm}} \cdot 10^{-6}$ is the atmospheric CO2 pressure (Pa), s_{CO2} being the concentration of atmospheric CO_2 in ppm. VPD is the vapour pressure deficit (kPa), and VPD_0 is an empirically fitted parameter representing the sensitivity of stomata to changes in VPD (usually around 15 kPa; Knauer et al. [2015]). θ_g is the empirical soil water stress factor, ranging between 0 when the soil is too dry for the roots and 1 in absence of water stress (see subsection 11.3.1.3).

The CO_2 diffusion equation (Fick's law) links A_n with g_{sc} as:

$$\mathrm{A_n}[i] = \mathrm{g_{sc}}(\mathrm{C_a} - \mathrm{C_i}) \quad , \tag{16.16}$$

Rearranging the terms allows to estimate C_i as a function of g_{sc}:

$$\mathrm{C_i}[i] = \mathrm{C_a} - \frac{\mathrm{A_n}}{\mathrm{g_{sc}}} \quad , \tag{16.17}$$

Combining Eqs. (16.15) and (16.17) and assuming $g_0 = 0$ thus gives:

$$\mathrm{C_i}[i] = \mathrm{C_a} - \frac{\mathrm{C_a} - \Gamma^*}{\mathrm{g_1} \cdot \mathrm{rh}} \quad , \tag{16.18}$$

with:

$$\mathrm{r_h}[i] = \frac{\theta_\mathrm{g}}{(1 + \mathrm{VPD}/\mathrm{VPD_0})} \quad , \tag{16.19}$$

.

Equation 16.15 can be rewritten as:

$$g_{sc} = \frac{A_n}{(C_i - C_a) \cdot 10^6 / P_{atm}} \quad . \tag{16.20}$$

16.3 Scaling photosynthesis from leaf to canopy: a two canopy layers approach

The canopy is divided into two layers, i.e. sunlit and shaded leaves. The leaf-level photosynthesis is upscaled independently for these two layers.

The absorbed photosynthetic photon flux density I (μmol m^{-2} s^{-1}) for the sun and shaded parts is calculated following:

$$I_{sun}[i] = E_{PAR} \cdot \frac{PAR_{sun}}{LAI_{sun}} \quad , \tag{16.21}$$

$$I_{shade}[i] = E_{PAR} \cdot \frac{PAR_{shade}}{LAI_{shade}} \quad . \tag{16.22}$$

where PAR is the photosynthetically active radiation, $E_{PAR} = 4.55$ μmol J^{-1} is the PAR photon energy ratio, and LAI represents the leaf area index. Projected LAI for sunlit (LAI_{sun}) and shaded (LAI_{shade}) canopy portions are defined as:

$$LAI_{sun}[i] = 1 - \exp(-LAI) \quad . \tag{16.23}$$

$$LAI_{shade}[i] = LAI - LAI_{sun} \quad . \tag{16.24}$$

The PAR absorbed by the sunlit (PAR_{sun}) and shaded (PAR_{shade}) canopy fractions are defined as:

$$PAR_{sun}[i] = k \cdot PAR_{top} \cdot LAI_{sun} \quad , \tag{16.25}$$

$$PAR_{shade}[i] = PAR_{abs} - LAI_{sun} \quad , \tag{16.26}$$

where:

$$PAR_{top}[i] = s_{PAR} \cdot (1 - \frac{\alpha_{SW}}{3}) \quad , \tag{16.27}$$

$$PAR_{abs}[i] = PAR_{top} \cdot (1 - \exp(-k \cdot LAI)) \quad . \tag{16.28}$$

Here, k is the extinction factor (unitless parameter, we set $k = 0.9$ for black spruce), $\alpha_{SW} = 0.2$ is the site shortwave albedo (unitless) and PAR is the photosynthetically active radiation from Eq. (9.37).

The photosynthetic photon flux density I (sunlit or shaded) is then used to calculated the electron transport rate J for the sunlit and shaded portions (J_{sun} and J_{shade}) from

Eq. (16.13). This in turn gives a different rate of RuBP regeneration W_j in Eq. (16.4), and thus we obtain different net assimilation rates of A_n for the sunlit and shaded parts, i.e. $A_{n,sun}$ and $A_{n,shade}$.

The canopy-scale photosynthesis, i.e. the gross primary production GPP (gC m^{-2} d^{-1}) is calculated independently for the sunlit (GPP_{sun}, gC m^{-2} d^{-1}) and shaded (GPP_{shade}, gC m^{-2} d^{-1}) parts and added together:

$$GPP[i] = GPP_{sun} + GPP_{shade} \quad , \tag{16.29}$$

$$GPP_{sun}[i] = A_{n,sun} \cdot LAI_{sun} \cdot dayL \cdot M_C \cdot 10^{-6} \quad , \tag{16.30}$$

$$GPP_{shade}[i] = A_{n,shade} \cdot LAI_{shade} \cdot dayL \cdot M_C \cdot 10^{-6} \quad , \tag{16.31}$$

where dayL is the daytime length (s d^{-1}, dependent on the day of the year, see Eq. (9.4)), $M_C = 12.011$ g mol^{-1} is the molecular weight of carbon, and the multiplicative factor 10^{-6} comes from the conversion of μmol to mol.

The net primary production NPP (gC m^{-2} d^{-1}) is calculated as a fraction of the GPP, controlled by a constant parameter npp_{frac} (fixed at $npp_{frac} = 0.47$ for black spruce) as already expressed in Eq. 15.1:

$$NPP[i] = npp_{frac} \cdot GPP \quad . \tag{16.32}$$

16.4 Transpiration

The stomatal conductance for water (μmol m^{-2} s^{-1}) can be expressed as:

$$g_{sw}[i] = 1.6\ g_{sc} = 1.6 \frac{A_n}{(C_a - C_i) \cdot 10^6 / P_{atm}} \quad . \tag{16.33}$$

Water transpiration (mm/d) is then calculated for the sunlit and shaded parts:

$$q_{tran,sun}[i] = g_{sw} \cdot \frac{VPD}{P_{atm}} \cdot LAI_{sun} \cdot dayL \cdot M_W \quad , \tag{16.34}$$

$$q_{tran,shade}[i] = \cdot g_{sw} \cdot \frac{VPD}{P_{atm}} \cdot LAI_{shade} \cdot dayL \cdot M_W \quad , \tag{16.35}$$

where $M_W = 18.0148 \cdot 10^{-3}$ kg mol^{-1} is the molecular weight of water.

The total daily transpiration (mm/d) is calculated as:

$$q_{tran}[i] = q_{tran,sun} + q_{tran,shade} \quad . \tag{16.36}$$

Bibliography

P. Abbott and R. Tabony. The estimation of humidity parameters. *Meteorological Magazine*, 114(1351):49–56, 1985.

D. Baldocchi and T. Meyers. On using eco-physiological, micrometeorological and biogeochemical theory to evaluate carbon dioxide, water vapor and trace gas fluxes over vegetation: a perspective. *Agricultural and Forest Meteorology*, 90(1-2):1–25, 1998. doi: 10.1016/S0168-1923(97)00072-5.

M. N. Berberan-Santos, E. N. Bodunov, and L. Pogliani. On the barometric formula. *American Journal of Physics*, 65(5):404–412, 1997. doi: 10.1119/1.18555.

C. Bernacchi, E. Singsaas, C. Pimentel, A. Portis Jr, and S. P. Long. Improved temperature response functions for models of rubisco-limited photosynthesis. *Plant, Cell & Environment*, 24(2):253–259, 2001. doi: 10.1111/j.1365-3040.2001.00668.x.

P. Blanken and T. Black. The canopy conductance of a boreal aspen forest, Prince Albert National Park, Canada. *Hydrological Processes*, 18(9):1561–1578, 2004. doi: 10.1002/hyp.1406.

É. Boucher, J. Guiot, C. Hatté, V. Daux, P.-A. Danis, and P. Dussouillez. An inverse modeling approach for tree-ring-based climate reconstructions under changing atmospheric CO_2 concentrations. *Biogeosciences*, 11(12):3245–3258, 2014. doi: 10.5194/bg-11-3245-2014.

W. Buhay, T. Edwards, and R. Aravena. Evaluating kinetic fractionation factors used for ecologic and paleoclimatic reconstructions from oxygen and hydrogen isotope ratios in plant water and cellulose. *Geochimica et Cosmochimica Acta*, 60(12):2209–2218, 1996. doi: 10.1016/0016-7037(96)00073-7.

H. Craig and L. I. Gordon. *Deuterium and oxygen 18 variations in the ocean and the marine atmosphere*. Consiglio nazionale delle richerche, Laboratorio de geologia nucleare Pisa, 1965.

A. Dai. Temperature and pressure dependence of the rain-snow phase transition over land and ocean. *Geophysical Research Letters*, 35(12), 2008. doi: 10.1029/2008GL033295.

P.-A. Danis, C. Hatté, L. Misson, and J. Guiot. MAIDENiso: a multiproxy biophysical model of tree-ring width and oxygen and carbon isotopes. *Canadian journal of forest research*, 42(9):1697–1713, 2012. doi: 10.1139/x2012-089.

D. De Pury and G. Farquhar. Simple scaling of photosynthesis from leaves to canopies without the errors of big-leaf models. *Plant, Cell & Environment*, 20(5):537–557, 1997. doi: 10.1111/j.1365-3040.1997.00094.x.

M. J. DeNiro and S. Epstein. Relationship between the oxygen isotope ratios of terrestrial plant cellulose, carbon dioxide, and water. *Science*, 204(4388):51–53, 1979. doi: 10.1126/science.204.4388.51.

M. Ellehoj, H. C. Steen-Larsen, S. J. Johnsen, and M. B. Madsen. Ice-vapor equilibrium fractionation factor of hydrogen and oxygen isotopes: Experimental investigations and implications for stable water isotope studies. *Rapid Communications in Mass Spectrometry*, 27(19):2149–2158, 2013. doi: 10.1002/rcm.6668.

M. Evans. Toward forward modeling for paleoclimatic proxy signal calibration: A case study with oxygen isotopic composition of tropical woods. *Geochemistry, Geophysics, Geosystems*, 8(7), 2007. doi: 10.1029/2006GC001406.

G. Farquhar, K. Hubick, A. Condon, and R. Richards. Carbon isotope fractionation and plant water-use efficiency. In *Stable isotopes in ecological research*, pages 21–40. Springer, 1989. doi: 10.1007/978-1-4612-3498-2_2.

G. Farquhar, M. Barbour, and B. Henry. Interpretation of oxygen isotope composition of leaf material. In *Stable isotopes: Integration of Biological, Ecological and Geochemical Processes*, pages 27–62. Garland Science, 1998. ISBN 9781003076865.

G. D. Farquhar, S. v. von Caemmerer, and J. A. Berry. A biochemical model of photosynthetic co 2 assimilation in leaves of c 3 species. *Planta*, 149(1):78–90, 1980. doi: 10.1007/bf00386231.

J. R. Gat. Oxygen and hydrogen isotopes in the hydrologic cycle. *Annual Review of Earth and Planetary Sciences*, 24(1):225–262, 1996. doi: 10.1146/annurev.earth.24.1.225.

D. M. Gates. Solar radiation. In *Biophysical Ecology*, pages 96–147. Springer, 1980. ISBN 978-0486428840.

C. Gaucherel, F. Campillo, L. Misson, J. Guiot, and J.-J. Boreux. Parameterization of a process-based tree-growth model: comparison of optimization, mcmc and particle filtering algorithms. *Environmental Modelling & Software*, 23(10-11):1280–1288, 2008. doi: 10.1016/j.envsoft.2008.03.003.

G. Gea-Izquierdo, F. Guibal, R. Joffre, J. Ourcival, G. Simioni, and J. Guiot. Modelling the climatic drivers determining photosynthesis and carbon allocation in evergreen mediterranean forests using multiproxy long time series. *Biogeosciences*, 12(12):3695–3712, 2015. doi: 10.5194/bg-12-3695-2015.

F. Gennaretti, G. Gea-Izquierdo, E. Boucher, F. Berninger, D. Arseneault, and J. Guiot. Ecophysiological modeling of the climate imprint on photosynthesis and carbon allocation to the tree stem in the north american boreal forest. *Biogeosciences*, 14(21): 4851–4866, 2017. doi: 10.5194/bg-14-4851-2017.

J. Gibson, S. Birks, and T. Edwards. Global prediction of δa and δ2h-δ18o evaporation slopes for lakes and soil water accounting for seasonality. *Global Biogeochemical Cycles*, 22(2), 2008. doi: 10.1029/2007GB002997.

W. Greuell and T. Konzelmann. Numerical modelling of the energy balance and the englacial temperature of the greenland ice sheet. calculations for the eth-camp location (west greenland, 1155 m asl). *Global and Planetary change*, 9(1-2):91–114, 1994. doi: 10.1016/0921-8181(94)90010-8.

A. Harpold, M. Kaplan, P. Klos, T. Lind, J. P. McNamara, S. Rajagopal, R. Schumer, and C. Steele. Rain or snow: hydrologic processes, observations, prediction, and research needs. *Hydrology and Earth System Sciences*, 2017. doi: 10.5194/hess-21-1-2017.

B. R. Helliker and S. L. Richter. Subtropical to boreal convergence of tree-leaf temperatures. *Nature*, 454(7203):511–514, 2008. doi: 10.1038/nature07031.

I. Hermoso de Mendoza, E. Boucher, F. Gennaretti, A. Lavergne, R. Field, and L. Andreu-Hayles. A new snow module improves predictions of the isotope-enabled maideniso forest growth model. *Geoscientific Model Development*, 15(5):1931–1952, 2022.

J. Horita and D. J. Wesolowski. Liquid-vapor fractionation of oxygen and hydrogen isotopes of water from the freezing to the critical temperature. *Geochimica et Cosmochimica Acta*, 58(16):3425–3437, 1994. doi: 10.1016/0016-7037(94)90096-5.

J. V. Iribarne and W. L. Godson. *Atmospheric thermodynamics*, volume 6. Springer Science & Business Media, 1981. ISBN 978-94-009-8509-4.

C. Jaupart and J.-C. Mareschal. *Heat generation and transport in the Earth*. Cambridge university press, 2010.

M. Jones. Plant microclimate. In *Photosynthesis and production in a changing environment*, pages 47–64. Springer, 1993. doi: 10.1007/978-94-011-1566-7_4.

J. Knauer, C. Werner, and S. Zaehle. Evaluating stomatal models and their atmospheric drought response in a land surface scheme: A multibiome analysis. *Journal of Geophysical Research: Biogeosciences*, 120(10):1894–1911, 2015. doi: 10.1002/2015jg003114.

A. Lavergne, F. Gennaretti, C. Risi, V. Daux, E. Boucher, M. Savard, M. Naulier, R. Villalba, C. Begin, and J. Guiot. Modelling tree ring cellulose $\delta^1$8o variations in two temperature-sensitive tree species from north and south america. *Climate of the Past*, 13(11):1515–1526, 2017. doi: 10.5194/cp-13-1515-2017.

A. Lavergne, D. Sandoval, V. J. Hare, H. Graven, and I. C. Prentice. Impacts of soil water stress on the acclimated stomatal limitation of photosynthesis: insights from stable carbon isotope data. *Global Change Biology*, 26(12):7158–7172, 2020. doi: 10.1111/gcb.15364.

D. M. Lawrence, R. A. Fisher, C. D. Koven, K. W. Oleson, S. C. Swenson, G. Bonan, N. Collier, B. Ghimire, L. van Kampenhout, D. Kennedy, E. Kluzek, P. J. Lawrence, F. Li, H. Li, D. Lombardozzi, W. J. Riley, W. J. Sacks, M. Shi, M. Vertenstein, W. R. Wieder, C. Xu, A. A. Ali, A. M. Badger, G. Bisht, M. van den Broeke, M. A. Brunke, S. P. Burns, J. Buzan, M. Clark, A. Craig, K. Dahlin, B. Drewniak, J. B. Fisher, M. Flanner, A. M. Fox, P. Gentine, F. Hoffman, G. Keppel-Aleks, R. Knox, S. Kumar, J. Lenaerts, L. R. Leung, W. H. Lipscomb, Y. Lu, A. Pandey, J. Pelletier, Jon D. andn Perket, J. T. Randerson, D. M. Ricciuto, B. M. Sanderson, A. Slater, Z. M. Subin, J. Tang, R. Q. Thomas, M. Val Martin, and X. Zeng. The community land model version 5: Description of new features, benchmarking, and impact of forcing uncertainty. *Journal of Advances in Modeling Earth Systems*, 11(12):4245–4287, 2019. doi: 10.1029/2018MS001583.

G. Leavesley, P. J. Restrepo, S. Markstrom, M. Dixon, and L. Stannard. The modular modeling system (mms): User's manual. *US Geological Survey Open-File Report*, 96 (151,142), 1996. URL `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.6662&rep=rep1&type=pdf`.

M. Lehmann and U. Siegenthaler. Equilibrium oxygen-and hydrogen-isotope fractionation between ice and water. *Journal of Glaciology*, 37(125):23–26, 1991. doi: 10.3189/S0022143000042751.

R. Leuning, F. M. Kelliher, D. De Pury, and E.-D. Schulze. Leaf nitrogen, photosynthesis, conductance and transpiration: scaling from leaves to canopies. *Plant, Cell & Environment*, 18(10):1183–1200, 1995. doi: 10.1111/j.1365-3040.1995.tb00628.x.

E. Linacre. A note on a feature of leaf and air temperatures. *Agricultural Meteorology*, 1(1):66–72, 1964. doi: 10.1016/0002-1571(64)90009-3.

J. Mahfouf and J. Noilhan. Comparative study of various formulations of evaporations from bare soil using in situ data. *Journal of Applied Meteorology*, 30(9):1354–1365, 1991. doi: 10.1175/1520-0450(1991)030⟨1354:CSOVFO⟩2.0.CO;2.

G. J. McCabe and D. M. Wolock. Recent declines in western us snowpack in the context of twentieth-century climate variability. *Earth Interactions*, 13(12):1–15, 2009. doi: 10.1175/2009EI283.1.

L. Misson. MAIDEN: a model for analyzing ecosystem processes in dendroecology. *Canadian Journal of Forest Research*, 34(4):874–887, 2004. doi: 10.1139/x03-252.

J. L. Monteith. Evaporation and environment. In *Symposia of the society for experimental biology*, volume 19, pages 205–234. Cambridge University Press (CUP) Cambridge, 1965.

J. R. O'Neil. Hydrogen and oxygen isotope fractionation between ice and water. *The Journal of Physical Chemistry*, 72(10):3683–3684, 1968.

A. Passerat de Silans. *Transferts de masse et de chaleur dans un sol stratifié soumis à une excitation atmosphérique naturelle: comparaison: modèles-expérience*. PhD thesis, Grenoble INPG, 1986.

J. Pomeroy, D. Gray, K. Shook, B. Toth, R. Essery, A. Pietroniro, and N. Hedstrom. An evaluation of snow accumulation and ablation processes for land surface modelling. *Hydrological Processes*, 12(15):2339–2367, 1998. doi: 10.1002/(SICI)1099-1085(199812)12:15⟨2339::AID-HYP800⟩3.0.CO;2-L.

J. S. Roden and J. R. Ehleringer. Hydrogen and oxygen isotope ratios of tree ring cellulose for field-grown riparian trees. *Oecologia*, 123(4):481–489, 2000. doi: 10.1007/s004420000349.

J. Ross. Radiative transfer in plant communities. *Vegetation and the Atmosphere*, pages 13–55, 1975.

M. Saurer, K. Aellen, and R. Siegwolf. Correlating δ13c and δ18o in cellulose of trees. *Plant, Cell & Environment*, 20(12):1543–1550, 1997. doi: 10.1046/j.1365-3040.1997.d01-53.x.

L. D. S. Sternberg, M. J. Deniro, and R. A. Savidge. Oxygen isotope exchange between metabolites and water during biochemical reactions leading to cellulose synthesis. *Plant Physiology*, 82(2):423–427, 1986. doi: 10.1104/pp.82.2.423.

E. E. Stigter, M. Litt, J. F. Steiner, P. N. Bonekamp, J. M. Shea, M. F. Bierkens, and W. W. Immerzeel. The importance of snow sublimation on a himalayan glacier. *Frontiers in Earth Science, Cryosphere*, 6, 2018. doi: 10.3389/feart.2018.00108.

P. Szejner, W. E. Wright, S. Belmecheri, D. Meko, S. W. Leavitt, J. R. Ehleringer, and R. K. Monson. Disentangling seasonal and interannual legacies from inferred patterns of forest water and carbon cycling using tree-ring stable isotopes. *Global change biology*, 24(11):5332–5347, 2018. doi: 10.1111/gcb.14395.

L. van Kampenhout, J. T. Lenaerts, W. H. Lipscomb, W. J. Sacks, D. M. Lawrence, A. G. Slater, and M. R. van den Broeke. Improving the representation of polar snow and firn in the community earth system model. *Journal of Advances in Modeling Earth Systems*, 9(7):2583–2600, 2017. doi: 10.1002/2017MS000988.

D. Yakir. Variations in the natural abundance of oxygen-18 and deuterium in plant carbohydrates. *Plant, Cell & Environment*, 15(9):1005–1020, 1992. doi: 10.1111/j.1365-3040.1992.tb01652.x.

Appendix A: MAIDENiso outputs

Output	C++ variable	File	Type	Unit	Definition
period	alloc->period	outalloc_d	daily	NA	Phenological period
Acanopy	photo->A_canopy	outalloc_d	daily	gC / m^2 stand / d	Gross Primary Production (GPP)
AutoR	alloc->auto_resp	outalloc_d	daily	gC / m^2 stand / d	Autothrophic respiration
Anet	alloc->A_net	outalloc_d	daily	gC / m^2 stand / d	Net production
Cleaf	alloc->C_leaf	outalloc_d	daily	gC / m^2 stand	Leaf carbon store
Croot	alloc->C_root	outalloc_d	daily	gC / m^2 stand	Root carbon store
Cstem	alloc->C_stem	outalloc_d	daily	gC / m^2 stand	Stem carbon store
Cstor	alloc->C_stor	outalloc_d	daily	gC / m^2 stand	Reserve carbon store
in_leaf	alloc->inC_leaf	outalloc_d	daily	gC / m^2 stand	Leaf carbon daily input
in_root	alloc->inC_root	outalloc_d	daily	gC / m^2 stand	Root carbon daily input
in_stem	alloc->inC_stem	outalloc_d	daily	gC / m^2 stand	Stem carbon daily input
in_stor	alloc->inC_stor	outalloc_d	daily	gC / m^2 stand	Reserve carbon daily input
out_leaf	alloc->outC_leaf	outalloc_d	daily	gC / m^2 stand	Leaf carbon daily output
out_root	alloc->outC_root	outalloc_d	daily	gC / m^2 stand	Root carbon daily output
out_stem	alloc->outC_stem	outalloc_d	daily	gC / m^2 stand	Stem carbon daily output
out_stor	alloc->outC_stor	outalloc_d	daily	gC / m^2 stand	Reserve carbon daily output
d_leaf	alloc->dC_leaf	outalloc_d	daily	gC / m^2 stand	Leaf carbon daily change
d_root	alloc->dC_root	outalloc_d	daily	gC / m^2 stand	Root carbon daily change
d_stem	alloc->dC_stem	outalloc_d	daily	gC / m^2 stand	Stem carbon daily change
d_stor	alloc->dC_stor	outalloc_d	daily	gC / m^2 stand	Reserve carbon daily change
lai	alloc->LAI	outalloc_d	daily	NA	Leaf Area Index (LAI)
glumax	alloc->glumax	outalloc_d	daily	gC / m^2 stand	Maximum storage capacity
day12	phenol->day12	outalloc_y	yearly	DOY	Start of GDD accumulation period
day23	phenol->day23	outalloc_y	yearly	DOY	Raw Budburst / Start day of growth period
day23_sp	phenol->day23_sp	outalloc_y	yearly	DOY	Corrected Budburst
day3spr3sum	alloc->day3spr3sum	outalloc_y	yearly	DOY	Start of Summer
day34	phenol->day34	outalloc_y	yearly	DOY	Start of Autumn
day41	phenol->day41	outalloc_y	yearly	DOY	Start of Winter
GPP	alloc->GPP	outalloc_y	yearly	gC / m^2 stand / y	Yearly Gross Primary Production (GPP)
NPP	alloc->NPP	outalloc_y	yearly	gC / m^2 stand / y	Yearly Net Primary Production (NPP)
auto_resp	alloc->auto_resp_y	outalloc_y	yearly	gC / m^2 stand / y	Yearly Autotrophic Respiration
LAIyear	alloc->LAI_year	outalloc_y	yearly	gC / m^2 stand	Maximum Leaf Area Index during the year
Dstem	alloc->stemi	outalloc_y	yearly	gC / m^2 stand	Stem carbon yearly change
leaf	alloc->C_leaf_year	outalloc_y	yearly	gC / m^2 stand	Maximum leaf carbon during the year
root	alloc->C_root_year	outalloc_y	yearly	gC / m^2 stand	Maximum root carbon during the year
stem	alloc->C_stem_year	outalloc_y	yearly	gC / m^2 stand	Maximum stem carbon during the year
stor	alloc->C_stor_year	outalloc_y	yearly	gC / m^2 stand	Maximum reserve carbon during the year
in_leaf	alloc->inC_leaf_y	outalloc_y	yearly	gC / m^2 stand	Leaf carbon yearly input

Output	C++ variable	File	Type	Unit	Definition
out_leaf	alloc->outC_leaf_y	outalloc_y	yearly	gC / m^2 stand	Leaf carbon yearly output
in_root	alloc->inC_root_y	outalloc_y	yearly	gC / m^2 stand	Root carbon yearly input
out_root	alloc->outC_root_y	outalloc_y	yearly	gC / m^2 stand	Root carbon yearly output
in_stem	alloc->inC_stem_y	outalloc_y	yearly	gC / m^2 stand	Stem carbon yearly input
out_stem	alloc->outC_stem_y	outalloc_y	yearly	gC / m^2 stand	Stem carbon yearly output
in_stor	alloc->inC_stor_y	outalloc_y	yearly	gC / m^2 stand	Reserve carbon yearly input
out_stor	alloc->outC_stor_y	outalloc_y	yearly	gC / m^2 stand	Reserve carbon yearly output
stor_leaf	alloc->stor_leaf	outalloc_y	yearly	gC / m^2 stand	Reserve storage in the leaves
stor_root	alloc->stor_root	outalloc_y	yearly	gC / m^2 stand	Reserve storage in the roots
stor_stem	alloc->stor_stem	outalloc_y	yearly	gC / m^2 stand	Reserve storage in the stem
npp_stor	alloc->npp_stor	outalloc_y	yearly	gC / m^2 stand	Reserve carbon input during Summer and Autumn
c_balance	alloc->c_balance	outalloc_y	yearly	gC / m^2 stand	Carbon balance: inputs - outputs (should be 0)
glumin	alloc->glumin	outalloc_y	yearly	gC / m^2 stand	Maximum storage capacity
Cmax2	alloc->Cmax2	outalloc_y	yearly	gC / m^2 stand	Canopy carbon target
T_previous	meteo->T_previous_Y	outalloc_y	yearly	C	Average Summer temperature of previous year
Pspring	meteo->Pspring	outalloc_y	yearly	cm	Spring precipitation of previous year
d2Hp	d2H->precip	outd2H	daily	permil	Precipitation d2H
d2Hxylem	d2H->xylem	outd2H	daily	permil	Xylem water d2H
d2Hsnow	d2H->snow	outd2H	daily	permil	Snow ice d2H
d2Hsnowwat	d2H->snowwat	outd2H	daily	permil	Snow water d2H
d2Hsw1	d2H->soil[0]	outd2H	daily	permil	Soil water d2H in layer 1
d2Hsw2	d2H->soil[1]	outd2H	daily	permil	Soil water d2H in layer 2
d2Hsw3	d2H->soil[2]	outd2H	daily	permil	Soil water d2H in layer 3
d2Hsw4	d2H->soil[3]	outd2H	daily	permil	Soil water d2H in layer 4
d2HTRC	d2H->TRC	outd2H	daily	permil	Tree Ring Cellulose d2H
d2Hcanopy	d2H->cws	outd2H	daily	permil	Canopy water d2H
d2Hevap	d2H->Epot_through	outd2H	daily	permil	Evaporated water d2H
d2Hthrough	d2H->through	outd2H	daily	permil	Throughfall d2H
d2Hvapor	d2H->vapor	outd2H	daily	permil	Atmospheric water vapor d2H
d2Hsoilevap	d2H->soilevap	outd2H	daily	permil	Soil evaporation d2H
d2Hsnowevap	d2H->snowevap	outd2H	daily	permil	Snow evaporation d2H
Dd13C_TRC	d13C->DTRC	outd13C_d	daily	permil	Discrimination against 13C
newd13Cstem	d13C->TRCnew	outd13C_d	daily	permil	Tree Ring Cellulose d13C
d13C_stor	d13C->stor	outd13C_d	daily	permil	Stored carbon d13C
d13C_CO2	d13C->d13CO2	outd13C_d	daily	permil	Atmospheric d13C
alloc_stem	alloc->inC_stem	outd13C_d	daily	gC / m^2 stand	Stem carbon daily input

Output	C++ variable	File	Type	Unit	Definition
CCi_CO2-ppmV	d13C->CCi_CO2	outd13C_d	daily	permil	Leaf-internal partial pressure of CO_2
CCa_CO2-ppmV	meteo->s_CO2	outd13C_d	daily	ppm	Atmospheric CO_2 concentration
D13C	d13C->DTRC_y	outd13C_y	yearly	permil	Year-average discrimination against 13C, weighted with stem C
d13Cstem	d13C->TRCnew_y	outd13C_y	yearly	permil	Year-average Tree Ring Cellulose d13C, weighted with stem C
d13CGPP	var_mean	outd13C_y	yearly	permil	Year-average Tree Ring Cellulose d13C, weighted with GPP
d13CO2	d13C->d13CO2_y	outd13C_y	yearly	permil	Year-average atmospheric d13C, unweighted
CCi_CO2-ppmV	d13C->CCi_CO2_y	outd13C_y	yearly	permil	Year-average CCi, weighted with stem C
CCa_CO2-ppmV	d13C->CCa_CO2_y	outd13C_y	yearly	permil	Year-average CCa, weighted with stem C
C_ratio	d13C->Cratio_y	outd13C_y	yearly	permil	Cci / Cca ratio
d18Orainfall	d18O->precip	outd18O_d	daily	permil	Rainfall d18O
d18Osnowfall	d18O->snowfall	outd18O_d	daily	permil	Snowfall d18O
d18Oxylem	d18O->xylem	outd18O_d	daily	permil	Xylem water d18O
d18Osnow	d18O->snow	outd18O_d	daily	permil	Snow ice d18O
d18Osnowwat	d18O->snowwat	outd18O_d	daily	permil	Snow water d18O
d18Osw1	d18O->soil[0]	outd18O_d	daily	permil	Soil water d18O in layer 1
d18Osw2	d18O->soil[1]	outd18O_d	daily	permil	Soil water d18O in layer 2
d18Osw3	d18O->soil[2]	outd18O_d	daily	permil	Soil water d18O in layer 3
d18Osw4	d18O->soil[3]	outd18O_d	daily	permil	Soil water d18O in layer 4
d18Odrai	d18O->drai	outd18O_d	daily	permil	Drainage water d18O
d18Orun	d18O->srun	outd18O_d	daily	permil	Runoff water d18O
d18Oinfl	d18O->infl	outd18O_d	daily	permil	Infiltrated water d18O
d18OTRC	d18O->TRC	outd18O_d	daily	permil	Tree Ring Cellulose d18O
d18Ocanwat	d18O->cws	outd18O_d	daily	permil	Canopy water d18O
d18Ocansnow	d18O->css	outd18O_d	daily	permil	Canopy snow d18O
d18Oevap	d18O->Epot_through	outd18O_d	daily	permil	Evaporated water d18O
d18Othrough	d18O->through	outd18O_d	daily	permil	Throughfall d18O
d18Ovapor	d18O->vapor	outd18O_d	daily	permil	Atmospheric water vapor d18O
d18Osoilevap	d18O->soilevap	outd18O_d	daily	permil	Soil evaporation d18O
d18Osnowevap	d18O->snowevap	outd18O_d	daily	permil	Snow evaporation d18O
d18Ostem	d18O->TRC_y	outd18O_y	yearly	permil	Year-average Tree Ring Cellulose d18O, weighted with stem C
d18OGPP	var_mean	outd18O_y	yearly	permil	Year-average Tree Ring Cellulose d18O, weighted with GPP
d18Oxylem	d18O->xylem_y	outd18O_y	yearly	permil	Year-average xylem water d18O
d18Ovapor	d18O->vapor_y	outd18O_y	yearly	permil	Year-average atmospheric water vapor d18O
Tmax	meteo->s_tmax	outmet	daily	C	Maximum atmospheric temperature
Tmin	meteo->s_tmin	outmet	daily	C	Minimum atmospheric temperature
Tavg	meteo->s_tavg	outmet	daily	C	Average atmospheric temperature

Output	C++ variable	File	Type	Unit	Definition
Tday	meteo->s_tday	outmet	daily	C	Daylight temperature
TdayTr	meteo->s_tday_transf	outmet	daily	C	Daylight transformed temperature
rain	meteo->s_prcp	outmet	daily	mm / d	Rainfall
snow	meteo->s_prcp_snow	outmet	daily	mm / d	Snowfall
VPD	meteo->s_hum	outmet	daily	Pa	Vapor Pressure Deficit
srad	meteo->s_srad	outmet	daily	W / m^2	Solar radiation
daylen	meteo->s_dayl	outmet	daily	s	Daylight length
PAR	meteo->s_par	outmet	daily	W / m^2	Photosynthetically Active Radiation
gbh	meteo->s_gbh	outmet	daily	mol CO_2 / m^2 / s	Boundary layer conductance (from Montheit)
gaer	meteo->s_gaer	outmet	daily	mol H_2O / m^2 / s	Canopy aerodynamic conductance (from Montheit)
svp	meteo->s_svp	outmet	daily	Pa	Saturated Vapor Pressure
avp	meteo->s_avp	outmet	daily	Pa	Atmospheric Vapor Pressure
avp_mtclim	meteo->avp_mtclim	outmet	daily	Pa	Transformed Atmospheric Vapor Pressure? Serves no purpose
srad_mtclim	meteo->srad_mtclim	outmet	daily	W / m^2	Transformed solar radiation? Serves no purpose
soilmm	water->soil_mm	outmet	daily	mm	Soil water content (sum of all layers)
wstress	water->stress	outmet	daily	NA	Stress from soil moisture
vpdstress	photo->vpd_stress	outmet	daily	NA	Stress from vapor pressure deficit
atms_p	meteo->s_pa	outmet	daily	Pa	Atmospheric pressure
PARsun	rad->ppfd_per_plaisun	outmet	daily	umol / m^2 / s	Photosynthetic Photon Flux Density in the sunlight portion of the canopy
PARshade	rad->ppfd_per_plaishade	outmet	daily	umol / m^2 / s	Photosynthetic Photon Flux Density in the shaded portion of the canopy
CO2	meteo->s_CO2	outmet	daily	ppm	Atmospheric CO_2 concentration
RH?	meteo->rh	outmet	daily	NA	Atmospheric Relative Humidity
RHavg	meteo->rhavg	outmet	daily	NA	Average Relative Humidity
veg_phase	phenol->veg_phase	outphenol	daily	NA	vegetation phase
sumdd	phenol->sumdd	outphenol	daily	C	Accumulated sum of degree days
thawed	water->thawed	outphenol	daily	NA	flag indicating if the soil is above the thawed threshold
laisun	rad->plaisun	outphoto	daily	NA	Leaf Area Index (Sunlight portion)
laishade	rad->plaishade	outphoto	daily	NA	Leaf Area Index (Shaded portion)
Anet	alloc->A_net	outphoto	daily	gC / m^2 stand / d	Net production
Acanopy	alloc->A_canopy	outphoto	daily	gC / m^2 stand / d	Gross Primary Production (GPP)
Asuncanopy	alloc->A_sun_canopy	outphoto	daily	gC / m^2 stand / d	GPP from the sunlight canopy portion
Ashadecanopy	alloc->A_shade_canopy	outphoto	daily	gC / m^2 stand / d	GPP from the shaded canopy portion
Asun	photo->A_sun	outphoto	daily	umol / m^2	GPP from the sunlight canopy portion
Ashade	photo->A_shade	outphoto	daily	umol / m^2	GPP from the shaded canopy portion
Ajsun	photo->Aj_sun	outphoto	daily	mmol / m^2 / s	CO_2 assimilation rate limited by RuBP regeneration (light limited/RuBP limited) in the sunlight canopy portion

Output	C++ variable	File	Type	Unit	Definition
Ajshade	photo->Aj_shade	outphoto	daily	mmol / m^2 / s	CO_2 assimilation rate limited by RuBP regeneration (light limited/RuBP limited) in the shaded canopy portion
Avsun	photo->Av_sun	outphoto	daily	mmol / m^2 / s	CO_2 assimilation rate limited by the amount and activity of Rubisco (enzyme limited/RuBP saturated) in the sunlight canopy portion
Avshade	photo->Av_shade	outphoto	daily	mmol / m^2 / s	assimilation rate limited by the amount and activity of Rubisco (enzyme limited/RuBP saturated) in the shaded canopy portion
Rday	photo->Rday	outphoto	daily	umol / m^2 / s	Mitochondrial/dark respiration rate
gssun	photo->gs_c_sun	outphoto	daily	umol / m^2 / s	Stomatal conductance from the sunlight canopy portion
gsshade	photo->gs_c_shade	outphoto	daily	umol / m^2 / s	Stomatal conductance from the shaded canopy portion
cssun	photo->Cs_sun	outphoto	daily	Pa	partial pressure of CO_2 in the sunlight canopy portion
csshade	photo->Cs_shade	outphoto	daily	Pa	partial pressure of CO_2 in the shaded canopy portion
cisun	photo->Ci_sun	outphoto	daily	Pa	Leaf-internal partial pressure of CO_2 in the sunlight canopy portion
cishade	photo->Ci_shade	outphoto	daily	Pa	Leaf-internal partial pressure of CO_2 in the shaded canopy portion
Rdsun	photo->Rd_sun	outphoto	daily	umol / m^2 / s	Mitochondrial/dark respiration rate in the sunlight canopy portion
Rdshade	photo->Rd_shade	outphoto	daily	umol / m^2 / s	Mitochondrial/dark respiration rate in the shaded canopy portion
vmax	photo->Vmax	outphoto	daily	umol / m^2 / s	Maximum carboxylation rate
km	photo->Km	outphoto	daily	Pa	Effective Michaelis-Menten constant for carboxylation reactions
gamma	photo->gamma	outphoto	daily	Pa	Compensation point in the absence of dark respiration
kc	photo->Kc	outphoto	daily	Pa	Michaelis-Menten constant of Rubisco for carboxylation
ko	photo->Ko	outphoto	daily	Pa	Michaelis-Menten constant of Rubisco for oxygenation
o2	photo->O2	outphoto	daily	Pa	Partial pressure of O_2
jsun	photo->J_sun	outphoto	daily	umol / m^2 / s	Electron transport rate in the sunlight canopy portion
jshade	photo->J_shade	outphoto	daily	umol / m^2 / s	Electron transport rate in the shaded canopy portion
photostress	photo->rh	outphoto	daily	NA	Stress for the photosynthesis (accounts for wstress and vpdstress)
wstress	water->stress	outphoto	daily	NA	Stress from soil moisture
vpdstress	photo->vpd_stress	outphoto	daily	NA	Stress from vapor pressure deficit
fpar	rad->f_par_srad	outrad	daily	NA	Fraction of radiation that is PAR?
lai	alloc->LAI	outrad	daily	NA	Leaf Area Index
laisun	rad->plaisun	outrad	daily	NA	Leaf Area Index (Sunlight portion)
laishade	rad->plaishade	outrad	daily	NA	Leaf Area Index (Shaded portion)
par	meteo->s_par	outrad	daily	W / m^2	Photosynthetically Active Radiation
partop	rad->par_top	outrad	daily	W / m^2	PAR not reflected by albedo
parabs	rad->parabs	outrad	daily	W / m^2	PAR absorbed
parabs_sun	rad->parabs_plaisun	outrad	daily	W / m^2	PAR absorbed by the sunlight canopy portion
parabs_shade	rad->parabs_plaishade	outrad	daily	W / m^2	PAR absorbed by the shaded canopy portion
parabs_sun	rad->parabs_per_plaisun	outrad	daily	W / m^2	PAR absorbed per LAI unit by the sunlight canopy portion

Output	C++ variable	File	Type	Unit	Definition
parabs_shade	rad->parabs_per_plaishade	outrad	daily	W / m^2	PAR absorbed per LAI unit by the shaded canopy portion
ppfd_sun	rad->ppfd_per_plaisun	outrad	daily	umol / m^2 / s	Photosynthetic Photon Flux Density in the sunlight portion of the canopy
ppfd_shade	rad->ppfd_per_plaishade	outrad	daily	umol / m^2 / s	Photosynthetic Photon Flux Density in the shaded portion of the canopy
Root1	soil->root[0]	outroot	daily	NA	Root fraction in soil layer 1
Root2	soil->root[1]	outroot	daily	NA	Root fraction in soil layer 2
Root3	soil->root[2]	outroot	daily	NA	Root fraction in soil layer 3
Root4	soil->root[3]	outroot	daily	NA	Root fraction in soil layer 4
simul	sum_d(alloc->inC_stem)	outsim	yearly	NA	Standarized (mean=1) stem growth
dzsnow(m)	water->dzsnow	outsnow	daily	m	Thickness of the snow layer
h2osnow(kg)	water->h2osnow	outsnow	daily	kg / m^2	Liquid water content of the snow layer
icesnow(kg)	water->icesnow	outsnow	daily	kg / m^2	Solid water content of the snow layer
density(kg/m)	water->icesnow/water->dzsnow	outsnow	daily	kg / m^3	Density of the ice in the snow layer
si[1](%)	water->si[0]	outsnow	daily	NA	Volumetric fraction of ice content in soil layer 1
si[2](%)	water->si[1]	outsnow	daily	NA	Volumetric fraction of ice content in soil layer 2
si[3](%)	water->si[2]	outsnow	daily	NA	Volumetric fraction of ice content in soil layer 3
si[4](%)	water->si[3]	outsnow	daily	NA	Volumetric fraction of ice content in soil layer 4
sm1	water->sm[0]	outsoil	daily	NA	Volumetric fraction of water content in soil layer 1
sm2	water->sm[1]	outsoil	daily	NA	Volumetric fraction of water content in soil layer 2
sm3	water->sm[2]	outsoil	daily	NA	Volumetric fraction of water content in soil layer 3
sm4	water->sm[3]	outsoil	daily	NA	Volumetric fraction of water content in soil layer 4
swp1	water->swp[0]	outsoil	daily	NA	Soil water potential (x -1) of layer 1
swp2	water->swp[1]	outsoil	daily	NA	Soil water potential (x -1) of layer 2
swp3	water->swp[2]	outsoil	daily	NA	Soil water potential (x -1) of layer 3
swp4	water->swp[3]	outsoil	daily	NA	Soil water potential (x -1) of layer 4
soilmm	water->soil_mm	outsoil	daily	mm	Soil water content (sum of all layers)
soilice	water->ice_mm	outsoil	daily	mm	Soil ice content (sum of all layers)
Se	water->se	outsoilev	daily	mm / d	Soil water evaporation
Ss	water->ss	outsoilev	daily	mm / d	Soil ice sublimation
time	meteo->s_dayl	outsoilev	daily	s	Daylight length
Epot	water->Epot_se	outsoilev	daily	kg / m^2 / s	Potential evaporation
aird	soilev->aird	outsoilev	daily	kg / m^3	Air density
satmrl	soilev->satmrl	outsoilev	daily	Pa / Pa	Saturated mixing ratio
soilrhl	soilev->soilrhl	outsoilev	daily	NA	Soil Relative Humidity
arh	meteo->rh	outsoilev	daily	NA	Atmospheric Relative Humidity
rsoil	soilev->rsoil	outsoilev	daily	s / m	Soil resistance to evaporation
ra	soilev->ra	outsoilev	daily	NA	Unknown

Output	C++ variable	File	Type	Unit	Definition
Tatm	meteo->s_tavg	outtemp	daily	C	Average atmospheric temperature
Tsnow	soil->temp[0]	outtemp	daily	C	Temperature of the snow layer
T1	soil->temp[1]	outtemp	daily	C	Temperature of soil layer 1
T2	soil->temp[2]	outtemp	daily	C	Temperature of soil layer 2
T3	soil->temp[3]	outtemp	daily	C	Temperature of soil layer 3
T4	soil->temp[4]	outtemp	daily	C	Temperature of soil layer 4
ct	water_an->ct	outwatbalance	yearly	mm	Transpired canopy water
ddr	water_an->ddr	outwatbalance	yearly	mm	Drained (percolated) soil water
se	water_an->se	outwatbalance	yearly	mm	Evaporated water
ss	water_an->ss	outwatbalance	yearly	mm	Sublimated ice/snow
through	water_an->through	outwatbalance	yearly	mm	Throughfall
prcp	water_an->prcp	outwatbalance	yearly	mm	Precipitation
srun	water_an->srun	outwatbalance	yearly	mm	Runoff
intercept	water_an->intercept	outwatbalance	yearly	mm	Intercepted precipitation
inf	water_an->inf	outwatbalance	yearly	mm	Infiltration
delta.sm	water_an->delta_sm	outwatbalance	yearly	mm	Increase in soil moisture
init.sm	water_an->sm_initial	outwatbalance	yearly	mm	Initial soil moisture
fin.sm	water_an->sm_final	outwatbalance	yearly	mm	Final soil moisture
w_balance	water_an->w_balance	outwatbalance	yearly	mm	Water balance (should be 0)
prcp	meteo->s_prcp+meteo->s_prcp_snow	outwater	daily	mm	Precipitation
pdirect	water->pdirect+water->psnowdirect	outwater	daily	mm	Direct precipitation
through	water->through+water->through_snow	outwater	daily	mm	Throughfall
intercept	water->intercept	outwater	daily	mm	Intercepted precipitation
canopy_snow	water->canopy_snow	outwater	daily	mm	Precipitation
se	water->se	outwater	daily	mm	Evaporated water
ss	water->ss	outwater	daily	mm	Sublimated ice/snow
inf	water->inf	outwater	daily	mm	Infiltration
srun	water->srun	outwater	daily	mm	Runoff
ct	water->ct	outwater	daily	mm	Transpired canopy water
ddr	water->ddr	outwater	daily	mm	Drained (percolated) soil water
soil_water	water->soil_mm	outwater	daily	mm	Soil water content (sum of all layers)
Epot_through	water->Epot_through	outwater	daily	mm	Evaporated canopy water
Epot_through_snow	water->Epot_through_snow	outwater	daily	mm	Sublimated canopy snow
Epot_se	water->Epot_se	outwater	daily	kg / m^2 / s	Potential evaporation
lai	alloc->LAI	outwater	daily	NA	Leaf Area Index (LAI)
wstress	water->stress	outwater	daily	NA	Stress from soil moisture

Output	C++ variable	File	Type	Unit	Definition
vpdstress	photo->vpd_stress	outwater	daily	NA	Stress from vapor pressure deficit
photostress	photo->rh	outwater	daily	NA	Stress for the photosynthesis (accounts for wstress and vpdstress)
soil_rh	water->soil_rh	outwater	daily	NA	Soil Relative Humidity
r_soil	water->r_soil	outwater	daily	s / m	Soil resistance to evaporation
sat_mr	water->sat_mr	outwater	daily	Pa / Pa	Saturated mixing ratio
airrh	meteo->rh	outwater	daily	NA	Atmospheric Relative Humidity
s_swe	meteo->s_swe	outwater	daily	cm	Snowpack thickness (preliminary calculation, different from the actual snow layer)