
Tree Box Model

Release 0.3

Olli-Pekka Tikkanen

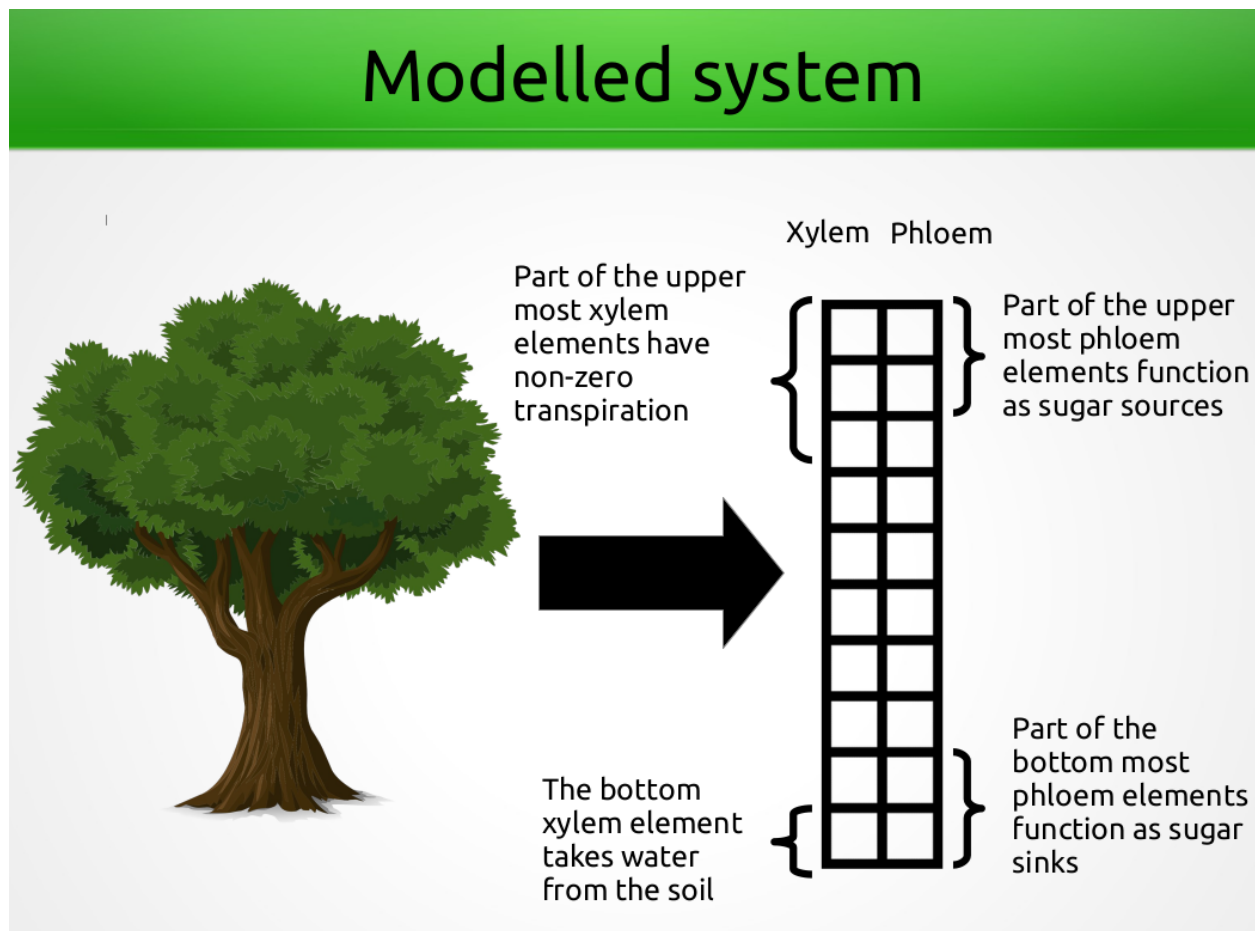
Jun 09, 2020

DETAILS OF THE MODEL

1	Description of the modelled system	1
1.1	Introduction	1
1.2	Input and output of the model	2
1.3	The fluxes inside the tree	3
1.4	The change of pressure, sugar concentration and element radius due to sap flux	4
1.5	Sugar loading and unloading rates	5
1.6	How the element radii are modelled	5
2	Instructions to run the model	7
2.1	Running from main.py	7
2.2	Importing src.model	7
3	Installation	11
3.1	Download the source	11
3.2	Install the required packages	11
4	Quick start	13
5	main.py	15
6	Modules, Classes & functions	17
	Python Module Index	27
	Index	29

DESCRIPTION OF THE MODELLED SYSTEM

1.1 Introduction



Modelled system is a 2-dimensional array representation of a tree. Each property of the tree is saved in a NumPy array. The first column in each array represents values of the xylem and the second column values of the phloem. The rows of the arrays represent vertical elements in the tree. The height of an element (l_i) is calculated from the input total tree height (H) and number of elements (N)

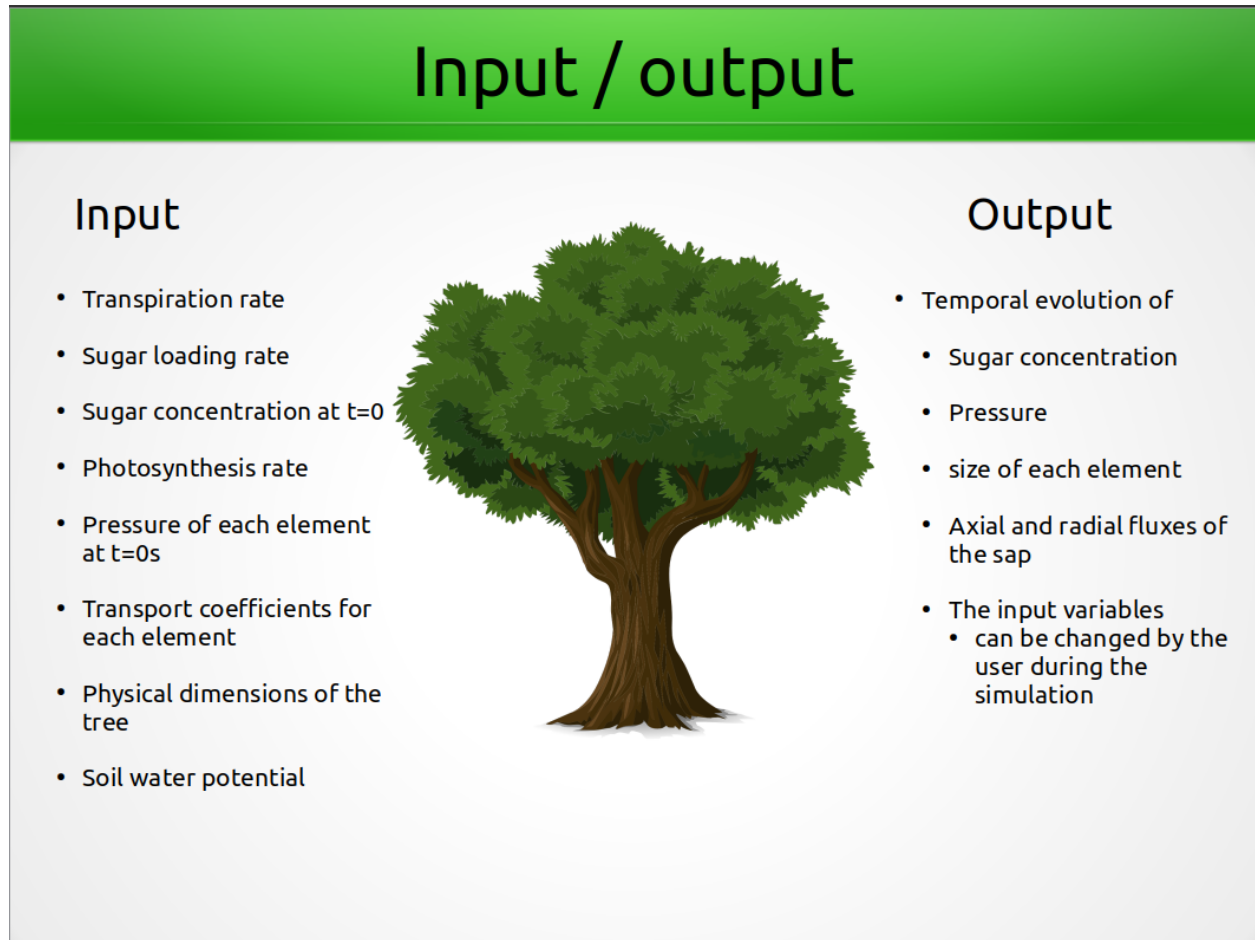
$$l_i = \frac{H}{N}$$

The height of an element does not change during the simulation.

The row numbering of the arrays starts from the top of the tree ($i=0$) and ends in the bottom of the tree ($i=N$).

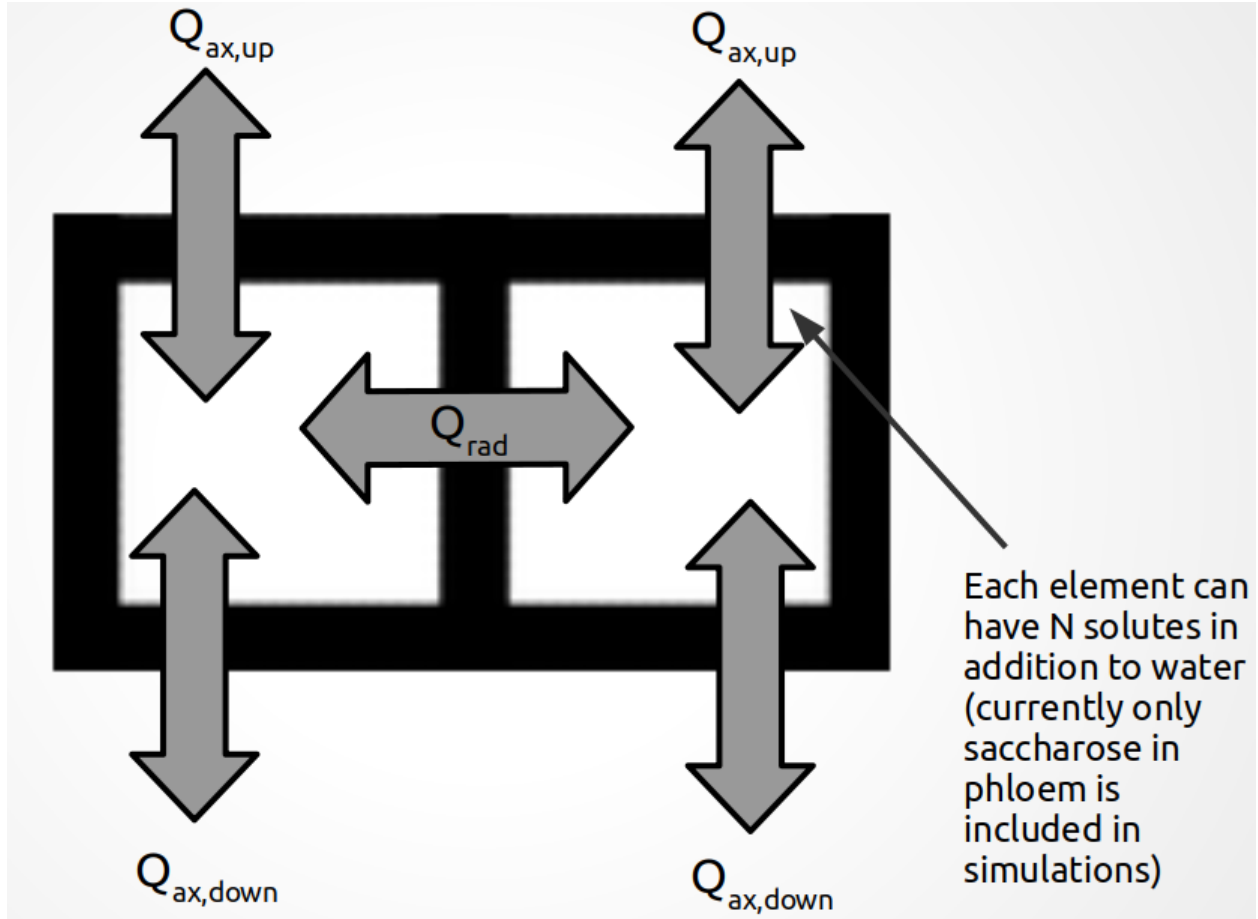
The number of elements that transpire, produce or unload sugar can be changed by the user. Currently only the bottommost element takes up water from the soil. The model is build with the idea, that the sap in both xylem and phloem can contain any number of soluts. However, currently functionality to set, retrieve and update solutes is limited only to sugar (sucrose) in the phloem.

1.2 Input and output of the model



See *instructions how to run the model* how to give the input to the model.

1.3 The fluxes inside the tree



The fluxes shown in the figure above are calculated as follows (the index i refers to rows in any 2D property array in the model). The fluxes are in the units (kg/s)

$$Q_{\{ax,i\}} = Q_{\{ax,bottom,i\}} + Q_{\{ax,top,i\}} - E_i$$

$$Q_{\{ax,bottom,i\}} = \frac{k_i}{A_{\{ax,i\}}} \frac{\rho_w}{\eta} \frac{l_i}{l_i} (P_{\{i+1\}} - P_{\{i\}} - P_h)$$

$$Q_{\{ax,top,i\}} = \frac{k_i}{A_{\{ax,i+1\}}} \frac{\rho_w}{\eta} \frac{l_i}{l_i} (P_{\{i-1\}} - P_{\{i\}} + P_h)$$

$$Q_{\{radial,phloem\}} = L_r A_{\{rad,i\}} \rho_w [P_{\{i,xylem\}} - P_{\{i,phloem\}} - \sigma(C_{\{i,xylem\}} - C_{\{i,phloem\}}) R T]$$

$$Q_{\{radial,xylem\}} = -Q_{\{radial,phloem\}}$$

where

- E_i : transpiration rate of the i th element (kg/s)
- k_i : axial permeability of the i th element (m²)
- $A_{ax,i}$: base surface area of xylem or phloem (m²)
- ρ_w : liquid phase density of water (kg/m³)
- η : viscosity of the sap in the i th element (Pa s)
- l_i : length (height) of the i th element (m)

- P_i : Pressure in the i th element (Pa)
- P_h : Hydrostatic pressure (Pa) - $P_h = \rho_w g \text{aggravitation li}$
- L_r : radial hydraulic conductivity (m/Pa/s)
- $A_{rad,i}$: lateral surface area of the xylem (m²)
- σ : reflection coefficient (Van't hof factor) (unitless)
- C_i : sugar concentration in the i th element (mol/m³)
- R : universal gas constant (J/K/mol)
- T : ambient temperature (K)

1.4 The change of pressure, sugar concentration and element radius due to sap flux

The equations are calculated based on Hölttä et al., (2006)

The change of pressure in the xylem and the phloem due to sap flux is calculated separately for both horizontal elements according to

$$\frac{dP_i}{dt} = \frac{\epsilon_i V_i \rho_w (Q_{ax,i} + Q_{rad,i})}{C_i V_i \left(\frac{C_i}{\rho_w} + L_i + U_i \right)}$$

The change in sugar concentration in the phloem is calculated according to

$$\frac{dC_i}{dt} = \frac{Q_{ax,i}}{V_i} \left(\frac{C_i}{\rho_w} + L_i + U_i \right)$$

The change in element radius in the xylem is calculated according to

$$\frac{dR_{xylem,i}}{dt} = \frac{Q_{ax,i} + Q_{rad,i}}{2 \pi \rho_w l_i (R_{xylem,i} + R_{heartwood})}$$

The change in element radius in the phloem is calculated according to

$$\frac{dR_{i,phloem}}{dt} = \frac{Q_{ax,i} + Q_{rad,i}}{2 \pi \rho_w l_i} \left(\frac{1}{\sum_{j=1}^3 R_{j,i}^3} - \frac{R_{phloem,i}}{\sum_{j=1}^2 R_{j,i}^3} \right)$$

where the sum goes from $j=1$ (the heartwood) to $j=3$ (the phloem)

- ϵ_i : elastic modulus of the i th element (Pa)
- V_i : volume of the i th element (m³)
- L_i : sugar loading rate of the i th element (mol/s)
- U_i : sugar unloading rate of the i th element (mol/s)
- ρ_w : liquid phase density of water (kg/m³)
- l_i : length (height) of the i th element (m)
- C_i : Sucrose concentration in the i th element (mol/m³)
- $Q_{ax,rad,i}$: axial or radial sap flux of the i th element (kg/s)
- R_i : radius of the i th element (m)
- P_i : pressure in the i th element (Pa)

1.5 Sugar loading and unloading rates

Currently in the model the sugar loading rate should be set equal to the photosynthesis rate when the tree is created.

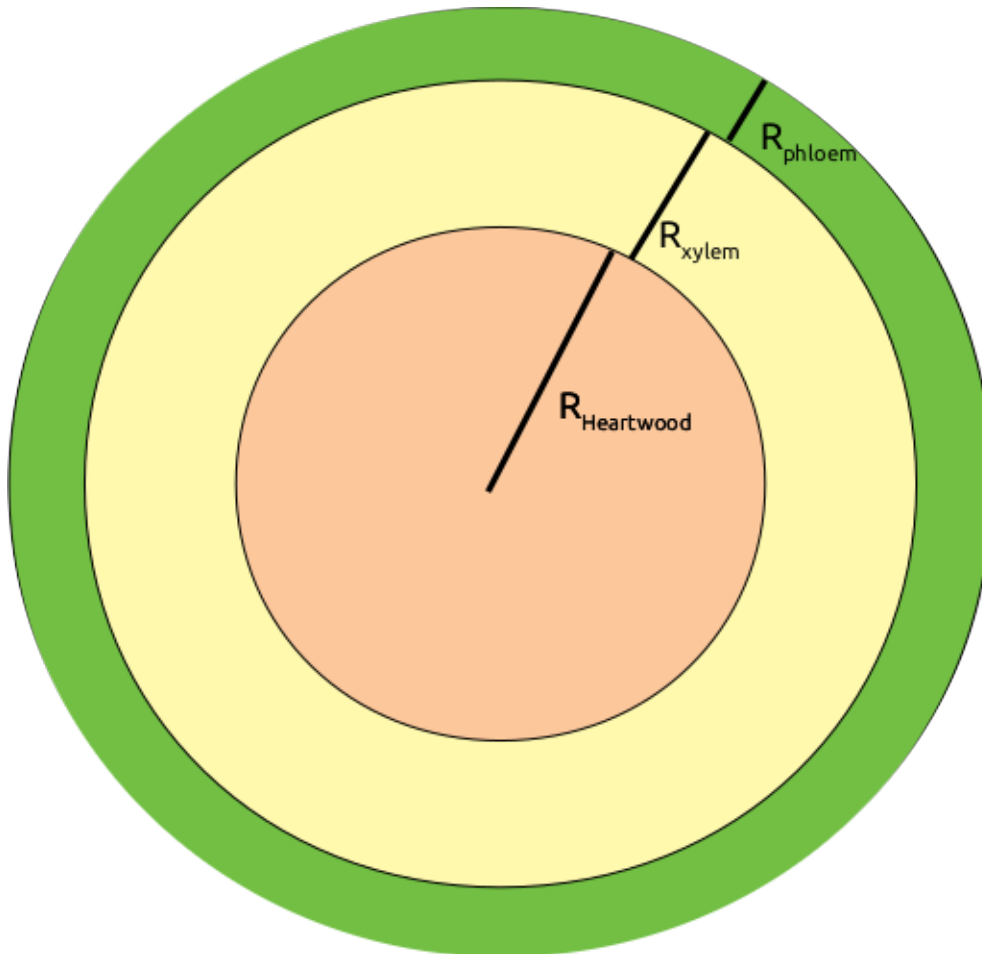
The initial unloading rate needs to be set but it is updated according to Nikinmaa et. al., (2014)

$$U_i = A_{\text{rad},i} \cdot \max\{0, u_s (C_i - C_0)\}$$

where

- $A_{\text{rad},i}$: radial surface area between the xylem and the phloem (i.e., lateral area of the xylem)
- u_s : sugar unloading slope set
- C_i : sugar concentration in the i th element
- C_0 : sugar target concentration

1.6 How the element radii are modelled



The radius of each horizontal element (heartwood, xylem and phloem) are given as the width of the element. For example, **given total tree diameter of 10 cm, heartwood radius of 2 cm and 90/10% split**

between the phloem and the xylem. The traditional radii of each element (distance from the pith) would be

- Heartwood: 2 cm
- Xylem: 4.7 cm
- Phloem 5 cm

In the model the radii are given as the width of the horizontal element, i.e.:

- Heartwood: 2 cm
- Xylem: 2.7 cm
- Phloem: 0.3 cm

INSTRUCTIONS TO RUN THE MODEL

There are two options to run the model. The quickest and easiest way is to run `main.py` which has a single profile loaded. The more challenging way is to import the `src.model` module, create instances of `tree` and `model` classes and run the model.

2.1 Running from `main.py`

The `main.py` can be run with command

```
>>> python main.py
```

Below the `main.py` is discussed line by line. Simulations that require only parameter value changes can be run by editing the `main.py`.

2.2 Importing `src.model`

2.2.1 Imports

```
from src.tools.iotools import tree_properties_to_dict, write_netcdf
from src.constants import PHLOEM_RADIUS, XYLEM_RADIUS
from src.model import Model
from src.tree import Tree
from typing import List
import numpy as np
import math
from datetime import datetime
```

The `iotools` module is used to write the simulation results to a NetCDF file. `PHLOEM_RADIUS` and `XYLEM_RADIUS` are used to define the radii of the tree horizontal elements. The `Model` and `Tree` classes are needed to create the instances of the modelled tree and to run the model. `Datetime` is used to output the current time when running the model and for setting the name of the output file.

2.2.2 Define tree properties

```
height: float = 2.37
num_elements: int = 40
```

Set the total height of the modelled tree and number of elements in the tree.

```
transpiration_profile: List[float] = [0.0 for i in range(num_elements)]
photosynth_profile: List[float] = [0 for i in range(num_elements)]
sugar_loading_profile = photosynth_profile
```

Set the initial values for both transpiration in the xylem and photosynthesis in the tree. Note that currently sugar loading profile must equal photosynthesis profile.

```
transpiration_max = 0.9e-3/600.0 # kg/s
photosynthesis_max = 2.5e-5/10.0 # mol/s

time = np.linspace(0, 24, 12*24+1)

transpiration = np.sin(time*math.pi/24.0)*transpiration_max
photosynthesis = np.sin(time*math.pi/24.0)*photosynthesis_max
```

Define the positive sine shaped curves for 24h our period that peak at the values set to maximum values (transpiration_max and photosynthesis_max). The time step in the time vector is 5 minutes.

```
sugar_profile: np.ndarray = np.zeros((num_elements, 1))
sugar_profile[0:20, 0] = 1400
sugar_profile[20:30, 0] = 1000
sugar_profile[30:40, 0] = 800
```

Set the initial sugar concentration in the phloem. Note that the tree properties can be set either as a list (as done for e.g. transpiration profile) or as NumPy arrays. In the Tree class all properties are converted to NumPy arrays with `numpy.asarray` function.

```
sugar_unloading_profile: List[float] = [0.0 for i in range(num_elements)]

sugar_target_concentration: float = 1200

sugar_unloading_slope = 3.5e-7
```

Set the initial sugar unloading rate and parameters for calculating the unloading rate further. The `sugar_target_concentration` is a concentration after which sugar unloading starts and `sugar_unloading_slope` is the slope parameter (units m/s) for the unloading. See [Nikinmaa et. al., \(2014\)](#) for further details.

```
axial_permeability_profile: np.ndarray = np.zeros((num_elements, 2))
axial_permeability_profile[:, 0] += 1.5e-13
axial_permeability_profile[:, 1] += 1.2e-13

for (row, permeability) in enumerate(axial_permeability_profile):
    axial_permeability_profile[row, 0] = np.min([permeability[0],
                                                  permeability[0]*1/math.sqrt(height/num_
↪elements*(row+1))])
    axial_permeability_profile[row, 1] = np.min([permeability[1],
                                                  permeability[1]*1/math.sqrt(height/num_
↪elements*(row+1))])
```

(continues on next page)

(continued from previous page)

```
axial_permeability_profile = np.flip(axial_permeability_profile, axis=0)
```

Define an axial permeability for the modelled tree. In this case the axial permeabilities are inversely proportional to the square root of distance of an element from the ground.

```
radial_hydr_conductivity: List[float] = [1e-13] * num_elements

elastic_modulus_profile: List[List[float]] = [[1000e6, 30e6]] * num_elements

radii: List[float] = [XYLEM_RADIUS, PHLOEM_RADIUS]

ground_water_potential: float = 0.0
```

Define the missing properties for the modelled tree. If there are multiple elements the first element is always for the xylem and the second for the phloem.

```
tree = Tree(height=height,
            num_elements=num_elements,
            initial_radius=radii,
            transpiration_profile=transpiration_profile,
            photosynthesis_profile=photosynth_profile,
            sugar_profile=sugar_profile,
            sugar_loading_profile=sugar_loading_profile,
            sugar_unloading_profile=sugar_unloading_profile,
            sugar_target_concentration=sugar_target_concentration,
            sugar_unloading_slope=sugar_unloading_slope,
            axial_permeability_profile=axial_permeability_profile,
            radial_hydraulic_conductivity_profile=radial_hydr_conductivity,
            elastic_modulus_profile=elastic_modulus_profile,
            ground_water_potential=ground_water_potential)
```

Create an instance of the Tree class with the desired properties.

```
outputfname = 'test_'
outputfname = outputfname + datetime.now().strftime("%y-%m-%dT%H:%M:%S") + ".nc"
```

Define name of the output file.

```
model = Model(tree, outputfile=outputfname)
```

Create an instance of the model class.

2.2.3 Run the model

```
for day in range(0, 2):
    for (ind, time) in enumerate(time[0:-1]):
        # set new transpiration rate
        transpiration_rate = transpiration_profile.copy()
        transpiration_rate[0:10] = [transpiration[ind]]*10
        model.tree.transpiration_rate = np.asarray(transpiration_rate).reshape(40, 1)

        photosynthesis_rate = photosynth_profile.copy()
        photosynthesis_rate[0:10] = [photosynthesis[ind]]*10
```

(continues on next page)

(continued from previous page)

```
model.tree.photosynthesis_rate = np.asarray(photosynthesis_rate).reshape(40, 1)
model.tree.sugar_loading_rate = model.tree.photosynthesis_rate.copy()

model.run_scipy(time_start=(day*60*60*24)+time*60*60, time_
↪end=(day*60*60*24)+time[ind+1]*60*60, ind=ind)
```

The model is run by calling the `model.run_scipy` method repeatedly. The outer loop defines how many days the model is run and the inner loop defines the current simulation time in a day.

In every time step new values for the transpiration rate and photosynthesis rate are set. Note that the `sugar_loading_rate` is set equal to the photosynthesis rate as is required in the current version of the model.

Finally the model is run from `time_start` to `time_end` which is always a 5 minute time interval. Afterwards, the last stage of the tree is saved in the `model.run_scipy` method.

INSTALLATION

3.1 Download the source

```
>>> git clone git@github.com:LukeEcomod/TreeBoxModel.git
```

or

download the source <https://github.com/LukeEcomod/TreeBoxModel>

3.2 Install the required packages

Ideally you have created a new virtual environment for this project.

To install all the packages required for the model to run use

```
>>> pip install -r requirements.txt
```

or

```
>>> conda install --file requirements.txt
```


QUICK START

run main.py

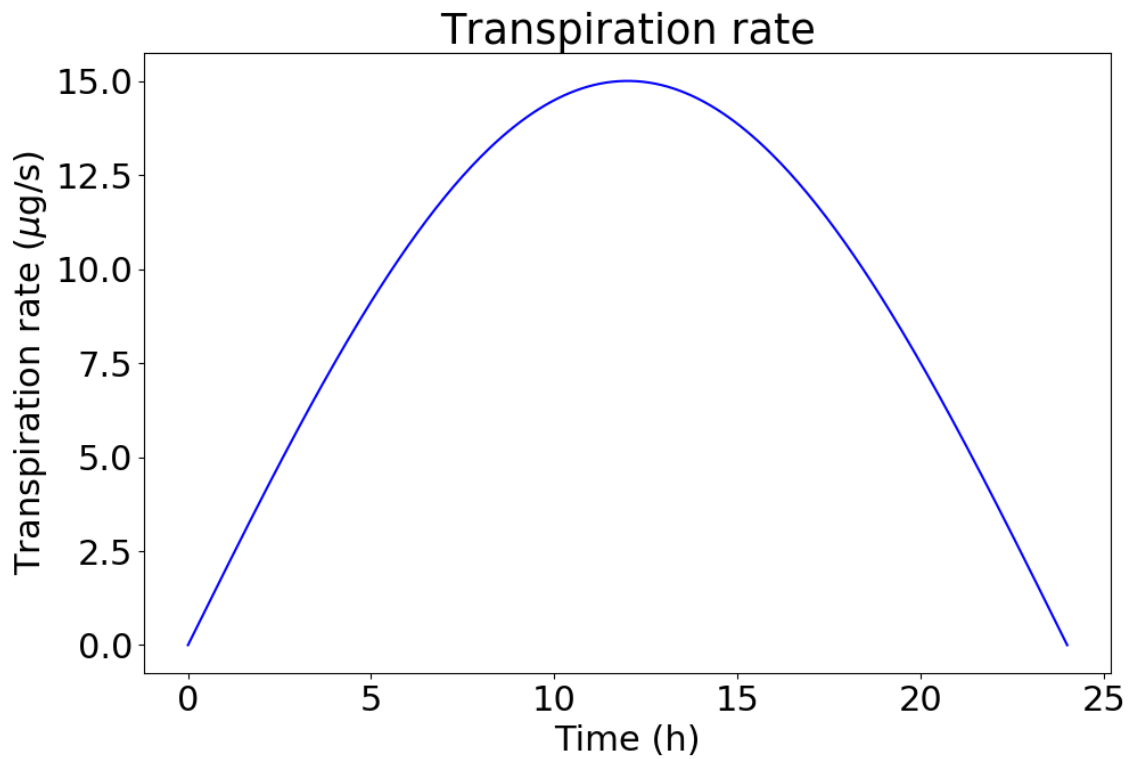
```
>>> python main.py
```


MAIN.PY

The purpose of this main file is to provide an easy way to run the model.

All the model parameters are set in the file and are taken from [Hölttä et. al. 2006](#) or [Nikinmaa et. al., \(2014\)](#).

A sine-like behaviour is assumed for the transpiration and photosynthesis



MODULES, CLASSES & FUNCTIONS

`class src.model.Model(tree: src.tree.Tree, outputfile: str = 'a.nc')`

Calculates the next time step for given tree and saves the tree stage.

Provides functionality for solving the ordinary differential equations (ODE) describing the behaviour of the modelled system.

Parameters

- **tree** (`Tree`) – instance of the tree class for which the ODEs are solved
- **outputfile** (`str`) – name of the file where the NETCDF4 output is written

tree

instance of the tree class for which the ODEs are solved

Type `Tree`

ncf

the output file

Type `netCDF4.Dataset`

axial_fluxes() → `numpy.ndarray`

Calculates axial sap mass flux for every element.

The axial flux in the xylem and phloem are calculated independently from the sum of bottom and top fluxes

$$Q_{ax,i} = Q_{ax,bottom,i} + Q_{ax,top,i} - E$$

$$Q_{ax,bottom,i} = \frac{k_i A_{ax,i} \rho_w}{\eta_i l_i} (P_{i+1} - P_i - P_h)$$

$$Q_{ax,top,i} = \frac{k_i A_{ax,i+1} \rho_w}{\eta_i l_i} (P_{i-1} - P_i + P_h)$$

where

- E_i : transpiration rate of the i th element ($\frac{kg}{s}$)
- k_i : axial permeability of the i th element (m^2)
- $A_{ax,i}$: base surface area of xylem or phloem (m^2)
- ρ_w : liquid phase density of water ($\frac{kg}{m^3}$)
- η : viscosity of the sap in the i th element ($Pa s$)
- l_i : length (height) of the i th element (m)

- P_i : Pressure in the i th element (Pa)
- P_h : Hydrostatic pressure (Pa) $P_h = \rho_w a_{gravitation} l_i$

Returns *numpy.ndarray* (*dtype=float, ndim=2*)[*self.tree.num_elements, 2*] – The axial fluxes in units kg/s

radial_fluxes() → *numpy.ndarray*

Calculates radial sap mass flux for every element.

The radial flux for the phloem of the i th axial is calculated similar to Hölttä et. al. 2006

$$Q_{radial,phloem} = L_r A_{rad,i} \rho_w [P_{i,xylem} - P_{i,phloem} - \sigma(C_{i,xylem} - C_{i,phloem})RT]$$

where

- L_r : radial hydraulic conductivity ($\frac{m}{Pa \cdot s}$)
- $A_{rad,i}$: lateral surface area of the xylem (m^2)
- ρ_w : liquid phase density of water ($\frac{kg}{m^3}$)
- P_i : Pressure in the i th element (Pa)
- σ : Reflection coefficient (Van't hof factor) (unitless)
- C_i : Sucrose concentration in the i th element ($\frac{mol}{m^3}$)
- R : Universal gas constant ($\frac{J}{K \cdot mol}$)
- T : Ambient temperature (K)

The radial flux for the xylem is equal to the additive inverse of the phloem flux

$$Q_{radial,xylem} = -Q_{radial,phloem}$$

Returns *numpy.ndarray* (*dtype=float, ndim=2*)[*self.tree.num_elements, 2*] – The radial fluxes in units kg/s

run(*time_start: float = 0.001, time_end: float = 120.0, dt: float = 0.01, output_interval: float = 60*) → None

Propagates the tree in time using explicit Euler method (very slow).

NB! This function needs to be updated. Use `run_scipy` instead!

Parameters

- **time_start** (*float*) – Time in seconds where to start the simulation.
- **time_end** (*float*) – Time in seconds where to end the simulation.
- **dt** (*float*) – time step in seconds
- **output_interval** – Time interval in seconds when to save the tree stage

run_scipy(*time_start: float = 0.001, time_end: float = 120.0, ind: int = 0*) → None

Propagates the tree in time using the `solve_ivp` function in the SciPy package.

The stage of the tree is saved only at the start of the simulation if `time_start < 1e-3` and at `time_end`. If the tree stage is desired on multiple time points the function needs to be called recurrently by splitting the time interval into multiple sub intervals.

Parameters

- **time_start** (*float*) – Time in seconds where to start the simulation.

- **time_ned** (*float*) – Time in seconds where to end the simulation.
- **ind** (*int*) – index which refers to the index in model.outputfile. The last stage of the tree is saved to model.outputfile[ind].

```
class src.tree.Tree(height: float, initial_radius: List[float], num_elements: int, transpiration_profile: List[float], photosynthesis_profile: List[float], sugar_profile: List[float], sugar_loading_profile: List[float], sugar_unloading_profile: List[float], sugar_target_concentration: float, sugar_unloading_slope: float, axial_permeability_profile: List[List[float]], radial_hydraulic_conductivity_profile: List[float], elastic_modulus_profile: List[List[float]], ground_water_potential: float)
```

Model of a tree.

Provides properties and functionality for saving and editing the modelled tree. Arguments whose type is List[float] or List[List[float]] are converted to numpy.ndarray with numpy.asarray method. Thus, also numpy.ndarray is a valid type for these arguments.

For arguments whose type is List[float] (except for initial_radius) the length of the arguments must be equal to num_elements. The order of the list should be from the top of the tree (the first item) to the bottom of the tree (the last item)

For arguments whose type is List[List[float]] the length of the arguments must be equal to num_elements and each sub list must contain two elements, one for the xylem and one for the phloem in this order. The order of the sub lists should be from the top of the tree (the first sub list) to the bottom of the tree (the last sub list).

Parameters

- **height** (*float*) – total tree height (*m*)
- **initial_radius** (*List [float] or numpy.ndarray*) – the radius of the xylem and the phloem (*m*) in this order. See from the [modelled system](#), how the radii should be given. Only two values can be given and the radius of each element is set to be the same in the tree initialization.
- **num_elements** (*int*) – number of vertical elements in the tree. The height of an element is determined by $\text{element height} = \frac{\text{tree height}}{\text{number of elements}}$
- **transpiration_profile** (*List [float] or numpy.ndarray*) – The rate of transpiration ($\frac{\text{kg}}{\text{s}}$) in the xylem. The length of the list must be equal to num_elements and the order is from the top of the tree (first value) in the list to the bottom of the tree (last value in the list).
- **photosynthesis_profile** (*List [float]*) – The rate of photosynthesis ($\frac{\text{mol}}{\text{s}}$). Currently this variable is not used and the rate of photosynthesis should be equal to the sugar_loading_profile.
- **sugar_profile** (*List [float]] or numpy.ndarray*) – The initial sugar (sucrose) concentration in the phloem ($\frac{\text{mol}}{\text{m}^3}$)
- **sugar_loading_profile** (*List [List [float]] or numpy.ndarray*) – the rate at which sugar concentration increases in each phloem element ($\frac{\text{mol}}{\text{s}}$)
- **sugar_unloading_profile** (*List [float] or numpy.ndarray*) – The initial sugar unloading rate (the rate at which the sugar concentration decreases in a given phloem element) ($\frac{\text{mol}}{\text{s}}$). The unloading rate is updated in `src.odefun.odefun`.
- **sugar_target_concentration** (*float*) – the target concentration after which the sugar unloading starts ($\frac{\text{mol}}{\text{m}^3}$)

- **sugar_unloading_slope** (*float*) – the slope parameter for unloading (see Nikinmaa et. al., (2014)).
- **axial_permeability_profile** (*List[List[float]] or numpy.ndarray*) – axial permeabilities of both xylem and phloem (m^2)
- **radial_hydraulic_conductivity_profile** (*List[float] or numpy.ndarray*) – radial hydraulic conductivity between the xylem and the phloem ($\frac{m}{Pa\ s}$)
- **elastic_modulus_profile** (*List[List[float]] or numpy.ndarray*) – Elastic modulus of every element (Pa).
- **ground_water_potential** (*float*) – The water potential in the soil. This is used to calculate the sap flux between soil and the bottom xylem element.

height

total tree height (m)

Type float

num_elements

number of vertical elements in the tree.

Type float

transpiration_rate

The rate of transpiration ($\frac{kg}{s}$) in the xylem.

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 1]

photosynthesis_rate

The rate of photosynthesis ($\frac{mol}{s}$). Currently this variable is not used.

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 1]

sugar_loading_rate

The rate at which sugar concentration increases in each phloem element ($\frac{mol}{s}$).

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 1]

sugar_unloading_rate

The rate at which the sugar concentration decreases in a given phloem element ($\frac{mol}{s}$).

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 1]

sugar_target_concentration

The target concentration after which the sugar unloading starts ($\frac{mol}{m^3}$).

Type float

sugar_unloading_slope

The slope parameter for unloading (see [Nikinmaa et. al., (2014)](<https://academic.oup.com/aob/article/114/4/653/2769025>)).

Type float

solutes

Array of src.solute.Solute which contain the solutes in the sap of xylem and phloem.

Type numpy.ndarray(dtype=src.solute.Solute, ndim=2) [tree.num_elements, 2]

axial_permeability

Axial permeabilities of both xylem and phloem (m^2).

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

radial_hydraulic_conductivity

Radial hydraulic conductivity between the xylem and the phloem ($\frac{m}{Pa\ s}$).

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 1]

elastic_modulus

Elastic modulus of every element (Pa).

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

ground_water_potential

The water potential in the soil.

Type float

pressure

Pressure of each element (Pa)

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

element_radius

Radius of each element (m)

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

element_height

Height of each element (m)

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

viscosity

The dynamic viscosity of each element ($Pa\ s$)

Type numpy.ndarray(dtype=float, ndim=2) [tree.num_elements, 2]

cross_sectional_area(ind: List[int] = None) → numpy.ndarray

Calculates the cross-sectional area between the xylem and the phloem.

The cross sectional area is equal to lateral surface area of the xylem.

Parameters **ind** (*List[int] or numpy.ndarray(dtype=int, ndim=1), optional*) – the indices of the elements for which the cross-sectional area is calculated. If no ind is given, the cross-sectional area is calculated for every element.

Returns *numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]* – Cross-sectional area between the xylem and phloem elements (m^2)

element_area(ind: List[int] = None, column: int = 0) → numpy.ndarray

Calculates the base area of the xylem or the phloem.

Parameters

- **ind** (*List[int] or numpy.ndarray(dtype=int, ndim=1), optional*) – the indices of the elements for which the base area is calculated. If no ind is given, the base area is calculated for every element.
- **column** (*int, optional*) – The column in the tree grid for which the base area is calculated. use column=0 for the xylem and column=1 for the phloem. If not column is given returns the base area for the xylem.

Returns *numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]* – Base area of either the xylem or the phloem (m^2)

element_volume(ind: List[int] = None, column: int = 0) → numpy.ndarray

Calculates the volume of the xylem or the phloem.

Parameters

- **ind** (*List[int]* or *numpy.ndarray(dtype=int, ndim=1)*, *optional*) – the indices of the elements for which the volume is calculated. If no ind is given, the volume is calculated for every element.
- **column** (*int*, *optional*) – The column in the tree grid for which the volume is calculated. use column=0 for the xylem and column=1 for the phloem. If not column is given returns the volume for the xylem.

Returns *numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]* – Volume of either the xylem or the phloem (m^3)

sugar_concentration_as_numpy_array() → *numpy.ndarray*

Transforms the phloem sugar concentration in [self.solutes](#) into *numpy.ndarray*.

Returns *numpy.ndarray(dtype=float, ndim=2) [self.num_elements, 1]* – The sugar concentration in the phloem. ($\frac{mol}{m^3}$)

update_sap_viscosity() → None

Calculates and sets the viscosity in the phloem according to the sugar concentration.

The sap viscosity is calculated according to Morrison (2002)

$$\eta = \eta_w \exp \frac{4.68 \cdot 0.956\Phi_s}{1 - 0.956\Phi_s}$$

where

- η_w : Dynamic viscosity of water ($\eta_w \approx 0.001$)
- Φ_s : Volume fraction of sugar (sucrose) in the phloem sap.

References

Morison, Ken R. “Viscosity equations for sucrose solutions: old and new 2002.” Proceedings of the 9th APCCChE Congress and CHEMECA. 2002.

update_sugar_concentration(new_concentration: numpy.ndarray) → None

Sets the sugar concentration in [self.solutes](#) to new_concentration.

Parameters new_concentration (*numpy.ndarray(dtype=float, ndim=2) [self.num_elements, 1]*) – new concentration values. the order is from top of the tree (first element, new_concentration[0]) to bottom of the tree (last element, new_concentration[self.num_elements-1]) ($\frac{mol}{m^3}$)

src.odefun.odefun(t: float, y: numpy.ndarray, model) → *numpy.ndarray*

Calculates the right hand side of the model ODEs.

The modelled system and the ODEs are described in the [modelled system](#). The `scipy.solve_ivp()` function in `src.model.Model.run_scipy()` method calls this function during the simulation.

Parameters

- **t** (*float*) – time in the model simulation
- **y** (*numpy.ndarray(dtype=float, ndims=1)[5 · model.tree.num_elements,]*) – 1D array where elements 0:2 · model.tree.num_elements are the pressures in the xylem and phloem of the tree, elements 2 · model.tree.num_elements:3 · model.tree.num_elements are for the sucrose concentration in the phloem and elements 3 · model.tree.num_elements:5 · model.tree.num_elements are for the element radii both in the xylem and the phloem.

- **model** (`src.model.Model`) – Instance of the model class

Returns (`numpy.ndarray(dtype=float, ndims=1)[5 · model.tree.num_elements,]`) – 1D array of the right hand side values of the model ODEs where the elements 0:2 · model.tree.num_elements are $\frac{d(\text{pressure})}{dt}$, elements 2 · model.tree.num_elements: 3 · model.tree.num_elements are $\frac{d[C(\text{sucrose})]}{dt}$ and elements 3 · model.tree.num_elements:5 · model.tree.num_elements are $\frac{d(\text{radius})}{dt}$

`src.constants.MAX_ELEMENT_COLUMNS: int = 2`
Max number of columns in the `tree` class

`src.constants.HEARTWOOD_RADIUS = 0.0444`
The radius of the heartwood (m)

`src.constants.TEMPERATURE: float = 298.0`
Temperature of the tree (K)

`src.constants.M_WATER: float = 0.0182`
Molar mass of water ($\frac{kg}{mol}$)

`src.constants.RHO_WATER: float = 1000`
density of liquid water ($\frac{kg}{m^3}$)

`src.constants.VISCOSITY_WATER: float = 0.001`
dynamic viscosity of water ($Pa \cdot s$)

`src.constants.M_SUCROSE: float = 0.3423`
molar mass of sucrose ($\frac{kg}{mol}$)

`src.constants.RHO_SUCROSE: float = 1590.0`
density of sucrose ($\frac{kg}{m^3}$)

`src.constants.GRAVITATIONAL_ACCELERATION: float = 9.81`
acceleration due to Earth's gravity ($\frac{m}{s^2}$)

`src.constants.AVOGADROS_CONSTANT: float = 6.022e+23`
avogadro's constant ($\frac{1}{mol}$)

`src.constants.MOLAR_GAS_CONSTANT: float = 8.3145`
molar gas constant ($\frac{J}{K \cdot mol}$)

`src.model_variables = Name, descriptions, unit, dimension and precision of each variable that is saved`

`src.tools.iotools.initialize_netcdf(filename: str, axial_elements: int, variables: Dict) → netCDF4._netCDF4.Dataset`
Initializes a netcdf file to be ready for saving simulation results.

Parameters

- **filename** (*str*) – name of the NETCDF4 file that is created
- **axial_elements** (*int*) – Number of axial elements in the Tree
- **variables** (*Dict*) – The names, descriptions, units, dimensions and precision of each variable that is saved to the netcdf file. The key of each dictionary element is used to label the variables. The value of each dictionary element needs to be a list where list[0] (*str*) Description of the variable list[1] (*str*): unit of the variable list[2] (*Tuple*): NETCDF dimensions of the variable list[3] (*str*): Datatype (precision) of the variable The possible NETCDF dimensions are “index”, “radial_layers” and “axial_layers”.

Returns (`NETCDF4.Dataset`) – A NETCDF4 file where the simulation results can be saved.

`src.tools.iotools.tree_properties_to_dict(tree: src.tree.Tree) → Dict`

Transfers tree properties into a dictionary.

Parameters `tree` ([Tree](#)) – Instance of the tree class.

Returns (*Dict*) – Dictionary of the tree properties.

`src.tools.iotools.write_netcdf(ncf: netCDF4._netCDF4.Dataset, results: Dict) → None`

Write a simulation result dictionary into a netcdf file.

The variables that can be written are defined in the `src.model_variables` file

Parameters

- `ncf` ([netCDF4.Dataset](#)) – the netcdf file where the results are written
- `results` (*Dict*) – the results dictionary. Use the [tree_properties_to_dict](#) function to create the dictionary.

`src.tools.plotting.plot_phloem_pressure_top_bottom(filename: str) → None`

Plot the phloem pressure at the top and bottom of the tree.

The figure is saved in the current working directory with the same name as the filename with “_phloem_pressure.png” appended.

Parameters `filename` (*str*) – name of the NetCDF file in the current directory that includes the simulation results.

`src.tools.plotting.plot_simulation_results(filename: str, foldername: str) → None`

Plot sugar concentration, fluxes and pressures from one simulation.

Currently you need edit the start and end indices in the code to capture the correct time window in the simulation results. The indices refer to the index dimension in the NetCDF files.

Parameters

- `filename` (*str*) – name of the NetCDF file in the current directory that includes the simulation results.
- `foldername` (*str*) – name of the folder where the figures are saved

`src.tools.plotting.plot_variable_vs_time(filename: str, params: Dict = None) → None`

Plot any variable as a function of time from the NetCDF file that contains the simulation results.

Parameters

- `filename` (*str*) – name of the NetCDF file in the current directory that includes the simulation results.
- `params` (*Dict*, *optional*) – Parameters and instructions for making the plot (see Examples)

Examples

An example of the params dictionary is

```
params = {'variable_name': 'pressure',
'time_divide': 86400,
'variable_divide': 1e6
'labels': [['top'], ['bottom'], ['middle']],
'line_colors': [['k'], ['r'], ['b']],
'line_widths': [[3], [3], [3]],
'cut': {'index': range(1500),
```

(continues on next page)

(continued from previous page)

```

        'axial_layers': [0, 39],
        'radial_layers': [1]},
'xticks': np.linspace(0, 10, 11),
'xlabel': 'Time (d)'
'ylabel': 'Pressure (MPa)',
'title': 'Pressure in the xylem'
'folder', 'figure/',
'filename_ending': 'xylem_pressure.png'}
```

- Variable name is the name of the variable that is to be plotted. The name must equal to a variable in the NetCDF file
- time_divide is a float which is used to divide the time vector (x-axis) in the plot. e.g., 86400 converts seconds to days
- variable_divide is a float which is used to divide the variable vector (y-axis) in the plot
- labels are the labels of each line that is drawn
- line_colors are the colors of each line that is drawn
- line_widths are the line widths of each line that is drawn
- cut contains the data range in the NetCDF file that are plotted. The function can handle only 3-dimensional data. If the variable that is plotted has shape (1500,2,1) like in this example, the function expects that there will be 2*1=3 lines in the plot. Consequently, the labels, line_colors, and line_widths values in the params dictionary need to have 3 elements like in this example.
- xticks are the matplotlib.pyplot.xticks function argument
- xlabel is the matplotlib.pyplot.xlabel function argument
- ylabel is the matplotlib.pyplot.ylabel function argument
- title is the matplotlib.pyplot.title function argument
- folder refers to the folder starting from the current working directory where the figure is saved
- filename_ending is a string that is appended to the filename when the figure is saved
- If no folder is specified the argument filename and filename_ending is used to create the name of the figure that is saved.

`src.tools.plotting.plot_xylem_pressure_top_bottom(filename: str) → None`

Plot the xylem pressure at the top and bottom of the tree.

The figure is saved in the current working directory with the same name as the filename with “_xylem_pressure.png” appended.

Parameters filename (*str*) – name of the NetCDF file in the current directory that includes the simulation results.

PYTHON MODULE INDEX

m

`main`, [15](#)

S

`src.model`, [17](#)

`src.solute`, [22](#)

`src.tools.iotools`, [23](#)

`src.tools.plotting`, [24](#)

`src.tree`, [19](#)

INDEX

A

AVOGADROS_CONSTANT (*in module src.constants*), 23
axial_fluxes() (*src.model.Model method*), 17
axial_permeability (*src.tree.Tree attribute*), 20

C

cross_sectional_area() (*src.tree.Tree method*), 21

E

elastic_modulus (*src.tree.Tree attribute*), 21
element_area() (*src.tree.Tree method*), 21
element_height (*src.tree.Tree attribute*), 21
element_radius (*src.tree.Tree attribute*), 21
element_volume() (*src.tree.Tree method*), 21

G

GRAVITATIONAL_ACCELERATION (*in module src.constants*), 23
ground_water_potential (*src.tree.Tree attribute*), 21

H

HEARTWOOD_RADIUS (*in module src.constants*), 23
height (*src.tree.Tree attribute*), 20

I

initialize_netcdf() (*in module src.tools.iotools*), 23

M

M_SUCROSE (*in module src.constants*), 23
M_WATER (*in module src.constants*), 23
main
 module, 15
MAX_ELEMENT_COLUMNS (*in module src.constants*), 23
Model (*class in src.model*), 17
model_variables (*in module src*), 23
module
 main, 15
 src.model, 17
 src.solute, 22

src.tools.iotools, 23
src.tools.plotting, 24
src.tree, 19

MOLAR_GAS_CONSTANT (*in module src.constants*), 23

N

ncf (*src.model.Model attribute*), 17
num_elements (*src.tree.Tree attribute*), 20

O

odefun() (*in module src.odefun*), 22

P

photosynthesis_rate (*src.tree.Tree attribute*), 20
plot_phloem_pressure_top_bottom() (*in module src.tools.plotting*), 24
plot_simulation_results() (*in module src.tools.plotting*), 24
plot_variable_vs_time() (*in module src.tools.plotting*), 24
plot_xylem_pressure_top_bottom() (*in module src.tools.plotting*), 25
pressure (*src.tree.Tree attribute*), 21

R

radial_fluxes() (*src.model.Model method*), 18
radial_hydraulic_conductivity (*src.tree.Tree attribute*), 20
RHO_SUCROSE (*in module src.constants*), 23
RHO_WATER (*in module src.constants*), 23
run() (*src.model.Model method*), 18
run_scipy() (*src.model.Model method*), 18

S

solutes (*src.tree.Tree attribute*), 20
src.model
 module, 17
src.solute
 module, 22
src.tools.iotools
 module, 23
src.tools.plotting

module, 24
src.tree
 module, 19
sugar_concentration_as_numpy_array()
 (*src.tree.Tree* method), 22
sugar_loading_rate (*src.tree.Tree* attribute), 20
sugar_target_concentration (*src.tree.Tree* at-
tribute), 20
sugar_unloading_rate (*src.tree.Tree* attribute), 20
sugar_unloading_slope (*src.tree.Tree* attribute), 20

T

TEMPERATURE (*in module src.constants*), 23
transpiration_rate (*src.tree.Tree* attribute), 20
Tree (*class in src.tree*), 19
tree (*src.model.Model* attribute), 17
tree_properties_to_dict() (*in module*
src.tools.iotools), 23

U

update_sap_viscosity() (*src.tree.Tree* method), 22
update_sugar_concentration() (*src.tree.Tree*
method), 22

V

viscosity (*src.tree.Tree* attribute), 21
VISCOSITY_WATER (*in module src.constants*), 23

W

write_netcdf() (*in module src.tools.iotools*), 24