Tree Box Model

Release 0.3

Olli-Pekka Tikkanen

May 26, 2020

DETAILS OF THE MODEL

CHAPTER

ONE

DESCRIPTION OF THE MODELLED SYSTEM

Modelled system is a tree

CHAPTER

TWO

INSTRUCTIONS TO RUN THE MODEL

There are two options to run the model

2.1 Running from main.py

2.2 Importing src.model

CHAPTER

THREE

INSTALLATION

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

or

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

CHAPTER

FOUR

QUICK START

run main.py

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

CHAPTER

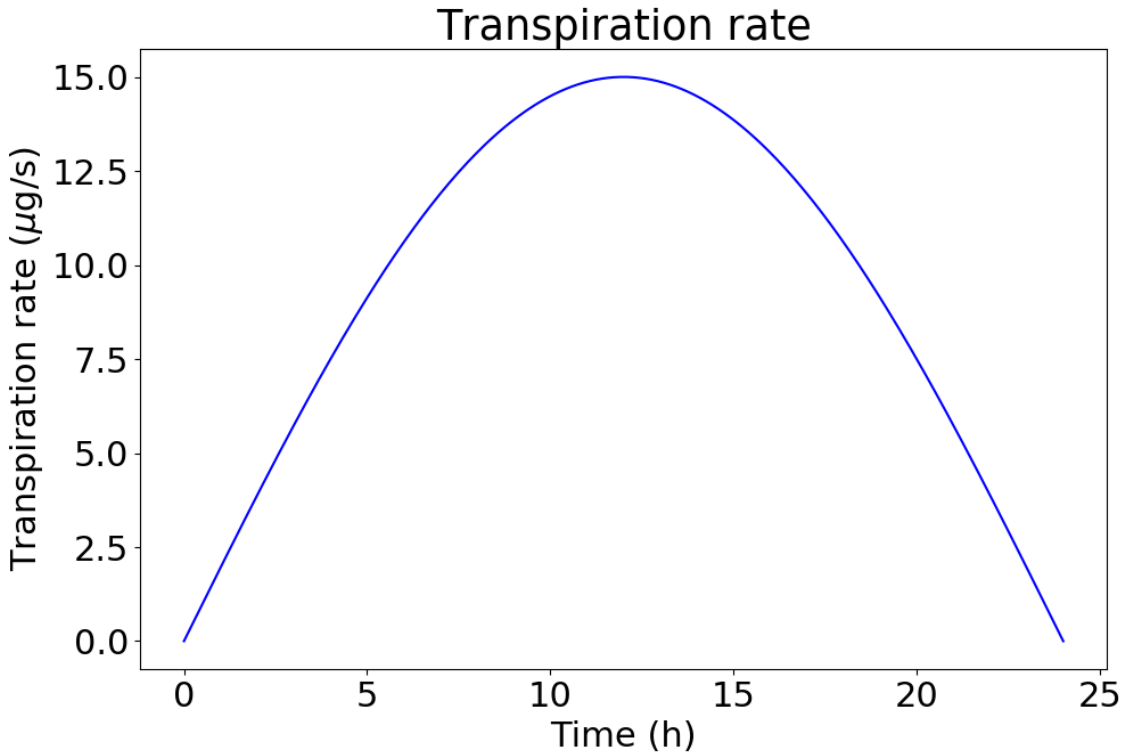
FIVE

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



CHAPTER

SIX

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*

outputfile

name of the file where the output is written

Type str

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

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

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

where

k_i: axial permeability of the ith 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 ith element $(Pa\ s)$ l_i: length (height) of the ith element (m) P_i: Pressure in the ith element (Pa) P_h: Hydrostatic pressure (Pa) $P_h = \rho_w a_{gravitation} l_i$

Returns The axial fluxes in units kg/s

Return type numpy.ndarray (dtype=float, ndim=2)[self.tree.num_elements, 2]

radial_fluxes() → numpy.ndarray
Calculates radial sap mass flux for every element.

The radial flux for the phloem of the ith 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 ($\frac{m}{Pa\ s}$) $A_{rad,i}$: lateral surface area of the xylem (m^2) ρ_w: liquid phase density of water ($\frac{kg}{m^3}$) P_i: Pressure in the ith element (Pa) σ: Reflection coefficient (Van't hoff factor) (unitless) C_i: Sucrose concentration in the ith element ($\frac{mol}{m^3}$) R: Universal gas constant ($\frac{J}{K\ mol}$) 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 The radial fluxes in units kg/s

Return type numpy.ndarray (dtype=float, ndim=2)[self.tree.num_elements, 2]

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_ned** (*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 arguemnts 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 arguemnts 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 elemenets in the tree. The height of an element is determined by element height $= \frac{\text{tree height}}{\text{number of elements}}$
- **transpiration_profile** (*List[float] or numpy.ndarray*) – The rate of transpiration ($\frac{kg}{s}$) 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 ($\frac{mol}{s}$). 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 ($\frac{mol}{m^3}$)
- **sugar_loading_profile** (*List[List[float]] or numpy.ndarray*) – the rate at which sugar concentration increases in each phloem element ($\frac{mol}{s}$)
- **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) ($\frac{mol}{s}$). The unloading rate is updated in src.odefun.odefun.
- **sugar_target_concentration** (*float*) – the target concentration after which the sugar unloading starts ($\frac{mol}{m^3}$)
- **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 elemenets 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 xylemn 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-sectinoal area is calculated. If no ind is given, the cross-sectional area is calculated for every element.

Returns Cross-sectional area between the xylem and phloem elements (m^2)

Return type numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]

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 Base area of either the xylem or the phloem (m^2)

Return type numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]

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 Volume of either the xylem or the phloem (m^3)

Return type numpy.ndarray(dtype=float, ndim=2) [len(ind) or self.num_elements, 1]

sugar_concentration_as_numpy_array() → numpy.ndarray
Transforms the phloem sugar concentration in self.solutes into numpy.ndarray.

Returns The sugar concentration in the phloem. ($\frac{mol}{m^3}$)

Return type numpy.ndarray(dtype=float, ndim=2) [self.num_elements, 1]

update_sap_viscosity() → None
Calculates and sets the viscosity in the phloem according to the sugar concenration.

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 APCChE 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]) ($\frac{mol}{m^3}$)

class src.solute.**Solute**(*name: str*, *molar_mass: float*, *density: float*, *concentration: float*)
Contains the variables to model a solute compound

Parameters

- **name** (*str*) – Name of the compound
- **molar_mass** (*float*) – Molar mass of the compound ($\frac{kg}{mol}$)
- **density** – Liquid phase density of the compound ($\frac{kg}{m^3}$)
- **concentration** – Concentration of the compound in the sap solution ($\frac{mol}{m^3}$)

name
Name of the compound

Type str

molar_mass
Molar mass of the compound ($\frac{kg}{mol}$)

Type float

density
Liquid phase density of the compound ($\frac{kg}{m^3}$)

concentration
Concentration of the compound in the sap solution ($\frac{mol}{m^3}$)

src.odefun.**odefun**(*t*, *y*, *model*)
Calculates the right hand side of the model ODEs.

src.tools.iotools.**initialize_netcdf**(*model*, *variables: Dict*) → netCDF4._netCDF4.Dataset

src.tools.iotools.**tree_properties_to_dict**(*tree:* src.tree.Tree) → Dict
cast tree properties to dictionary for saving

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

src.tools.plotting.**plot_phloem_pressure_top_bottom**(*filename: str*)

src.tools.plotting.**plot_simulation_results**(*filename: str*, *foldername: str*)

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

src.tools.plotting.**plot_xylem_pressure_top_bottom**(*filename: str*)

PYTHON MODULE INDEX

m

s

INDEX

T

U

V

W