

# Overview

## bgc\_md2 in Action

- Tables, Views and Queries

- Single Model inspection

## User Interface

- invisible graphs

- making them visible

## Structure

- Classes and Functions

- A Record

- Relation to other Python Packages

## Applications

# The Biogeochemical Model Database bgc\_md2

A screenshot of a JupyterLab interface displaying the TECOModelComparison notebook. The interface includes a browser window at the top showing the local host address and various tabs. The JupyterLab window has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and viewing. The notebook content shows a list of models with expandable sections:

- expand** cable\_general
- expand** TECOmm
- expand** TECO
- expand** Lux2012TE
- collapse** (expanded section showing a detailed diagram and equations)

The detailed section for the collapsed model includes a diagram illustrating the relationships between various carbon pools and fluxes. The diagram shows nodes for  $C_{wood}$ ,  $C_{foliage}$ ,  $C_{passsom}$ ,  $C_{slowsom}$ ,  $C_{roots}$ ,  $C_{metlit}$ , and  $C_{stilt}$ . Arrows indicate the flow between these pools, with some labeled as linear (blue) or nonlinear (green). A legend indicates: blue for linear, red for no state dependence, and grey for undetermined. The diagram also shows a flow from  $C_{wood}$  to  $C_{foliage}$  and from  $C_{foliage}$  to  $C_{passsom}$ .

Below the diagram, the following equations are listed:

$$\begin{array}{l} 27G_{B_{top}}(1-e^{-t^2})(C_{s}-C_{i}) \\ 6254 \\ 27G_{B_{top}}(1-e^{-t^2})(C_{s}-C_{i}) \\ 6254 \\ 27G_{B_{top}}(1-e^{-t^2})(C_{s}-C_{i}) \\ 6254 \\ 0 \\ 0 \end{array}$$

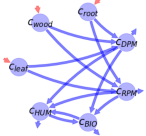
# Analysis with symbolic tools ...

inspectModel Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

Out[3]:



linear no state dependence undetermined

linear

```
In [28]: mvs.get_CompartmentalMatrix()
Out[28]:
```

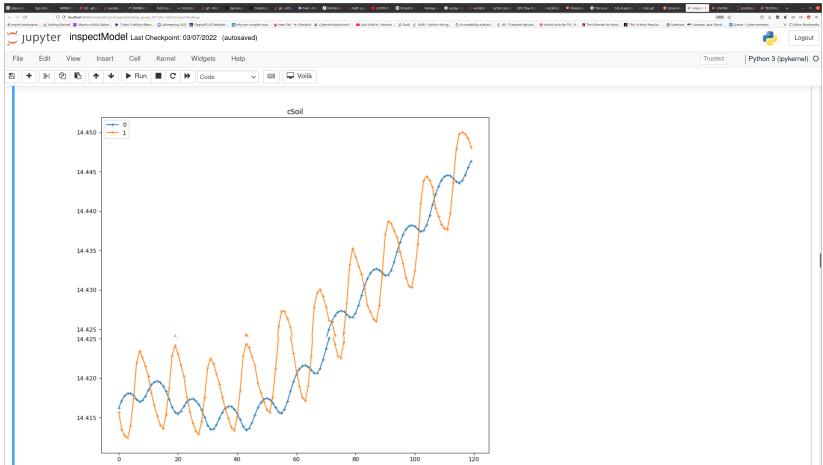
$-f_{\text{root}2\text{leaf}}$	0	0	0	0	0	0
0	$-f_{\text{wood}2\text{leaf}}$	$f_{\text{root}2\text{leaf}}$	0	0	0	0
0	0	0	$-f_{\text{leaf}2\text{root}}$	$f_{\text{wood}2\text{leaf}}$	0	0
$f_{\text{root}2\text{leaf}}$	$f_{\text{wood}2\text{leaf}}$	$f_{\text{root}2\text{leaf}}$	0	$-(f_{\text{root}2\text{leaf}} + f_{\text{wood}2\text{leaf}} + f_{\text{leaf}2\text{root}}) \tilde{C}$	0	0
$f_{\text{leaf}2\text{root}}$	$f_{\text{root}2\text{leaf}}$	$f_{\text{root}2\text{leaf}}$	0	$-(f_{\text{root}2\text{leaf}} + f_{\text{wood}2\text{leaf}} + f_{\text{leaf}2\text{root}}) \tilde{C}$	0	0
0	0	0	$f_{\text{leaf}2\text{root}}$	$f_{\text{root}2\text{leaf}}$	$-(f_{\text{root}2\text{leaf}} + f_{\text{wood}2\text{leaf}} + f_{\text{leaf}2\text{root}}) \tilde{C}$	$f_{\text{leaf}2\text{root}}$
0	0	0	$f_{\text{leaf}2\text{root}}$	$f_{\text{root}2\text{leaf}}$	$-(f_{\text{root}2\text{leaf}} + f_{\text{wood}2\text{leaf}} + f_{\text{leaf}2\text{root}}) \tilde{C}$	$-(f_{\text{root}2\text{leaf}} + f_{\text{wood}2\text{leaf}} + f_{\text{leaf}2\text{root}}) \tilde{C}$

```
In [ ]: mvs.get_BibInfo
## get_CompartmentalMatrix
## get_InfluxesBySymbol
Main get_InputTuple
get_InternalFluxesBySymbol
get_OutFluxesBySymbol
get_SmoothReservoirModel
In [4]: imp
# Rg get_StateVariableTuple
with not_TimeControl
```

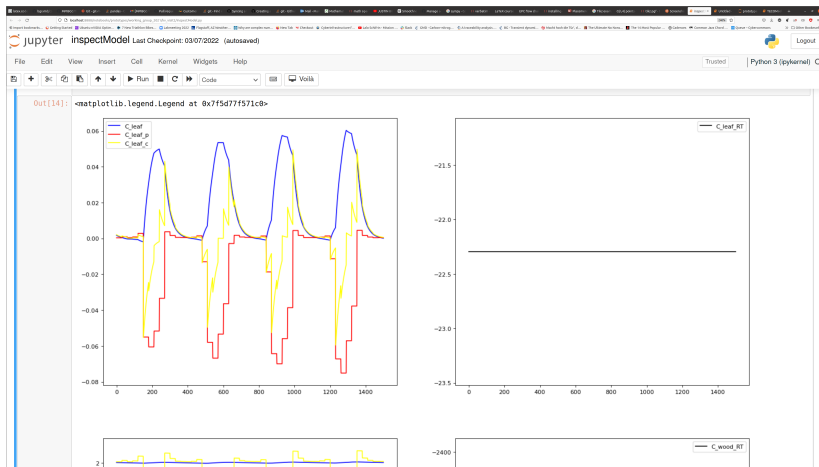
file in your model folder: <br>
name: "trendy-v9", password: "qcb-2020", dataPath: "/path/to/data/folder")

sth from config.json file
len(r) as f:

... or numerically



# Diagnostic Variables implemented once, available for all models



**Figure:** pool content + Traceability Analysis: carbon storage potential , carbon storage capacity and residence time

# Userinterface using computability graphs

The screenshot shows a Jupyter Notebook interface with the title "inspectModel". The interface includes a "File" menu, a "Not Trusted" status bar, and a "Python 3 (ipykernel)" environment. A dropdown menu is open, displaying a list of suggested methods: `computable_mvar_types`, `computers`, `get_BibInfo`, `get_CompartmentalMatrix`, `get_InFluxesBySymbol`, `get_InputTuple`, `get_InternalFluxesBySymbol`, `get_OutFluxesBySymbol`, `get_SmoothReservoirModel`, and `get_StateVariableTuple`. The notebook contains two input cells: `In [ ]: mvs.` and `In [3]: # we can also plot a picture  
h.compartmental_graph(mvs)`. The output of the second cell is `Out[3]:`, which includes a legend for the graph types: linear (blue), nonlinear (green), no state dependence (red), and undetermined (grey). Below the legend is a graph diagram with two nodes, `Croot` and `Cleafitter`, connected by arrows.

Figure: Suggested methods automatically created by a graph library

# Finding what's missing

given a set of

functions:

$a(i)$ ,  $b(c,d)$ ,  $b(e,f)$ ,

$c(b)$ ,  $d(b)$ ,  $d(g,h)$ ,

$e(b)$ ,  $f(b)$  and the

target variable **B**

e.g.

CompartmentalMatrix,

The algorithm

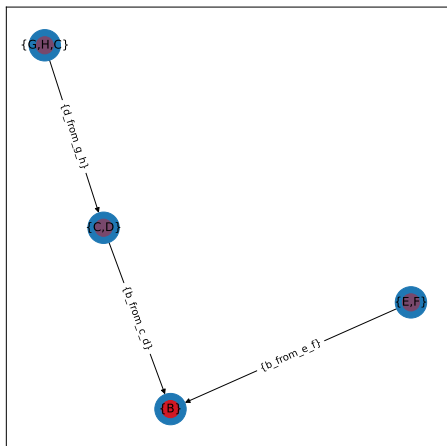
computes all

possible

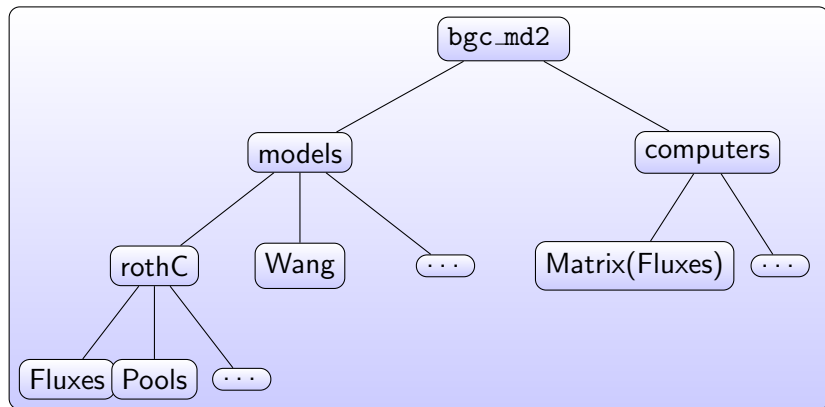
combinations and

paths from which **B**

can be computed.



# Internal Structure of bgc\_md2

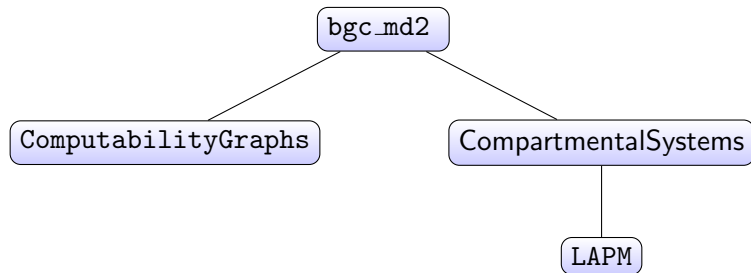




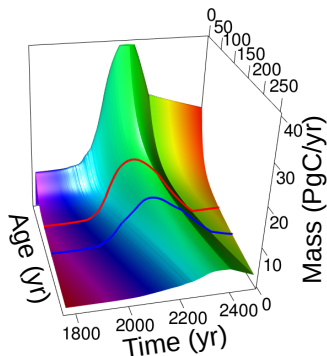
# Database records are python modules

```
1 from sympy import Symbol, Function
2 from ComputabilityGraphs.OHVS import OHVS
3 from bgc_md2.helper import models, computers
4 from bgc_md2.models.BibInfo import BibInfo
5 from bgc_md2.resolve_mvars import (
6     InFluxesBySymbol,
7     OutFluxesBySymbol,
8     InternalFluxesBySymbol,
9     TimeSymbol,
10     StateVariableTuple,
11 )
12 import bgc_md2.resolve_computers as bgc_c
13
14 # Make a small dictionary for the variables we will use
15 sym_dict={}
16 r_vl_2_wl: "Internal flux rate from leaf to wood",
17 r_wl_2_vl: "Internal flux rate from wood to leaf",
18 C_soil_fast: "",
19 C_soil_slow: "",
20 C_soil_passive: "",
21 C_leaf: "",
22 C_root: "",
23 C_wood: "",
24 C_leaf_litter: "",
25 C_root_litter: "",
26 C_wood_litter: "",
27 r_C_leaf_2_C_wood_litter: "",
28 r_C_root_2_C_wood_litter: "",
29 r_C_wood_2_C_wood_litter: "",
30 r_C_leaf_litter_rh: "",
31 r_C_root_litter_rh: "",
32 r_C_wood_litter_rh: "",
33 r_C_soil_fast_rh: "",
34 r_C_soil_slow_rh: "",
35 r_C_soil_passive_rh: "",
36 r_C_leaf_litter_2_C_soil_fast: "",
37 r_C_leaf_litter_2_C_soil_slow: "",
38 r_C_leaf_litter_2_C_soil_passive: "",
39 r_C_wood_litter_2_C_soil_fast: "",
40 r_C_wood_litter_2_C_soil_slow: "",
41 r_C_wood_litter_2_C_soil_passive: "",
42 r_C_root_litter_2_C_soil_fast: "",
43 r_C_root_litter_2_C_soil_slow: "",
44 r_C_root_litter_2_C_soil_passive: "",
45 tau: "air temperature",
46 rho: "air",
47 T_B: "",
48 E: "",
49 RH: "",
50 beta_leaf: "",
51 beta_wood: "",
52
53 for k in sym_dict.keys():
54     code=k + Symbol('()').format(k)
55     exec(code)
56
57 # some we will also use some symbols for functions (which appear with an argument)
58 func_dict={}
59 x1: "a scalar function of temperature and moisture and thereby ultimately of time",
60 NPP: "",
61
62 for k in func_dict.keys():
63     code=k + Function('()').format(k)
64     exec(code)
65
66 t=TimeSymbol('t')
67 beta_root = 1.0. (beta_leaf+beta_wood)
68 mvs = OHVS(
69
70     StateVariableTuple((
71         C_wood,
72         C_leaf,
73         C_root,
74         C_leaf_litter,
75         C_wood_litter,
76         C_root_litter,
77         C_soil_fast,
78         C_soil_slow,
79         C_soil_passive,
80     )),
81     InFluxesBySymbol(
82         {
83             C_leaf: NPP(t) * beta_leaf,
84             C_root: NPP(t) * beta_root,
85             C_wood: NPP(t) * beta_wood
86         }
87     ),
88     OutFluxesBySymbol(
89         {
90             C_leaf_litter: r_C_leaf_litter_rh*C_leaf_litter*x1(t),
91             C_wood_litter: r_C_wood_litter_rh*C_wood_litter*x1(t),
92             C_root_litter: r_C_root_litter_rh*C_root_litter*x1(t),
93             C_soil_fast: r_C_soil_fast_rh*C_soil_fast*x1(t),
94             C_soil_slow: r_C_soil_slow_rh*C_soil_slow*x1(t),
95             C_soil_passive: r_C_soil_passive_rh*C_soil_passive*x1(t),
96         }
97     ),
98     InternalFluxesBySymbol(
99         {
100             (C_leaf, C_leaf_litter): r_C_leaf_2_C_wood_litter*C_leaf,
101             (C_wood, C_wood_litter): r_C_wood_2_C_wood_litter*C_wood,
102             (C_root, C_root_litter): r_C_root_2_C_root_litter*C_root,
103             (C_leaf_litter, C_soil_fast) : r_C_leaf_litter_2_C_soil_fast * C_leaf_litter*x1(t),
104             (C_leaf_litter, C_soil_slow) : r_C_leaf_litter_2_C_soil_slow * C_leaf_litter*x1(t),
105             (C_leaf_litter, C_soil_passive) : r_C_leaf_litter_2_C_soil_passive * C_leaf_litter*x1(t),
106             (C_wood_litter, C_soil_fast) : r_C_wood_litter_2_C_soil_fast * C_wood_litter*x1(t),
107             (C_wood_litter, C_soil_slow) : r_C_wood_litter_2_C_soil_slow * C_wood_litter*x1(t),
108             (C_wood_litter, C_soil_passive) : r_C_wood_litter_2_C_soil_passive * C_wood_litter*x1(t),
109             (C_root_litter, C_soil_fast) : r_C_root_litter_2_C_soil_fast * C_root_litter*x1(t),
110             (C_root_litter, C_soil_slow) : r_C_root_litter_2_C_soil_slow * C_root_litter*x1(t),
111             (C_root_litter, C_soil_passive) : r_C_root_litter_2_C_soil_passive * C_root_litter*x1(t),
112         }
113     )
114 ),
115     BibInfo(Bibliographical Information
116         name="VistIt",
117         longName=" ",
118         version="1",
119         entryAuthor="Konstantyn Viatkin",
120         entryAuthorOrCids=" ",
121         entryCreationDate=" ",
122         doi=" "
123     )
124 )
```

## Relation to other Python Packages



# Applications



**Figure:** age distribution of a pool as function of time

Metzler, H., Müller, M., and Sierra, C. (2018). Transit-time and age distributions for nonlinear time-dependent compartmental systems. *Proceedings of the National Academy of Sciences*, 115:201705296.

# Summary: Give me what you have and I'll show you what I can do with it

`bgc_md` is a library providing:

1. Datatypes defining building blocks of models e.g. `CompartmentalMatrix`, `InternalFluxesBySymbol`, ...
2. Functions operating on those properties (forming the edges of the graph where the Datatypes are nodes)
3. A user interface based on graph algorithms to
  - 3.1 compute the set of computable properties (e.g. the comparable criteria for a set of models, database queries )
  - 3.2 actually compute the desired properties by recursively connecting several function applications.
  - 3.3 show what is missing to compute a desired property.
4. 30+ vegetation, soil or ecosystem models for carbon and nitrogen cycling as reusable python modules using the building blocks in a flexible way.
5. An interface to *many algorithms* in `CompartmentalSystems` to compute diagnostic variables for *many models* in `bgc_md2`.

# Links

- ▶ The README of the package on github (with installation instructions): [https://github.com/MPIBGC-TEE/bgc\\_md2](https://github.com/MPIBGC-TEE/bgc_md2)
- ▶ Work in progress using and extending the package:  
[https://github.com/MPIBGC-TEE/bgc\\_md2/tree/master/prototypes/working\\_group\\_2021](https://github.com/MPIBGC-TEE/bgc_md2/tree/master/prototypes/working_group_2021)
- ▶ An incomplete tutorial (jupyter notebook) for the creation of a new model. The package has to be installed.  
[https://github.com/MPIBGC-TEE/bgc\\_md2/blob/master/prototypes/working\\_group\\_2021/kv\\_visit2/createModel.py](https://github.com/MPIBGC-TEE/bgc_md2/blob/master/prototypes/working_group_2021/kv_visit2/createModel.py)