

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 expand/collapse buttons and a detailed view of the 'collapse' model.

The models listed are:

- cable_general
- TECOMm
- TECO
- Lux2012TE
- collapse

The 'collapse' model details include a legend for interaction types (linear, nonlinear, no state dependence, undetermined) and a network diagram showing interactions between carbon pools: C_{wood} , $C_{foliage}$, $C_{passsom}$, $C_{slowsom}$, C_{roots} , C_{metlit} , and C_{stilt} .

The mathematical representation of the collapse model is given by the following equations:

$$\begin{aligned} \frac{27G_{B_{top}}(1-e^{-t})}{6254} (C_{top}-C_{top}^0) \\ \frac{27G_{B_{top}}(1-e^{-t})}{6254} (C_{top}-C_{top}^0) \\ \frac{27G_{B_{top}}(1-e^{-t})}{6254} (C_{top}-C_{top}^0) \\ 0 \\ 0 \end{aligned}$$

Analysis with symbolic tools ...

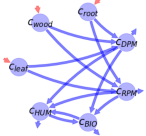
inspectModel Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

Out[3]:

linear no state dependence undetermined
nonlinear



```
input tuple, compartmental matrix)
(compartmental matrix, input tuple, internal fluxes by symbol, smooth reservoir model, out fluxes by symbol)
(compartmental matrix, input tuple, internal fluxes by symbol, smooth reservoir model, out fluxes by symbol)
```

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

$-f_{\text{root}2\text{DPM}} - f_{\text{root}2\text{RPM}}$	0	0	0	0	0
0	$-f_{\text{wood}2\text{DPM}} - f_{\text{wood}2\text{RPM}}$	0	0	0	0
0	0	$-f_{\text{root}2\text{DPM}} - f_{\text{root}2\text{RPM}}$	0	0	0
$f_{\text{leaf}2\text{DPM}}$	$f_{\text{wood}2\text{DPM}}$	$f_{\text{root}2\text{DPM}}$	$-(f_{\text{DPM}2\text{BIO}} + f_{\text{DPM}2\text{CHUM}} + f_{\text{DPM}2\text{CBIO}}) \tilde{C}$	0	0
$f_{\text{leaf}2\text{RPM}}$	$f_{\text{wood}2\text{RPM}}$	$f_{\text{root}2\text{RPM}}$	0	$-(f_{\text{DPM}2\text{BIO}} + f_{\text{DPM}2\text{CHUM}} + f_{\text{DPM}2\text{CBIO}}) \tilde{C}$	0
0	0	0	$f_{\text{DPM}2\text{BIO}} \tilde{C}$	$f_{\text{DPM}2\text{BIO}} \tilde{C}$	$-(f_{\text{BIO}2\text{BIO}} + f_{\text{BIO}2\text{CBIO}}) \tilde{C}$
0	0	0	$f_{\text{DPM}2\text{CHUM}} \tilde{C}$	$f_{\text{DPM}2\text{CHUM}} \tilde{C}$	$-(f_{\text{CHUM}2\text{BIO}} + f_{\text{CHUM}2\text{CBIO}}) \tilde{C}$

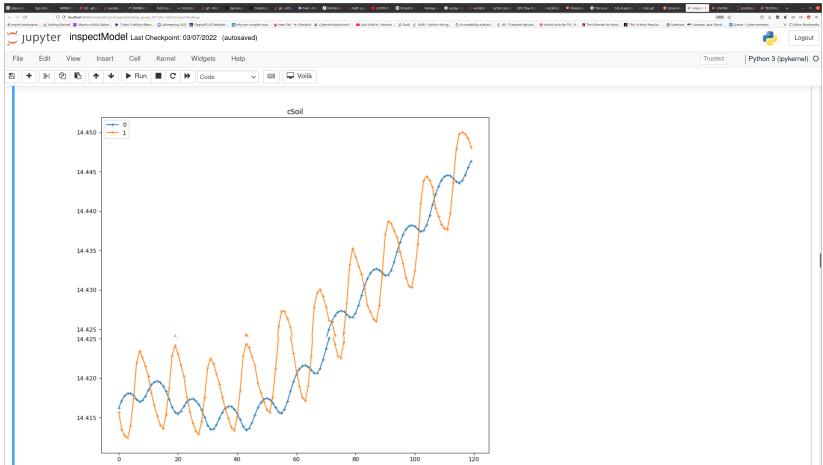
```
In [1]: mvs.get_bibinfo()
## get_compartmental_matrix
## get_internal_fluxes_by_symbol
## get_input_tuple
## get_internal_fluxes_by_symbol
## get_out_fluxes_by_symbol
## get_smooth_reservoir_model
## get_state_variable_tuple
with not_time_control
```

file in your model folder:

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

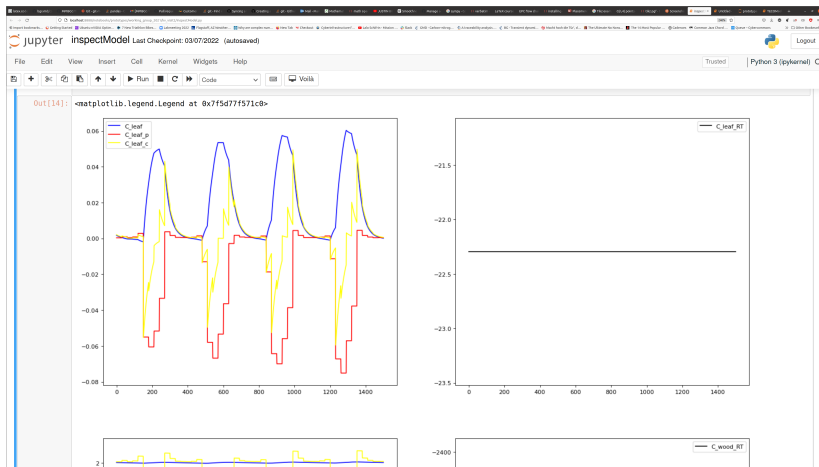


Figure: pool content + Traceability Analysis: carbon storage potential , carbon storage capacity and residence time

Userinterface using computability graphs

The screenshot shows a Jupyter Notebook interface with the title "inspectModel". The interface includes a file explorer, a code editor, and an output area. 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 code editor shows the following input:

```
In [ ]: mvs.
```

The output area displays the following text:

```
In [3]: # we can also plot a picture
h.compartmental_graph(mvs)
```

Out[3]:

A legend indicates the types of nodes in the graph:

- linear (blue square)
- nonlinear (green square)
- no state dependence (red square)
- undetermined (grey square)

The graph itself shows two nodes: `Croot` (blue circle) and `Cleafitter` (blue circle). Arrows indicate connections between the nodes and their inputs/outputs.

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

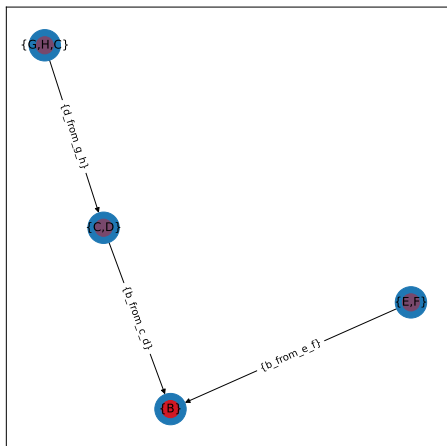
computes all

possible

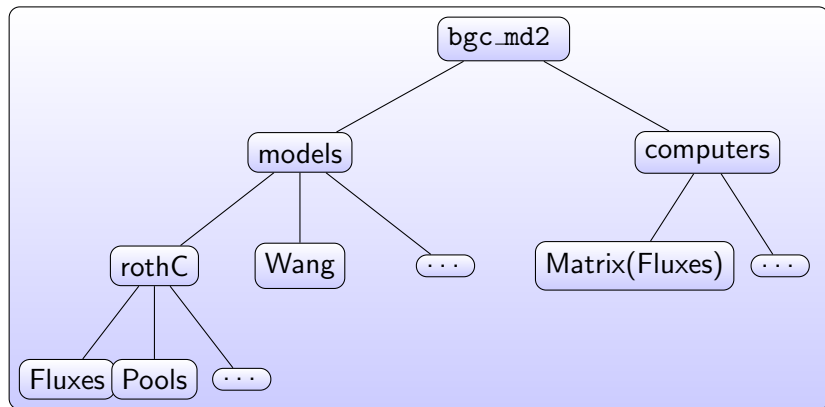
combinations and

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 theta: "soil moisture",
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
116 BibInfo(Bibliographical Information
117     name="VistIt",
118     longName=" ",
119     version="1",
120     entryAuthor="Konstantyn Viatkin",
121     entryAuthorOrCids=" ",
122     entryCreationDate=" ",
123     doi=" ")

```

~/bgc_md2/prototypes/working_group_2021/kv_vistIt/source.py

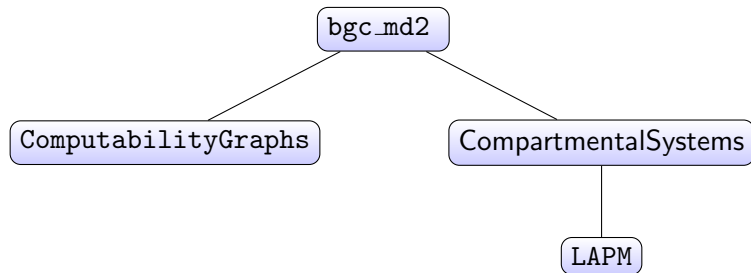
20,25

Top ~/bgc_md2/prototypes/working_group_2021/kv_vistIt/source.py

75,10-23

87%

Relation to other Python Packages



Applications

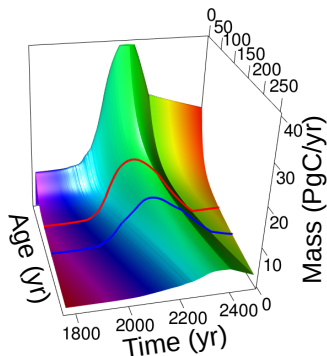


Figure: age distribution of a pool as function of time

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
- ▶ Work in progress using and extending the package:
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