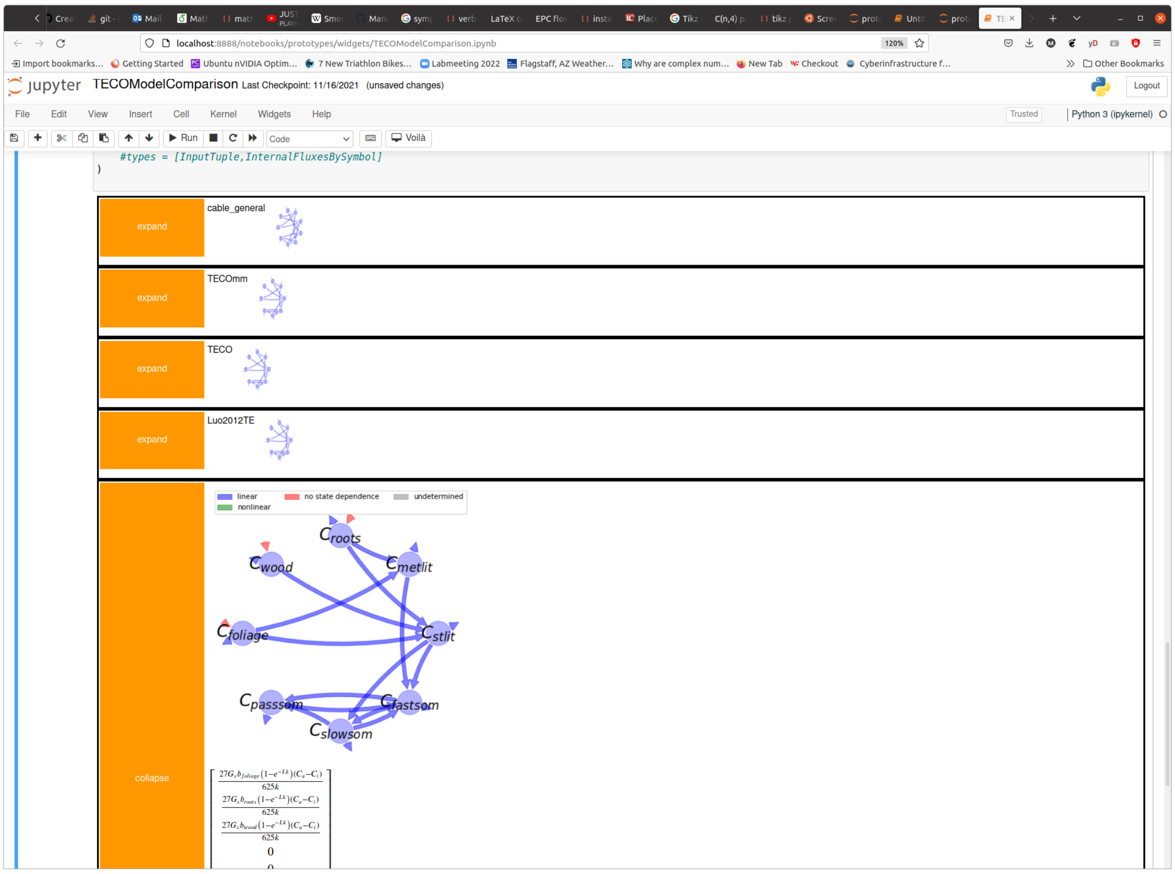
The Biogeochemical Model Database bgc_md2


localhost:8888/notebooks/prototypes/widgets/TECOModelComparison.ipynb
jupyter TECOModelComparison Last Checkpoint: 11/16/2021 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Voilà
expand
expand
expand
expand
collapse
linear
no state dependence
undetermined
nonlinear

```
    #types = [InputTuple,InternalFluxesBySymbol]
)
```

cable_general

TECOmm

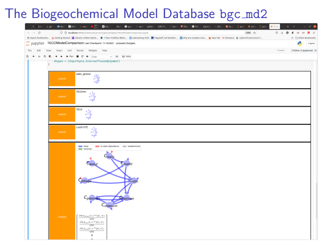
TECO

Luo2012TE

$$\begin{bmatrix} \frac{27 G_s b_{foliage} \left(1 - e^{-Lk}\right) (C_a - C_i)}{625k} \\ \frac{27 G_s b_{roots} \left(1 - e^{-Lk}\right) (C_a - C_i)}{625k} \\ \frac{27 G_s b_{wood} \left(1 - e^{-Lk}\right) (C_a - C_i)}{625k} \\ 0 \\ 0 \end{bmatrix}$$

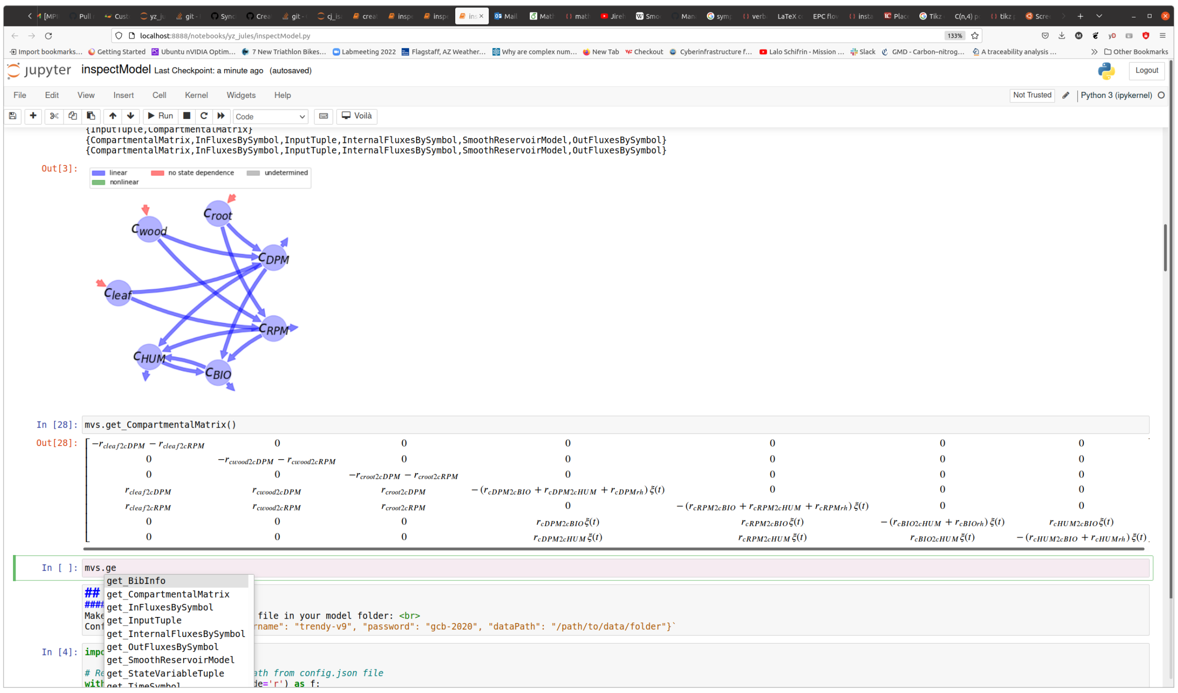
The Biogeochemical Model Database

└The Biogeochemical Model Database bgc_md2



1. bgc_md2 is an open source python package availabe on GitHub, developed at the Max-Planck-Institut for BioGeoChemistry in Jena and more recently in Yiqi Luo's Ecolab at NAU in Flagstaff
2. A set of libraries that can be used in other python scripts or interactively (jupyter or IPython) The picture shows a jupiter widget showing a table of models. The orange buttons can be clicked to expand or collape a more detailed view of the particular model.
3. $>$ 30 published vegetation, soil or ecosystems models in a format that for symbolic and numeric computations
4. A set of special datatypes that describe components of the models and functions that operate on these datatypes
5. A userinterface that uses a graph library to compute what is computable and can be used for comparisons.

Analysis with symbolic tools . . .



```
{InputTuple,CompartmentalMatrix}
{CompartmentalMatrix,InFluxesBySymbol,InputTuple,InternalFluxesBySymbol,SmoothReservoirModel,OutFluxesBySymbol}
{CompartmentalMatrix,InFluxesBySymbol,InputTuple,InternalFluxesBySymbol,SmoothReservoirModel,OutFluxesBySymbol}
Out[3]:
```

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

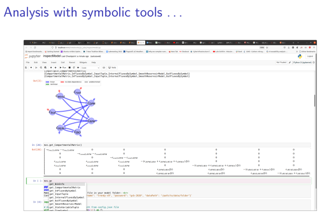
$$\begin{bmatrix} -r_{cleaf2cDPM} - r_{cleaf2cRPM} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -r_{cwood2cDPM} - r_{cwood2cRPM} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -r_{croot2cDPM} - r_{croot2cRPM} & 0 & 0 & 0 & 0 \\ r_{cleaf2cDPM} & r_{cwood2cDPM} & r_{croot2cDPM} & -(r_{cDPM2cBIO} + r_{cDPM2cHUM} + r_{cDPMrh})\,\xi(t) & 0 & 0 & 0 \\ r_{cleaf2cRPM} & r_{cwood2cRPM} & r_{croot2cRPM} & 0 & -(r_{cRPM2cBIO} + r_{cRPM2cHUM} + r_{cRPMrh})\,\xi(t) & 0 & 0 \\ 0 & 0 & 0 & r_{cDPM2cBIO}\,\xi(t) & r_{cRPM2cBIO}\,\xi(t) & -(r_{cBIO2cHUM} + r_{cBIOrh})\,\xi(t) & r_{cHUM2cBIO}\,\xi(t) \\ 0 & 0 & 0 & r_{cDPM2cHUM}\,\xi(t) & r_{cRPM2cHUM}\,\xi(t) & r_{cBIO2cHUM}\,\xi(t) & -(r_{cHUM2cBIO} + r_{cHUMrh})\,\xi(t) \end{bmatrix}$$

```
In [ ]: mvs.ge
         get_BibInfo
         get_CompartmentalMatrix
         get_InFluxesBySymbol
         get_InputTuple
         get_InternalFluxesBySymbol
         get_OutFluxesBySymbol
         get_SmoothReservoirModel
         get_StateVariableTuple
         get_TimeSymbol

         ##
         ####
         Make                 file in your model folder: <br>
         Conf                rname": "trendy-v9", "password": "gcb-2020", "dataPath": "/path/to/data/folder"}`

In [4]:  impo
         # Re                 ath from config.json file
         with                 de='r') as f:
```

2022-03-20

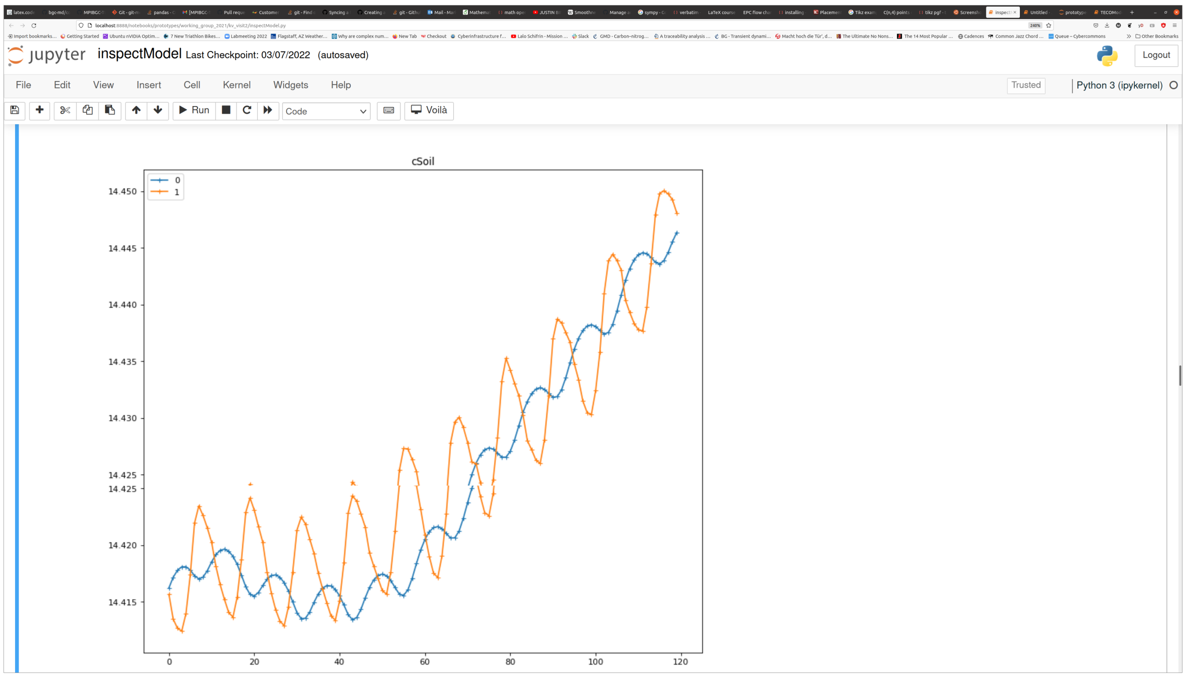
The Biogeochemical Model Database

└Analysis with symbolic tools . . .


Analysis with symbolic tools . . .

1. the structure (graph both in the mathematical and visual sense) can be derived from the symbolic description
2. other properties are flux equations the compartmental matrix

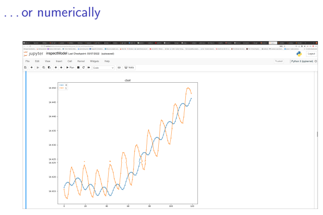
. . . or numerically



2022-03-20

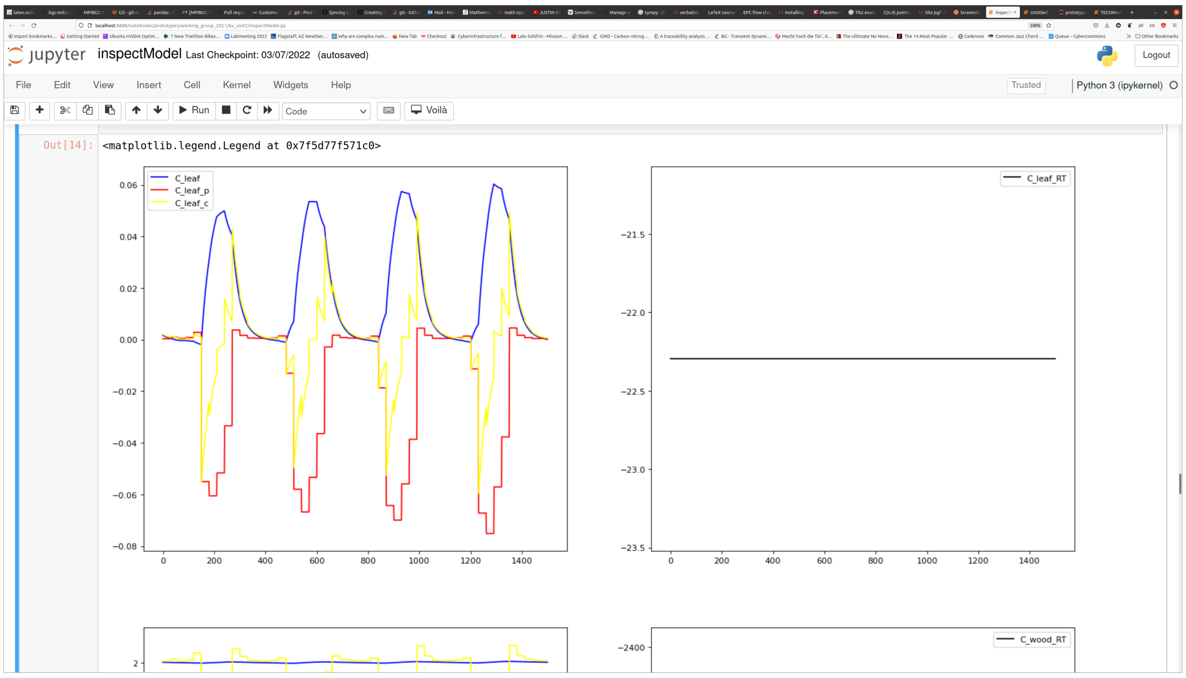
The Biogeochemical Model Database

└. . . or numerically



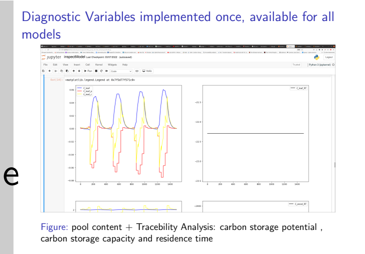
1. the symbolic model description can be parameterized and transformed into a numeric model
2. The picture shows the data assimilation result for the above model using trendy data.

Diagnostic Variables implemented once, available for all models



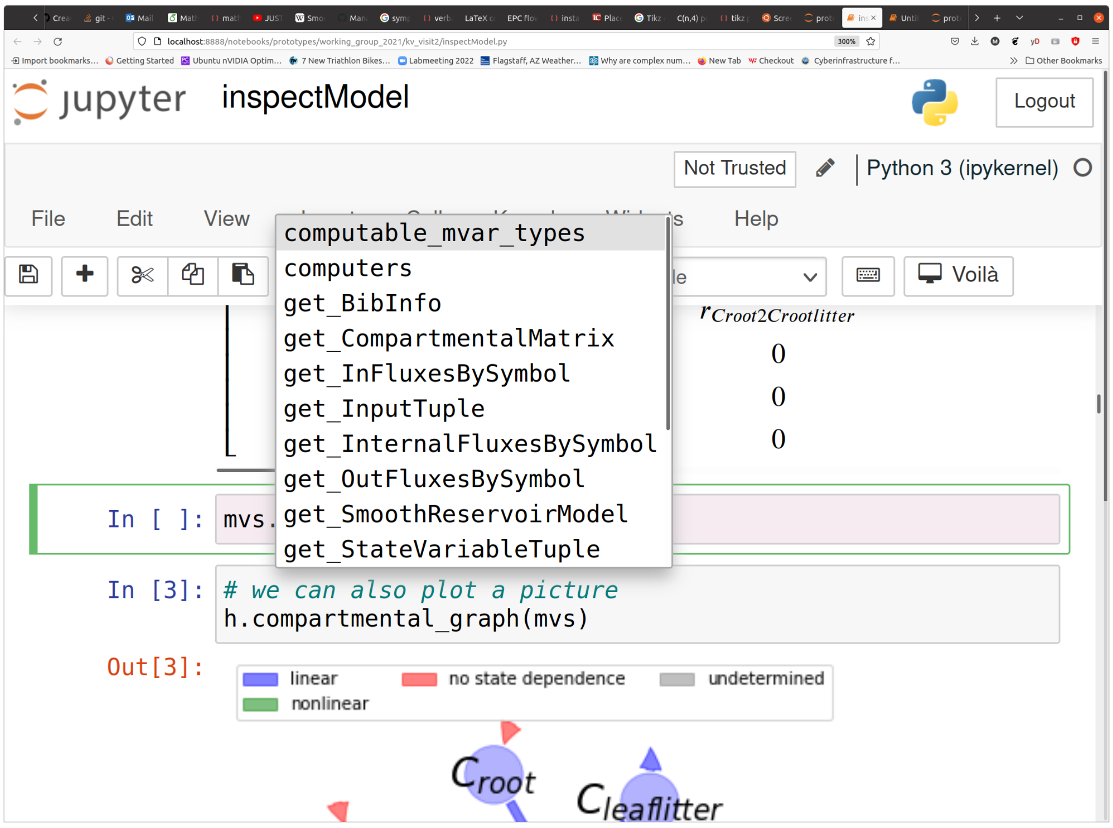
```
Out[14]: <matplotlib.legend.Legend at 0x7f5d77f571c0>
```

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



1. Diagnostic Variables can be computed for any model
2. The picture shows the Leaf pool

Userinterface using computability graphs



```
In [ ]: mvs.

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

Out[3]:
```

$r_{Croot2Crootlitter}$

0
0
0

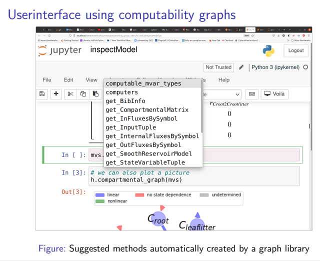
C_{root} $C_{leaflitter}$

Figure: Suggested methods automatically created by a graph library

2022-03-20

The Biogeochemical Model Database

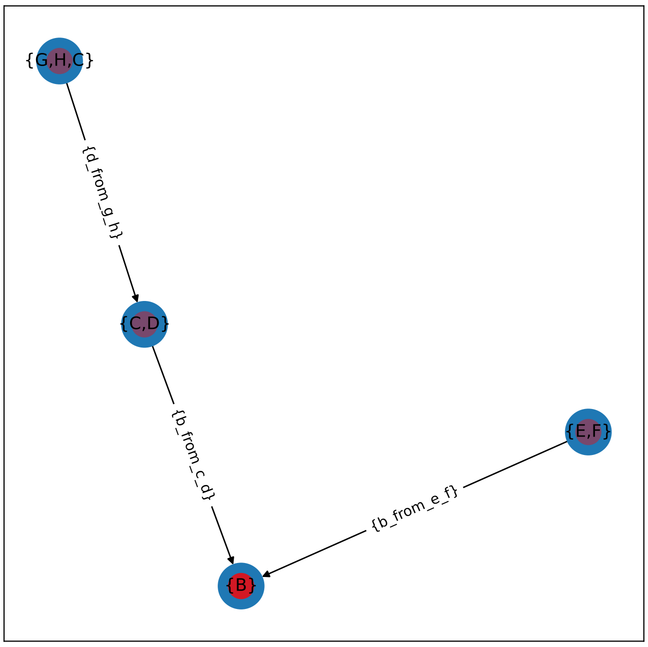
└Userinterface using computability graphs



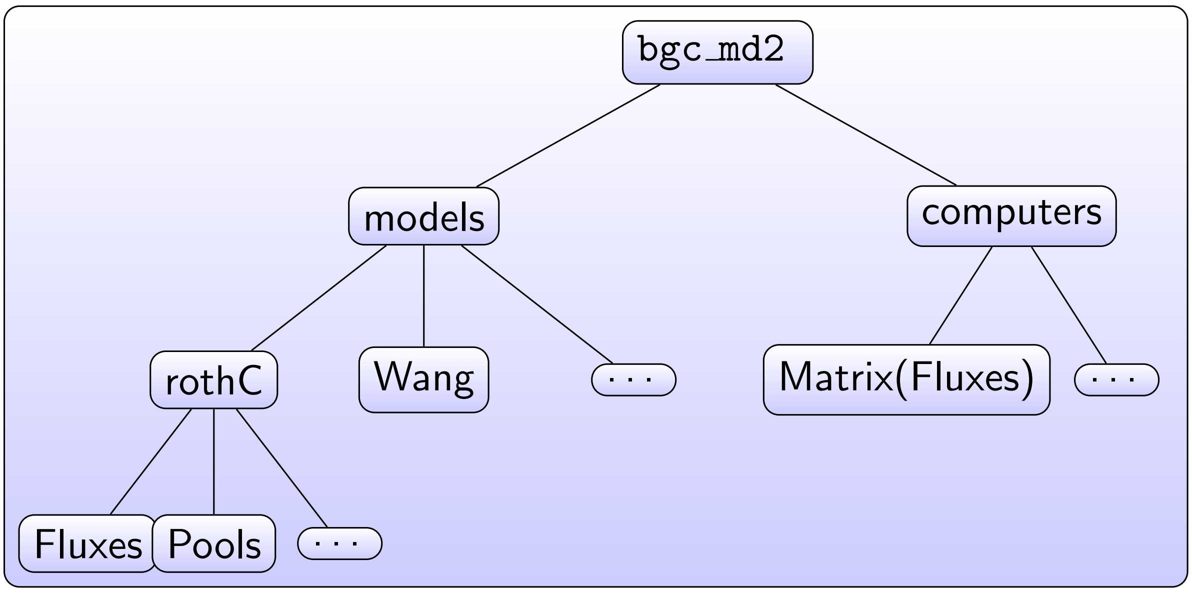
1. This is standard for mehtods of a python object. In this case the methods are automatically created and added by a graphalgorithm that computes which variables can be computed form the set of provided model properties. This has far reaching consequences. Models can be compared with respect to all variables in the "convex hull under computability', with respect to all *computable variables* not just the ones provided in the data base record.
2. As an example the matrix formulation of models A and B can be compared even if neither A nor B defines the matrix as long as it can be computed from other variable (in this case the internal and outfluxes and an ordering of the statevariables)
3. An obvious consequence is that the information about a model can be provided in different ways. There is no need to force the user into a rigid record format. Another consequence is that records do not have to be complete. The framework accepts all information about a model and (computes what it can do with it).

Finding what's missing

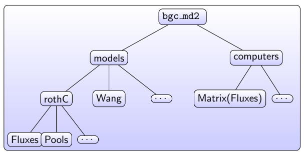
given a set of functions:
a(i), b(c,d), b(e,f), c(b), d(b), d(g,h), e(b), f(b) and the target variable B (e.g `CompartmentalMatrix`)
The algorithm computes all possible combinations and paths from which B can be computed.



Internal Structure of `bgc_md2`

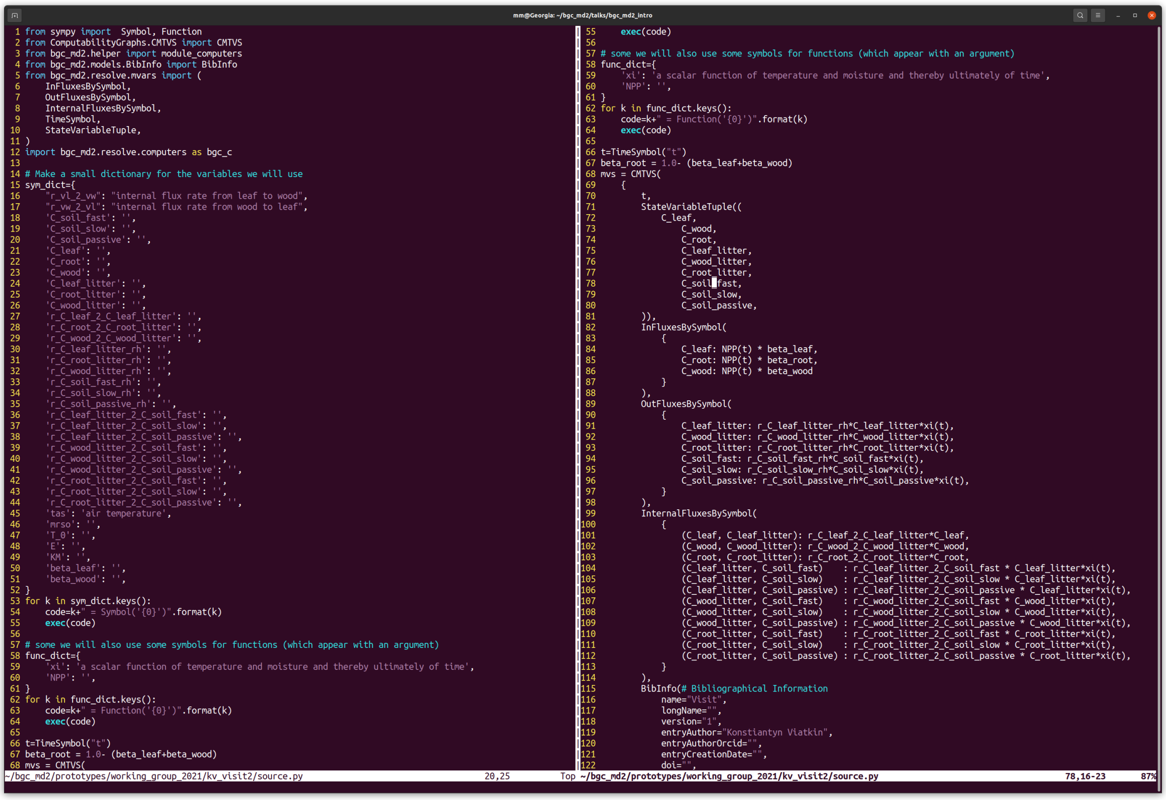


Internal Structure of bgc_md2



1. bgc_md2 is not just a collection of models, described as sets of varible of special type like fluxes or matrice, but also a collection of functions whose arguments and return values have these types. These functions are here called computers and use python type annotations. The computability graph used in the user interface and queries is derived from the annotations of a set of functions. The set of properties (defined by the types) is growing as well as the functions connecting them.

Database records are python modules



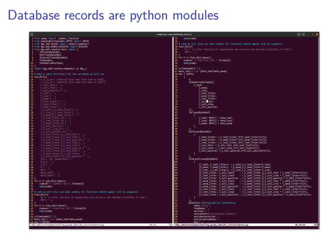
```python
from sympy import  Symbol, Function
from ComputabilityGraphs.CMTVS import CMTVS
from bgc_md2.helper import module_computers
from bgc_md2.models.BibInfo import BibInfo
from bgc_md2.resolve.mvars import (
    InFluxesBySymbol,
    OutFluxesBySymbol,
    InternalFluxesBySymbol,
    TimeSymbol,
    StateVariableTuple,
)
import bgc_md2.resolve.computers as bgc_c

# Make a small dictionary for the variables we will use
sym_dict={
    "r_vl_2_vw": "internal flux rate from leaf to wood",
    "r_vw_2_vl": "internal flux rate from wood to leaf",
    'C_soil_fast': '',
    'C_soil_slow': '',
    'C_soil_passive': '',
    'C_leaf': '',
    'C_root': '',
    'C_wood': '',
    'C_leaf_litter': '',
    'C_root_litter': '',
    'C_wood_litter': '',
    'r_C_leaf_2_C_leaf_litter': '',
    'r_C_root_2_C_root_litter': '',
    'r_C_wood_2_C_wood_litter': '',
    'r_C_leaf_litter_rh': '',
    'r_C_root_litter_rh': '',
    'r_C_wood_litter_rh': '',
    'r_C_soil_fast_rh': '',
    'r_C_soil_slow_rh': '',
    'r_C_soil_passive_rh': '',
    'r_C_leaf_litter_2_C_soil_fast': '',
    'r_C_leaf_litter_2_C_soil_slow': '',
    'r_C_leaf_litter_2_C_soil_passive': '',
    'r_C_wood_litter_2_C_soil_fast': '',
    'r_C_wood_litter_2_C_soil_slow': '',
    'r_C_wood_litter_2_C_soil_passive': '',
    'r_C_root_litter_2_C_soil_fast': '',
    'r_C_root_litter_2_C_soil_slow': '',
    'r_C_root_litter_2_C_soil_passive': '',
    'tas': 'air temperature',
    'mrso': '',
    'T_0': '',
    'E': '',
    'KM': '',
    'beta_leaf': '',
    'beta_wood': '',
}
for k in sym_dict.keys():
    code=k+" = Symbol('{0}')".format(k)
    exec(code)

# some we will also use some symbols for functions (which appear with an argument)
func_dict={
    'xi': 'a scalar function of temperature and moisture and thereby ultimately of time',
    'NPP': '',
}
for k in func_dict.keys():
    code=k+" = Function('{0}')".format(k)
    exec(code)

t=TimeSymbol("t")
beta_root = 1.0- (beta_leaf+beta_wood)
mvs = CMTVS(
```

```
~/bgc_md2/prototypes/working_group_2021/kv_visit2/source.py                20,25          Top
```

```python
    exec(code)

# some we will also use some symbols for functions (which appear with an argument)
func_dict={
    'xi': 'a scalar function of temperature and moisture and thereby ultimately of time',
    'NPP': '',
}
for k in func_dict.keys():
    code=k+" = Function('{0}')".format(k)
    exec(code)

t=TimeSymbol("t")
beta_root = 1.0- (beta_leaf+beta_wood)
mvs = CMTVS(
```

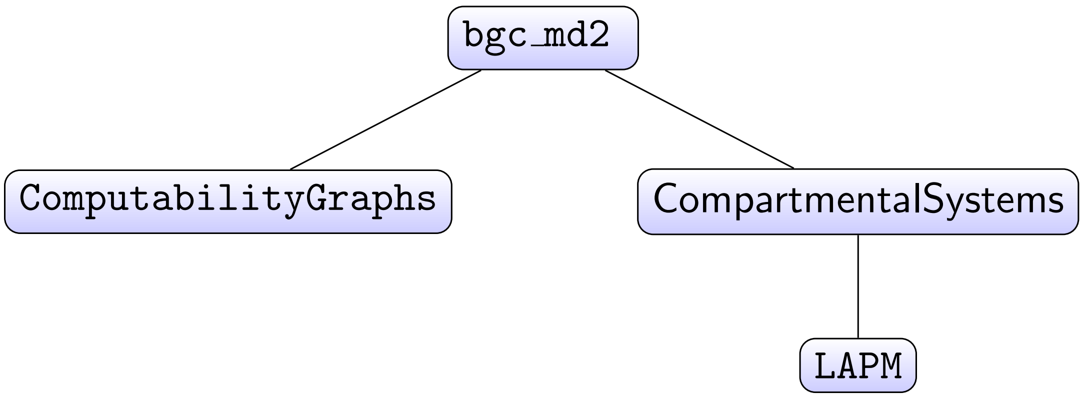
```python
    {
        t,
        StateVariableTuple((
            C_leaf,
                C_wood,
                C_root,
                C_leaf_litter,
                C_wood_litter,
                C_root_litter,
                C_soil_fast,
                C_soil_slow,
                C_soil_passive,
        )),
        InFluxesBySymbol(
            {
                C_leaf: NPP(t) * beta_leaf,
                C_root: NPP(t) * beta_root,
                C_wood: NPP(t) * beta_wood
            }
        ),
        OutFluxesBySymbol(
            {
                C_leaf_litter: r_C_leaf_litter_rh*C_leaf_litter*xi(t),
                C_wood_litter: r_C_wood_litter_rh*C_wood_litter*xi(t),
                C_root_litter: r_C_root_litter_rh*C_root_litter*xi(t),
                C_soil_fast: r_C_soil_fast_rh*C_soil_fast*xi(t),
                C_soil_slow: r_C_soil_slow_rh*C_soil_slow*xi(t),
                C_soil_passive: r_C_soil_passive_rh*C_soil_passive*xi(t),
            }
        ),
        InternalFluxesBySymbol(
            {
                (C_leaf, C_leaf_litter): r_C_leaf_2_C_leaf_litter*C_leaf,
                (C_wood, C_wood_litter): r_C_wood_2_C_wood_litter*C_wood,
                (C_root, C_root_litter): r_C_root_2_C_root_litter*C_root,
                (C_leaf_litter, C_soil_fast)    : r_C_leaf_litter_2_C_soil_fast * C_leaf_litter*xi(t),
                (C_leaf_litter, C_soil_slow)    : r_C_leaf_litter_2_C_soil_slow * C_leaf_litter*xi(t),
                (C_leaf_litter, C_soil_passive) : r_C_leaf_litter_2_C_soil_passive * C_leaf_litter*xi(t),
                (C_wood_litter, C_soil_fast)    : r_C_wood_litter_2_C_soil_fast * C_wood_litter*xi(t),
                (C_wood_litter, C_soil_slow)    : r_C_wood_litter_2_C_soil_slow * C_wood_litter*xi(t),
                (C_wood_litter, C_soil_passive) : r_C_wood_litter_2_C_soil_passive * C_wood_litter*xi(t),
                (C_root_litter, C_soil_fast)    : r_C_root_litter_2_C_soil_fast * C_root_litter*xi(t),
                (C_root_litter, C_soil_slow)    : r_C_root_litter_2_C_soil_slow * C_root_litter*xi(t),
                (C_root_litter, C_soil_passive) : r_C_root_litter_2_C_soil_passive * C_root_litter*xi(t),
            }
        ),
        BibInfo(# Bibliographical Information
            name="Visit",
            longName="",
            version="1",
            entryAuthor="Konstiantyn Viatkin",
            entryAuthorOrcid="",
            entryCreationDate="",
            doi="",
```

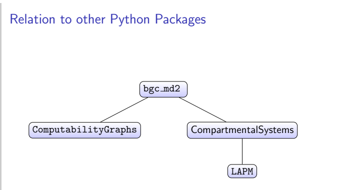
```
~/bgc_md2/prototypes/working_group_2021/kv_visit2/source.py                78,16-23       87%
```


Database records are python modules

1. The picture shows the screen shot of the source code of the above model using sympy and some datatypes provided by `bgc_md2` .
2. The entries of the database dont even have to be complete models. They are implemented in normal python and have in common that they define a set of model properties and a set of functions to connect them. There is no special format necessary. The creation of the symbolic formulation can be automated bye all means available in python. Extra information can but does not have to be provided.

Relation to other Python Packages




Relation to other Python Packages

1. The graph computation is outsourced into our package `ComputabilityGraphs`
2. Many of the advanced diagnostic variables (age and transittime distributions) are computed using our other packages `LAPM` and `CompartmentalSystems` for which `bgc_md2` acts as interface.

Applications

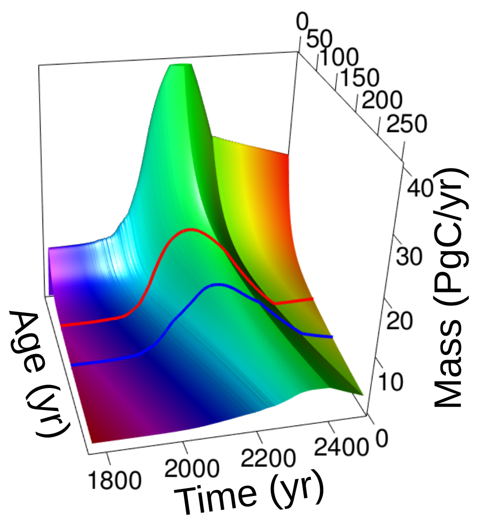


Figure: age distribuition of a pool as function of time

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.