Key Points:

- We discuss criteria for metrics with respect to which all compartmental models can be compared with focus on transit times and ages of subsystems.
- We investigate how software libraries could be designed to make the creation of new models easier faster and more flexible, their inspection richer, more colorful and transparent and turn a model collection into a queriable database.
- We implement these libraries, provide a model intercomparison example and discuss extended use cases.

The biogeochemichal model data base `bgc_md2` and python packages `LAPM`, `CompartmentalSystems`, `ComputabilityGraphs` for the analysis of compartmental dynamical systems

2]Markus Müller 4]Holger Metzler 3]Verónika Ceballos-Núñez 2]Konstiantin Viatkin 1]Carlos A. Sierra 2]Yiqi Luo [1]Max Planck Institute for Biogeochemistry, Hans-Knöll-Str. 10, 07745 Jena, Germany [2]Cornell [3] [4]

Corresponding author: Markus Müller, `markus.mueller.1.g@googlemail.com`

Abstract

Compartmental systems are a particular class of dynamical systems that describe the flow of conserved quantities such as mass and energy through a network of interconnected compartments. The main purpose and greatest challenge of this work is to make such models **transparent** by facilitating comparisons between them. The scientific challenge is to find the common diagnostics by which we can compare models. The technical challenges are, firstly to implement the diagnostics in a way that makes them applicable to all collected models, secondly how to *describe* models comparably to allow queries but flexibly enough to formulate them in the many different ways in which they appear in the literature and thirdly compactly enough to facilitate a maintainable collection of reasonable size

We enhance the common diagnostics of pool contents and fluxes by the computations of age and transit time distributions for whole models and especially subsystems like vegation or soil which is essential to compare models with conceptually different pools, i.e. two ecosystem models w.r.t the vegetation part , consisting however of two pools (i.e. leaf and root) in one but three pools (i.e. leaf, wood, root) in the other. We provide an extensible, declarative domain specific language (DSL) to enable (data base) queries and reduce the amount of model specific code by orders of magnitude. avoiding duplication in two novel ways: It defines an extensible set of building blocks common to all models and implemented in symbolic math via `sympy` as special types, and an extensible set of type annotated functions operating on these types as building blocks which can be combined automatically by a graph library to compute every result, reachable by any recursive combination of these functions. This allows us to formulate models compactly and flexibly in different equivalent ways i.e. via fluxes, flowdiagrams, or matrices and at the same time to avoid the implementation of many similar functions for the same result. Thus we can also not only query and compare what is *provided*, as in the model description records of a conventional data base, but also what is *computable* from them.

Apart from the technical aspects the rigorous description of models via a strictly typed functional DSL also informs the choice of diagnostics variables by excluding ambigously defined candidates that have been proposed in the literature. The combination of these capabilities is implemented in open source python packages `bgc_md2` (Biogeochemical Model Database), `LAPM`(Linear Autonomous Pool Models), `CompartmentalSystems`and `ComputabilityGraphs`, available on GitHub and for testing even without installation on binder. We proof the above mentioned concepts by comparing four trendy9 models with respect to transient ages and transit times.

Plain Language Summary

Imagine a bucket standing on a table under a water tap. Now drill some small holes in the bottom and attach some short pipes that lead to other buckets standing on some chairs and drill holes in their bottom to attach pipes to even more buckets standing on the ground. Drill holes again in the bottom so that water can leak out... It turns out that regardless of the number of buckets pipes or taps, the amount of water in the buckets at any time in the future can be predicted very accurately without knowing the details of how exactly the water flows through the buckets, for instance where the holes in the bottom are or which path a single water molecule takes through the bucket. Actually it is enough to know three things: how much water is in the buckets at the start, how much water comes out of the tap at any time and how the outflux through the holes depends on the amount of water in the bucket i.e. how big the holes are. The simplicity of this description is extremely attractive and has made bucket or pool models prolific in many branches of science, describing, apart from the obvious physical examples, phenomena including chemical reactors or ecological earth system models for the carbon cycle, essential for estimating future climate change. All these models are called *compart-*

mental models. We developed a set of software tools that makes it much easier to describe, translate, collect, inspect, and ultimately compare such models. While the tools are as widely applicable as compartmental models, the example applications we present are inspired by our own work in the global carbon cycle where the reduction of uncertainty of model predictions is a major goal for climate change mitigation and a strong motivation for model inter comparisons and the increased transparency facilitating them. Our tools can translate the simple description given above into a system of ordinary differential equations and vice versa visualize such a matrix equation as system of interconnected buckets. They can compute complex numerical results that can in principle be compared to measurable physical quantities. The ability to do this for a large number of models, made possible by a parsimonious description, is unprecedented and makes every single model much more transparent. Our software tools are free to use, change and extend. The aim of this article is to make them available to researchers who want to use them, show the way how they can be extended but also to discuss what more fundamental lessons we have learned in the process of their creation e.g. w.r.t which criteria compartmental models should be compared.

1 Introduction

The principle of mass conservation plays a central role in mathematical models of natural systems in a variety of scientific fields such as systems biology, toxicology, pharmacokinetics (Anderson, 1983), ecology (Eriksson, 1971; Rodhe & Björkström, 1979; Matis et al., 1979; Manzoni & Porporato, 2009), hydrology (Nash, 1957; Botter et al., 2011; Harman & Kim, 2014), biogeochemistry (Manzoni & Porporato, 2009; Sierra & Müller, 2015), and epidemiology (Jacquez & Simon, 1993). In most cases such models are non negative dynamical systems that can be described by first-order systems of ordinary differential equations (ODEs) with strong structural constraints. Such systems are called compartmental systems (Anderson, 1983; Jacquez & Simon, 1993; Walter & Contreras, 1999; Haddad et al., 2010), and have important mathematical properties that aid in their analysis and study.

1.1 Additional Diagnostics

Obvious properties of compartmental systems are the content of the pools, and the fluxes between the pools as functions of time. As mass moves across a compartmental system, it is often also of interest to study properties like the *age* of mass in a pool, the entire system or a subsystem, and the *transit time* of mass across the entire network of compartments or parts of it until its final exit. As we will see later for compartmental systems ages and transit times are characterized by distributions that are time dependent. However, methods traditionally used in the literature, to compute these metrics for compartmental systems depended on specific assumptions imposed on the system of equations (Sierra et al., 2017) and restricted either the applicability of the procedures or the interpretability of the results if applied to situations where those conditions where not fulfilled exactly.

1.2 Origin and Present State of Software Tools

This becomes a challenge when they are implemented in software, particularly if it is required to be *queried* i.e. a 'data base'. It is much easier to claim that something is computable than to implement it. It is much easier to warn a scientist to check the applicability of a method than to avoid a misapplication of that function automatically. It is much easier to present a proof of concept of one or two diagnostic variables for one or two models than to build and maintain a *queryable* collection of *many* models and *many* diagnostics.

Over the course of more than ten years and with the help of contributions of several people we have developed practically usable tools to create, inspect and query a collection of compartmental models, i.e. a model data base. We have learned some hard lessons in the process. While coding has become a major part of science, and greatly enhances our ability of prediction, it has also obscured the ways to obtain them. A way to apply the *same* tools to *many* models is a very attractive proposition to regain some of the transparency by well defined comparisons. As the number of models and the number of applicable diagnostics grow the the number of combinations grows bilinearly, and with it the technical challenges to maintain both collections. A 'copy and paste' approach becomes unmaintainable very quickly, in our case definitely before the present number of about 30+ models had been reached. Ideally we want to turn the bilinear growth of combinations of models and diagnostic tools into an advantage and make all diagnostics, and especially a new one, instantly available to all models. To make this possible the right abstractions have to be found, just to keep the size of the code base in check. At the outset it is very easy to overlook that the right abstractions are an emerging feature and *change* with the addition of more models and new methods with unforeseen properties. To be able to repeatedly refactor the code base w.r.t better abstractions requires software engineering techniques like unit testing and more specifically a 'test hull' i.e. a comprehensive set of tests to provide coverage for the features, so that their implementation can be changed with confidence. Apart from the benefit of inducing better engeneering practices the work on a model data base has the benefit of *unveiling* these abstractions. Their choice is not accidental but the result of a decade of refinement and a major outcome of the project. An example for this clarifying feedback of software development to science is the basic requirement, to be able to define a result of a computation by a function implementing this computation. Astonishingly this excludes some metrics for compartmental systems that have been proposed, as we will see.

Our first encounter with the challenge of a model collection was the development of `SoilR` (Sierra et al., 2012), a `R` package that collects a number of compartmental models specialized on soil models. The software we are going to present can be seen as the result of countless refactoring and two major design iterations of this initial project, the result of contributions of many codevelopers, user input, and a decade of critical evaluation and resulting change. The result not only far exceeds `SoilR`'s capabilities but also constitutes our best proposal for an extendable community tool so far. In particular the new approach

1. is based on symbolic math, which allows users to look at models as they appear in publications, in the form of equations and flow diagrams, and enables symbolic analysis (e.g. computing derivatives for sensitivity analysis)
2. describes compartmental models as graphs, *or* matrix equations and enables the *automatic conversions* between them, thereby enable ling the definition of subsystems by simply naming their pools.
3. adds diagnostics (transit time and age distributions) not present in `SoilR`.
4. provides a much simpler user interface to the vast number of analytical tools that had already made the creation of a navigable documentation an overwhelming job for `SoilR`.
5. simplifies the description of models by much smaller building blocks that are combined *automatically*, and combination of few or many such building blocks into a 'model record', rather than forcing the developers to implement (and a user to choose) a huge number of model constructing functions.
6. *automatically combines functions* to compute results recursively

We developed three python packages for the representation, classification, collection and analysis of compartmental systems, implementing methods for the computation of age and transit times, that can be performed for particular systems under given

assumptions, and tools to automatically *restrict* their use. These open source packages are called: `LAPM` (Linear Autonomous Pool Models), `CompartmentalSystems` and `bgc_md2` (Biogeochemical Model Database). This manuscript provides a general introduction to these packages with emphasis of their combined use via `bgc_md2`. The latter implements a set of model describing properties that form the (exclusive) arguments and return values of strictly typed functions, forming a network of computability which we use to implement a declarative domain specific language for model creation, inspection, querying and thus comparison. To facilitate this we implemented a fourth package `ComputabilityGraphs` based upon the type hinting syntax of modern `Python` versions, extending its use to predicting what is computable. The requirement to define every model property as a variable of a specific type, and to provide a function to compute it from other model property (i.e. variable of well defined *types*) also provides a rigorous filter for what constitutes a diagnostic variable or model property. We use it as such to discuss why we implemented or excluded some diagnostics proposed in the literature. Practically we use this language to create a small (30+) collection of models, motivated by our own work on the terrestrial carbon cycle. Some screenshot of the software are shown in Fig. 1.

As a conceptual proof of the approach implemented by `bgc_md2` we reconstruct four models of the trendy9 model intercomparison from their description in the literature. We express them symbolically, provide some parameters run them with some trendy9 driver data and compare them with respect to transient mean ages and transit times of carbon trough the vegetation and soil subsystems. While some of the implemented algorithms for compartmental subsystems are novel and yet unpublished, this work focuses on the algorithmic and structural proposal for a model collection i.e. a model data base to facilitate their use for model intercomparison. We provide examples based on the global carbon cycle, but the packages can be used for a large variety of systems in which mass or energy conservation is required. The remainder of the article is structured as follows. In section 2 we briefly describe compartmental systems, with some of the details relegated to appendix Appendix A. We then provide an overview over the main role of the implemented packages in section 3. A brief introduction to example applications is given in section 4 followed by a discussion of possible present and future use cases in section 5.

2 Conceptual framework

2.1 Definition and classification of compartmental systems

The most parsimonious and general description of a compartmental system is a graph in the mathematical sense, which is a tuple of two sets, the set of nodes, and the set of edges. In the particular graphs that describe compartmental systems the compartments (or pools) form the nodes and the fluxes between them and the exterior the edges. The fluxes are allowed to be functions of time and the contents of the pools exclusively. This is usually referred to as 'well mixed' or 'kinetically homogeneous' compartments.

If we assume any arbitrary ordering of the pools we can represent the mass (content of the pools) by an ordered tuple $\mathbf{x}$. Because mass is a non-negative quantity, this vector of mass contents can only occupy the non-negative orthant of the state-space; i.e. $x \in \mathbb{R}^n_+$. Likewise the mass, that the system receives from outside is represented by a tuple of mass input fluxes (mass over time) $u \in \mathbb{R}^n_+$. It has been shown in (Jacquez & Simon, 1993) that for smooth enough fluxes, mass is transferred among compartments and released back to the external environment can be according to rates encoded in a compartmental matrix $B \in \mathbb{R}^{n\times n}$. Therefore, we can write the dynamics of a compartmental system as a set of ordinary differential equations of the form

$$\dot{\mathbf{X}} = \frac{d\mathbf{X}}{dt} = \mathbf{I}(\mathbf{X}, t) + M(\mathbf{X}, t)\,\mathbf{X} \tag{1}$$

The key property of compartmental systems is, in order for the system to balance mass, that the square matrix $M = (M_{ij})$ exhibits three main properties

1. $M_{ii} \leq 0$ for all i,
2. $M_{ij} \geq 0$ for all $i \neq j$, and
3. $\sum_i M_{ij} \leq 0$ for all j.

Then, M is called *compartmental* and governs all internal cycling of material as well as the exit of material from the system. We describe the relationship between the graph and matrix based descriptions in appendix Appendix A.

We distinguish between different types of compartmental systems, according to linearity and autonomy. If the vector of inputs and the compartmental matrix depend on the vector of states in system (1), we call it non-linear, and linear otherwise. Similarly, if the vector of inputs and the compartmental matrix depend on time, we call the system non-autonomous, and autonomous otherwise. Both descriptions of compartmental system, the intuitive graph of pools and fluxes and the matrix based formulation as an ODE system have advantages for some applications. While the graph based description allows for composition of subgraphs and decomposition of models into e.g vegetation and soil part, the ODE description facilitates the description and implementation of some advanced numerical algorithms as explained below. In our approach both formulations are converted into each other automatically as need arises. This effortless switching between the viewpoints is the main facilitator of the new level of transparency of our approach, their combination enables the computation of transit time and mean age systems for subsystems of selected pools with no more effort than defining the pools constituting them.

2.2 System and sub-system level metrics: contents, fluxes, age, transit time

Compartmental systems can be described by a set of building blocks and metrics. We will show later that the formulation of these metrics as return values of strictly typed functions does make the creation, inspection and comparison of compartmental systems much more feasible by software tools but it can also guide the scientific discussion about what is to be considered a metric. It is the first of the following criteria:

1. In our definition we require that a metric must be unambiguously computable by a function of other metrics or building blocks of a model.
2. Preferably metrics should be applicable to all (nonautonomous, nonlinear) compartmental systems, i.e. not rely on e.g. equilibrium conditions.
3. Preferably metrics should also be physical properties i.e. have a clearly defined way to measure them at least in principle.

Beside the content of pools the whole system or subsystem and fluxes between them, ages, and transit times are key quantities of compartmental systems that fulfill all of these criteria.They help to better understand underlying system dynamics and to compare models with different sizes or structures. They can in principle be measured: While age describes how old material in the system is, transit time describes how long material needs to travel through the entire system from entry to exit (Bolin & Rodhe, 1973; Sierra et al., 2017). They are applicable to general nonautonomous nonlinear systems. This makes them the first choice for model comparisons.

There are metrics that are not suitable to compare *all* compartmental systems because they make further assumptions, e.g. the existence of equilibria. General compartmental systems can be nonlinear and nonautonomous, in the latter case making the con-

cept of equilibrium itself meaningless and in the former the computation of the possibly empty set of fixed points by far to difficult to attempt (by a function). If we wanted to include a strictly typed function to do it, its argument would have to be of a new type for a linear autonomous system. Achieving the same flexibility and complete transparency w.r.t. flux, rate, or matrix centered description will necessitate additional types for constant rates or linear fluxes. We note that metrics build upon equilibrium assumptions fail to meet criteria 2..

Another class that fails at least one of the criteria are properties that can still be unambiguously defined by a function and therefore computed but are not physical quantities and are therefore to be considered properties of the model rather than reality. [1] Flux rates and compartmental matrices, and more importantly (some) properties derived from them, fall into this category. One example is 'turnover time' which is defined as the quotient between content of and outflux from a pool, or as the inverse of the rate. 'Turnover time' is an interesting example as it violates, 3. or 2., because it turns out that for a one pool system in equilibrium it actually coincides with the mean backward transit time which could be approximated by many measurements but looses this interpretation in any non-equilibrium state.

More recently proposed metrics like 'carbon storage capacity', 'carbon storage potential' and 'residence time' suffer from the same issue. comment: @Kostia, could you collect papers citing CarbonStoragePotential and CarbonStorageCapacity in the bib file and cite them here? They have a physical meaningful interpretation only in equilibrium but loose it if their formal definition is extended to the transient case (nonautonomous compartmental systems) as in (Luo et al., 2017) While we could implement a function applying such a definition, fulfilling 1. it would violate 3. and because of the name also be very misleading. A user asking for 'ResidenceTime' (by applying such a function) would reasonably expect the transient 'time of residence' of material and not the time of residence the material *would* have *if* the system was frozen at this moment *and* allowed (infinite) time to reach its equilibrium. comment: @Holger In the following paragraph I played it save. Carlos had mentioned entropy in an old draft, but I don't know if the restriction to autonomous is a feature of the theorie or of `LAPM` A similar restriction of generality applies to applications of Shannon's information entropy as a complexity measure of dynamical systems (Ebeling et al., 1998). For autonomous linear systems it has been used to describe the uncertainty of a particle's path through a compartmental system, quantifying how difficult it is to predict this path. and to compare path properties of models with different number of compartments and connections among them (Metzler, 2020). It's use can be extended to systems in equilibrium.

Surprisingly the literature even contains proposals for metrics that can not be defined as functions because they are ambiguously defined like the following matrix factorization approach. It has been claimed that for most Carbon cycle models the linear version of (1) can be written in product form.

$$\frac{d\mathbf{X}}{dt} = \mathbf{I}(t) + M(t)\,\mathbf{X} \tag{2}$$

$$= \mathbf{I}(t) + A\xi K(t)\,\mathbf{X} \tag{3}$$

[1] That a flux 'rate' should be considered a model property (or latent variable) rather than a physical property becomes more apparent when we consider processes that treat material of different age or position in the pool differently. In this case the assumption that the material in the pools is 'well mixed', and can therefore be treated by the *same* exit 'rate' would be violated, and the model no longer 'compartmental. However such processes clearly exist and fluxes and mass are still measurable. The concept of a single 'rate' applied to all material in a pool is implied by the definition of compartmental systems.

comment:

@Kostia, could you collect the 'matrix approach references' (w.r.t. the decomposition into $A\xi K$ in the TEE-clean.bib file and cite them here? While it is certainly possible to derive (2) unambiguously from (3) the opposite direction is not possible. Neither is the form (3) as general as (2) nor is the decomposition uniquely defined. While the issue with generality could be solved similarly to the equilibrium issue by introducing subtypes of decomposable matrices and fluxes the issue of ambiguity prevents the implementation of a function in the mathematical sense, which requires the return value to be unambiguously defined by the arguments. This actually shows that any analysis built upon the decomposition is not suited to discuss compartmental systems in general, not even if they can be written in the form or (3). This is relevant since such results have been published without discussion of the inherent ambiguity. We show in detail why this is the case in appendix Appendix A.

To summarize, among the traditionally proposed metrics for compartmental systems fulfill our conditions of unambiguous definition, general applicability to nonautonomous nonlinear and physical interpretability as principally measurable. The few that do are: contents of pools,the system or subsystems, the fluxes into, between or out of them and the age and transit times distributions for the whole system or parts of it. To provide those metrics for all models is therefore a priority that is reflected by the structure of the project: While we provide tools for many special cases and specialized results in the depth of our libraries, the most general results are made much more accessible.

2.3 Compartmental systems in equilibrium

The concept of equilibrium is restricted to autonomous systems. It does not even make sense to ask the question otherwise. If the autonomous system is nonlinear it is possible but not certain that an equilibrium exists. The only case where we can expect an equilibrium are linear, autonomous, pool models,

$$\dot{\mathbf{X}} = \mathbf{I} + M\,\mathbf{X}, \qquad \text{with} \quad \mathbf{X}(t_0) = \mathbf{X}_0. \tag{4}$$

for which interesting properties can be obtained by the `LAPM` package. The equilibrium x^* is defined by the condition $\dot{\mathbf{X}} = 0$ which translates to $-M\mathbf{X}^* = \mathbf{I}$ which means that for pool contents $\mathbf{X}^*$ the influxes $\mathbf{I}$ match the outfluxes $M\mathbf{X}^*$ exactly. It is straightforward to see that for this to happen all pools with influxes must be connected (possibly via other pools) to an outflux of the system, and that the (constant) rates for all the flux rates out of all pools along this paths are greater than zero, since an input receiving pool p without these conditions would necessarily grow over time, violation the equilibrium condition $\dot{\mathbf{X}}_p = 0$. Interestingly these conditions also guarantee that M is invertible and the equilibrium therefore uniquely determined by:

$$\mathbf{X}^* = -M^{*-1}\,\mathbf{I}^*. \tag{5}$$

Although at different times different material moves through the system, the size of the pools does not depend on time if the system is in equilibrium: $\mathbf{X}(t) = \mathbf{X}^*$ This is also true for other properties such as the age distribution of mass in particular compartments and in the entire system and the transit time distribution, which is defined as the time it takes masses in the input flux to appear in the output flux. Although the material moving through the system does change the amount, age and transit time distributions do not. They are in fact characterized by the Phase Type distribution, which depends on compartmental matrix B and the equilibrium solution x^* for the system age distribution and the B and the input u for the transit time distribution (Metzler & Sierra, 2018). Compartmental systems at equilibrium have similar properties as continuous-time absorbing Markov Chains (Metzler & Sierra, 2018). Therefore, we can obtain other quantities of interest such as the path entropy of particles that travel across the system and

the occupation time of particles inside compartments (Metzler, 2020). These properties of linear autonomous compartmental systems at equilibrium can be obtained with the `LAPM` package.

Interestingly the properties of linear autonomous systems in equilibrium can also be computed for nonlinear systems in equilibrium if such an equilibrium exists.

$$\dot{\mathbf{X}} = \mathbf{I}(\mathbf{X}) + M(\mathbf{X})\,\mathbf{X}, \qquad \text{with} \quad \mathbf{X}(t_0) = \mathbf{X}_0, \tag{6}$$

In equilibrium the system is indistinguishable from a linear autonomous one

$$0 = \dot{\mathbf{X}} = \mathbf{I}^* + M^*\,\mathbf{X}^* \tag{7}$$

with $M^* = M(\mathbf{X}^*)$ and $\mathbf{I}^* = \mathbf{I}(\mathbf{X}^*)$ If the inverse M^{*-1} exists transit time and age distributions can be computed (Metzler & Sierra, 2018). and therefore also nonlinear autonomous systems at equilibrium can be analyzed with the by the `LAPM` package. Note however,that (5) is useless to determine $\mathbf{X}^*$ and no such $\mathbf{X}^*$ might exist for some systems while others may have multiple fixed points and the age and transit time distribution may be very different for these different equilibria.

2.4 Time Evolution

We consider now linear non-autonomous systems of the form

$$\dot{\mathbf{X}}(t) = \mathbf{I}(t) + M(t)\,\mathbf{X}, \qquad \text{with} \quad \mathbf{X}(t_0) = \mathbf{X}_0. \tag{8}$$

In this case, the inputs and the compartmental matrix are time-dependent and the system never converges to a fixed-point solution. In most cases, an analytical solution cannot be obtained, but the solution can be obtained numerically. In particular, the solution for systems of the form of equation (8) can be written as

$$(t, t_0, \mathbf{X}_0) = \Phi(t, t_0)\mathbf{X}_0 + \int_{t_0}^{t} \Phi(t, \tau)\mathbf{I}(\tau)\mathrm{d}\tau. \tag{9}$$

The state transition operator $\Phi(t, t_0)$ is a matrix-valued function that multiplied with the state x_0 at t_0 transitions it to the state $x(t)$ subsequent time $t > t_0$. It is numerically computable by solving an matrix ode derived from (2.4) From $\Phi(t, t_0)$ we can obtain not only the temporal evolution of the solution $\mathbf{X}(t)$ but also of the distributions of ages of the mass in the compartments and in the entire system (Metzler et al., 2018). The `CompartmentalSystems` package provides all the functionality necessary to do these computations, which rely on a description of the time-dependent input vector $u(t)$ and the compartmental matrix $B(t)$, as well as initial age distributions for the compartments.

Furthermore, these computations can also be obtained for nonlinear systems of the form

$$\dot{\mathbf{X}}(t) = \mathbf{I}(t, \mathbf{X}) + M(t, \mathbf{X})\,\mathbf{X}(t), \qquad \text{with} \quad \mathbf{X}(t_0) = \mathbf{X}_0. \tag{10}$$

by numerically solving (10) and plugging the solution $x(t, t_0, x_0)$ back into it, which results in $\tilde{M}(t) = B(t, \mathbf{X}(t, t_0, \mathbf{X}_0))$ and $\tilde{\mathbf{I}}(t) = \mathbf{I}(t, \mathbf{X}(t, t_0, x_0)$ i.e a linear system in the form (8). Therefore, age and transit time distributions can be obtained for nonlinear nonautonomous systemreimplements along the specific trajectory. Detailed methods for the computation are provided in Metzler et al. (2018)

3 The `Python` packages

3.1 `LAPM`

Linear Autonomous Pool Models (`LAPM`) is a python 3 package for the study of autonomous compartmental systems at equilibrium such as those described in section 2.3. It implements the LinearAutonomousPoolModel class, with methods for the symbolic and numerical solutions of the steady state content in the compartments, and the steady state release out of the compartments. For transit time, system age, and pool age, it provides symbolic and numerical computations of distribution densities, cumulative distribution functions, mean, standard deviation, variance, higher order moments, and Laplace transforms.

For the analysis of compartmental systems in analogy to absorbing Markov chains, `LAPM` provides methods for the computation of the entropy rate per jump, the entropy rate per unit time, and path entropy. It provides the class DTMC (discrete-time Markov chains), with methods to compute the fundamental matrix, stationary distribution, and expected number of jumps of the Markov chain.

3.2 `CompartmentalSystems`

This package deals with non-equilibrium trajectories of compartmental systems. In particular, it provides the class smooth_reservoir_model to describe symbolically the general class of non-autonomous nonlinear compartmental dynamical systems of equation (1). It does not require code for numerical computations or model simulations, but rather defines the underlying structure of the respective model. All fluxes or matrix entries are expected to be `SymPy` expressions.

To obtain numerical results, the package provides the class smooth_model_run, which is initialized with the initial conditions of the system of equations, a set of parameter values, and a time sequence. It computes the solution trajectory for the given initial conditions and parameter values , finds the corresponding linear system with the same solution following the strategy described in section 2.4 computes the state transition operator $\Phi(t, t_0)$ for these solution trajectories and provides methods to obtain time dependent densities with corresponding moments and quantiles for system age, compartment age, and transit time.

An additional module provides functions to obtain initial age distributions required for the computation of time-dependent age distributions. This module uses `LAPM`.

3.3 `ComputabilityGraphs`

This package was specifically developed for use in `bgc_md2` but is also usable in separation. It allows a declarative description of results (model properties) i.e. completely abstracting from the way they are obtained as e.g in a `Make` [2] file. This abstraction is the precondition for queries as in other declarative languages like e.g. `SQL`, whose purpose is the comparison of data, as opposed to the implementation of data base software. Comparison of models w.r.t. their predictions poses a similar challenge: The amount of e.g. Carbon or Nitrogen in a compartmental system or their ages or transit times through it are (in principle) measurable quantities. They do not change if the description of the model is changed from a graph (pools and fluxes) to a matrix or a product of matrices. Instead of forcing a user of the data base to use a fixed description it is much more desirable to allow as many equivalent ones as possible *and* make the equivalence explicit by well defined mappings i.e. *functions*.

[2] Which won the ACM software system award in 2002

`ComputabilityGraphs` facilitates exactly this. It implements a class `CMTVS` which stands for **C**onnected **M** ulti **T**yped **V**ariable **S**et. Instances consist of a set of variables with unique type (only one variable per type) and a set of type annotated functions that exclusively use these types in their signature (as arguments or return values) which we call *computers* from now on. This combination implicitly implements a declarative model description language to describe results of computations by requesting the type of the result.

It also confines what we consider a model property, building block or metric. The network of functions using these types as arguments or return values enforces an unambiguous definition, which prevents the description of models to become vague, e.g. the compartmental matrix is a function of the internal and outgoing fluxes and the ordering of the state variables. This rigorous description actually defines computability graphs that we exploit mainly in the following ways.

1. To compute which types of information (target results) are obtainable given the types of the provided variables. Since a `CMTVS` instance knows the types of all its variables and the signatures of all available functions, it can iteratively add the result types of all applicable functions until no more result types can be reached. This is a forward graph search (Fig. 2).
2. To find out what type of information is *missing* to obtain a target variable (backward search). Fig. 3 shows the bipartite dependency tree for a quite complex numerical result, printed by the `CMTVS` instance. The red node T35 at the bottom represents the target that we want to compute: `NumericVegetationCarbonMeanBackwardTransitTimeSolution`. The green nodes are the building blocks that we have provided in the model description so far. The light red nodes show that are not provided but have to be computed via the function represented by the grey nodes. This is possible if all their (ultimate recursive) dependencies are green nodes. In this example this is the case except for nodes T70 and T38 , `StartConditionMaker` [3] and `VegetationCarbonStateVariableTuple` respectively.
3. To actually compute the result of a targeted type. If a result type is in the computable set determined by 1. then the search tree created under 2. can be traversed in reverse, starting at the given nodes and ending in the final result.

Using 1. and 3. together a `CMTVS` can add get methods for computable results dynamically. This is very useful for the user interface in interactive python sessions, including Jupyter notebooks since on pressing the tab key, the python interpreter suggest available method calls as autocompletion options for any object followed by a "." For an `CMTVS` object these methods become more numerous automatically as more and more information is added to it. Apart from the user interface 1. is necessary for queries. If we have a (large) set of different `CMTVS` objects and want to compare them with respect to a certain property, we must first find out for which of them this property is actually computable. A welcome side effect of the CMTVS model description language is the possibility to include models about which we know very little. In a traditional database these record would most likely be incomplete, whereas here we just get offered fewer computable results. This is especially useful for the creation of new models, in the process of which one naturally starts with a small set of variables that is gradually extended. Typically one of the first steps is a dictionary of symbolic flux equations, which already allows the visualization of the compartmental graph or flow diagram, symbolic matrices and vectors and is extremely useful for debugging.

[3] Variables of this type are actually a functions that can compute a set of consistent start conditions (mass, age density, mean age), which we note here to show that the concept of type is not confined to a traditional data types.

3.4 The biogeochemical model database `bgc_md2`

This is the central package and can be seen as a front end to `CompartmentalSystems` and `LAPM` facilitated by `ComputabilityGraphs`. For the following discussion it is important to note that `bgc_md2` is a library, rather than a framework, since there is no 'inversion of control', i.e. `bgc_md2` as well as all the other packages are called from normal python code, rather than `bgc_md2` calling user code in a restricted execution environment as a framework would do. Although internally `ComputabilityGraphs` acts like an extended compiler (not only testing type consistency of function calls but actually suggesting them) for a DSL tailored to the domain of compartmental models, it is technically represented by `bgc_md2`'s API (consisting of the provided types, functions, and the dynamically created methods of the `CMTVS` objects for computable properties). This makes it much more flexible to use than a framework. We can use the full tool set of the *python* ecosystem to create instances of the model building blocks and to postprocess results afterwards. While any comparison requires standards i.e. rigorously defined properties which can be formulated for all models and therefore naturally suppresses idiosyncrasies of models, it can be very helpful to use these idiosyncrasies for the creation of a model. This is possible since we can use absolutely unrestricted python code to create the model building blocks. Imagine for example a model that includes pools in many soil layers with similar connecting fluxes. A model description code using `bgc_md2` could use loops to express this. This also applies to postprocessing. Some of the functionality in `LAPM` and `CompartmentalSystems` is not generally applicable to *all* models in `bgc_md2`'s collection and therefore cannot (and should not) be offered by the 'computers'. This very deliberate restriction is however no problem since nothing prevents a user to apply them (or any other bit of code) to the results of a `bgc_md2` computations. Armed with this flexibility we know that our DSL does not have to provide everything and can restrict it to sensible categories for comparisons. In particular `bgc_md2` provides:

1. Types defining **building blocks and comparable diagnostic results** specifically tailored to compartmental models, e.g. `CompartmentalMatrix`, `InternalFluxesBySymbol`, `NumericVegetationCarbonMeanBackwardTransitTimeSolution` ...[4] specifically including the symbolic expressions for pool contents, in- out- and internal-fluxes for subsystems, single pools or the entire model, as well as their numerical counterparts obtained from by solving the parameterized IVPs and the most general diagnostics e.g. transient mean-age and transit time solutions. Since many of these types are based on a symbolic mathematical representation, using `SymPy`, many symbolic results are computable without the need to know parameters or data to run the models. Using the underlying graph representation as sets of pools and fluxes, we can reorder pools and thereby automatically transform the compartmental matrices, group them into different subsets (e.g. vegetation, soil, litter, carbon, nitrogen ...), substitute pool names, get mathematical expression for cumulative fluxes e.g. from vegetation to soil or simply plot the graphs (Fig. 4). Using this symbolic transformations we can compare models that might have looked very different initially, simply due to arbitrary choices of pool names or their even more arbitrary order in which they appear.
2. A set of type annotated functions (from now on called computers) operating on those types, which combined by `ComputabilityGraphs` form a much simplified interface to *many algorithms* in `CompartmentalSystems` and `LAPM` to compute diagnostic variables for *many models* in `bgc_md2`.
3. 30+ vegetation, soil or ecosystem models for carbon and nitrogen cycling as reusable python modules using the building blocks in a flexible way.

[4] A complete list can be produced by a single command

3.4.1 Intentionally Missing Computers

It is also interesting to note what `bgc_md2` intentionally does not implement: A trivial example is that the computation of equilibria is missing from `bgc_md2`'s set of strictly typed functions. While we can easily compute the uniquely defined fixed point of a linear autonomous system, and indeed provide functions to do so in `LAPM`, we have to resist the temptation to make them available via `bgc_md2`'s set of types and computers for *all* compartmental systems. This restriction also pertains to equilibrium start age distributions. If they simulation is to be started from an equilibrium this is reflected not only by the correct equilibrium but also a specific age distribution. At the moment neither equilibria nor the start age distributions are inferred automatically, but have to be provided per model by user code to clearly define responsibilities (although, thanks to `LAPM`, the user code amounts to only three lines). While we could in the future provide special types for liner matrices or fluxes, constant rates, along with their appropriate constructors implementing automatically testable criteria for accidental misdeclaration, implement computers for them and thus formalize comparisons between linear models this has not been done yet and such results are not suggested automatically by `bgc_md2`. Users can however use the many functions in `LAPM` or `CompartmentalSystems` directly.

We also avoid 'computers' depending directly or indirectly on ambiguously defined matrix factorizations, as we mentioned above and in explained in detail in appendix Appendix A. These computations can still be performed aided by `LAPM` or `CompartmentalSystems` on intermediate results provides by `bgc_md2` computers but also not automatically, since results would be ambiguous (in contradiction to the mathematical definition of a function which is our criteria for admission).

4 Example Applications

It is impossible to fully exhibit the potential of this approach without examples. To this end we created some illustrative `Jupyter` notebooks that are available via binder online without the need to install the packages. This paragraph contains only pointers to those much more elaborate notebooks and some example plots from them. The examples demonstrate how `bgc_md2`:

1. Simplifies and unifies the creation of a new model from scratch while using the symbolic and graphic diagnostic capabilities to inspect it while we build it and use `ComputabilityGraphs` to point out what missing information we have to provide to make desired results computable. This is demonstrated in binder˙notebooks/illustrativeExamples/createModel.py
2. aided by `ComputabilityGraphs` 's ability to compute which properties are computable allows to query the collection of models that are already part of `bgc_md2`. This is demonstrated in binder˙notebooks/illustrativeExamples/databaseAspects.py
3. Several Models for which numeric parameter values and driver data are available can be compared w.r.t. abstract properties. We chose the mean age and transit time of the vegetation and soil subsystems since they are defined for all models while the concrete pools of the models differ and not only have different names but also different meaning. (Fig. 6, Fig. 7)

We will look closer at 3.. It is interesting to see how only the same 10 variables as shown in Fig. 2 (T18: InFluxesBySymbol, T20: InternalFluxesBySymbol, T31: NumericParameterization, T35: NumericSimulationTimes, T46: OutFluxesBySymbol, T56: SoilCarbonStateVariableTuple, T57: StartConditionMaker, T59: StateVariableTuple, T60: TimeSymbol, T69: VegetationCarbonStateVariableTuple) are actually given, and how it is possible to compute such a relative complex result like the mean backward transit time for the vegetation part automatically Fig. 5. comment:

@Holger, Kostia:

We should try to publish the following in PNAS as an unexpected short Corollary to (Metzler et al., 2018) called 'Inverse Pool Algebra', because it is generalizable to any subsystem. The actual computation can be made more elegant: Similar to the extension of the original IVP to the age moment system, we can add more equations for the subsystem age moments and solve them simultaneously without first solving the original system and subsequently substituting the solutions as described below. One special case is the computation of "pool" ages (age w.r.t. entry into a pool not the whole system) with IVPs (instead of our more elaborate (unpublished) approach with the bins that also works for non-well mixed). Kostia asked me to do it (and patiently waited until the plots looked reasonable) and after only several refactorings I realized that this is the "Pool Algebra backwards" Its actually quite surprising that we can compute the age with respect to a set of pools in the middle with an IVP since the whole well mixed approach has no explicit concept of crossing a pool boundary for a single particle (only implicitly by the Markov Chain Interpretation...) The connection is the start condition (does not have to be equilibrium based, pinup or empty would also do) and the Input Age of zero for the flux into the subsystems, Crazy sh...

The short version of the mathematical description is as follows:

1. Assemble the ODEs for the whole system, from the flux equations for in- out- and internal fluxes that are given in the model descriptions.
2. Apply the `StartConditionMaker` a function that yields a consistent tuple of contents, mean ages, and age distributions for all pools to the ODE. (In this case based on the assumption of the trendy9 S2 experiment of starting in equilibrium, which can only apply to a frozen system within which all time dependent functions have been substituted by there value at $t = 0$. Other ways to compute start conditions is to assume empty pools, with age and mean age 0 or the result of a 'spin up' from this situation for given or assumed functions of *past* times.)
3. Solve the IVP for the whole system.
4. Create the ODEs for the vegetation subsystem: Extract the sub graph for the vegetation pools, find all fluxes into it and out of it. Use the symbolic representation of the flux functions to add up the incoming fluxes to a Vegetation Input. Compute the CompartmentalMatrix for the vegetation subsystem (which will also depend on pools, that are NOT part of this subsystem.)
5. Substitute the solutions for the pool contents computed under 3.. into the Influx and matrix expressions computed under 4.
6. Apply the (same) `StartConditionMaker` to the vegetation subsystem. Together with 4. this forms a new vegetation IVP.
7. Extend the vegetation IVP by the first age moment (mean) equations.
8. Solve the mean age vegetation IVP. While this (unnecessarily) reproduces the solution for the vegetation pool contents as function of time, the mean age solution differs because influx into the vegetation system (that actually comes from other pools) now has age 0.
9. Substitute the solution computed under 3. into the fluxes out of the vegetation pools (w.r.t. the whole system these are either outfluxes, like autotrophic respiration, or internal fluxes e.g. fluxes to soil or litter pools). This makes these fluxes functions of time.
10. The mean backward transit time is the mean age of the material in the vegetation outfluxes and can be computed from the results of 8. and 9..

We could have implemented a separate function to do all this (which we actually did in the beginning), but we broke it down in to much smaller functions that only compute one step and now `ComputabilityGraphs` reconstructs the whole computation from these smaller bits. This has several consequences:

1. The necessary user code is one line long:

```
mvs.get_NumericVegetationCarbonMeanBackwardTransitTimeSolution()
```

Where mvs is the `CMTVS` instance imported from the data base for the respective model. The result of this is an array that can be plotted over time as we show in Fig. 6. The computation for the soil part is nearly identical except for a different subset of nodes that leads to a different subgraph.

2. The code is *declarative* i.e only describes the result not the computation. In particular the result would have been obtained by the same line of code if the model had been described using other building blocks, (`CompartmentalMatrix` and `InputTuple`) because `ComputabilityGraphs` would have looked for, found and applied the additional conversion functions automatically.
3. Because the code is declarative we could even have iterated over all the models (implemented as python modules) in the 'data base' to first find all for which we *can* compute it (somehow) and then do it. (Although in this case the choice of model was more constrained by the availability of common driver data)

5 Conclusions

We implemented a practically usable tool for the creation, collection, inspection and comparison of compartmental models and demonstrated how it could be applied quickly to the practical problem, of comparing several models with respect to a numerical result complex enough to make its implementation on a per model basis prohibitively expensive. We created examples to teach new users how use the existing capabilities, how to query the collection and how to add models to it. These immediate practical benefits are sufficient for many simple application and are to our knowledge unprecedented. In the process some lessons have been learned, that could be valuable to anybody undertaking a similar project or interested in extending our work. An important one is the fertile feedback of the self imposed restriction to properties that are either results or arguments of computations on the clarity of the description of features. Another one is the implementation of all the tools as libraries rather than a framework, which allows the use of the provided tools in situations that we do not anticipate yet and allows us to be strict in the data base part without creating a bottleneck. We have also collected some more general observations: Creating collections of many models and many metrics consistently applicable to all of them is hard, finding well defined ways to allow users to contribute even harder, documenting them comprehensively harder still. The necessity for rigorous definition, absolute clarity and avoidance of duplication becomes more and more important as the collection of models grows. While we have reached a point where the contribution of new models is easy enough to be done by the users of the software, implementation of new features will likely require the permanent attention of a technical project lead for the lifetime of the project, since occasional refactoring is likely to be necessary to keep the code base maintainable.

5.1 Possible Extensions and Improvements

5.1.1 More Models

The easiest way to extend the data base is to contribute new models. We have created an example notebook as an interactive tutorial. Since very few building blocks are necessary, adding a new model is actually very simple. The symbolic part takes usually much less than a day, and adding parameters much less time than finding them in the literature. Since `CMTVS` instances are normal python objects and have update and remove methods, models can use other models as source and extend them dynamically. The already stored models descriptions are `Python` modules and can import others or be imported by them. When you are happy with your code a pull request will put your model into the default location and make it subject to the tests.

5.1.2 More Real Parametrizations and Driver data

Many of the model descriptions contain a variable of type `NumericParameterization` which contains dictionaries for parameter values and functions, mapping the symbolic description to a runnable model. These `NumericParameterization` instances are usually extremely small example data sets, whose purpose is to give users some arguments to test the numerical results with. This is intended. `bgc_md2` is not a place to store data. However we would like to apply models to real data. Usually via model and data set specific code that makes available data accessible to our models. It is surprisingly tedious comment:

Big THANKS to Veronika, for doing it for many models and for some models impossible to find scientifically meaningful parameters in the literature. In the case of earth system models we would need such a set of parameters and driver functions for each spatial pixel. Even for model intercomparison projects like trendy comment:

@Kostia Trendy Quotation? these parameters are usually not publicly available and it is often not possible to estimate them accurately even from synthetic model output. [5] We have created a small trendy9 specific package for gluing the data to `bgc_md`'s models, with parameter estimation code for four models. It would be very beneficial to have more gluing code for more data sets available...

5.1.3 New Metrics and Building Blocks

Adding 'computers' is technically very simple. It will make you think though. In the 'language' of `bgc_md2` anything worth saying about a compartmental model is the result of a computation, or an argument that influences other computations or most likely both. Adding a new computer is the simplest case, since one has only to make sure that for *any* arguments that fit the type signature a unique result can be computed. If new argument types or new result types have to be introduced it is usually worth asking if these arguments could be computed from something simpler. The reiteration of this process can lead to astonishing results. At some point we wanted to add information about the vegetation subsystems of some ecosystem models. We needed the related subsystem compartmental matrix as in (Ceballos-Núñez et al., 2018). Adding it as a model property would have been very error prone, adding a projection operation on the matrix would have been possible but far more complicated than the graph description that is now implemented and allows to specify a vegetation state variable tuple from which not only the vegetation compartmental matrix can be computed but any other vegetation related result too. Adding a new word to the vocabulary is a very powerful extension.

5.1.4 Algebraic Types

The sets of 'computers' and their argument and result types is by no means comprehensive. In fact there are many results that could be implemented quite easily. E.g. we implemented the two types `VegetationCarbonStateVariableTuple` and `SoilCarbonStateVariableTuple`. It would have been easy to created something similar for e.g. Nitrogen. However the number of types would quickly become large since their are also related types that would have to be duplicated, i.e. `NumericVegetationCarbonSolutionArray` or `NumericVegetationCarbonMeanBackwardTransit`. To combat this 'combinatoric explosion' it would be more elegant to express these relationships between types by a concept called algebraic data types which is used e.g. in `Haskell` but also available in `Python` 's type hint syntax but not yet implemented in `ComputabilityGraphs`. The concept is actually very simple as e.g. a type that uses another type as parameter like a list of integers `List[int]`. Algebraic types would give us a bigger namespace for

[5] Usually due to identifiability issues. 30 or more parameters can hardly be expected to be estimable from much fewer data streams

less awkward type names and also avoid some boilerplate code for the 'computers'. [6] Another related extension of `ComputabilityGraphs` is to allow 'computers' that return tuples. This could be used to minimize the number of recursive computer application for sets of desired results.

5.1.5 Automatic Code Generation, Other Languages and Integration into Earth System Models

`bgc_md2` uses symbolic math to describe the model equations. It then uses `SymPy`'s facilities to first translate these expressions into regular python code and execute it in a special environment that can be influenced. `SymPy` provides a printer module that allows the translation of `SymPy` expressions to many different programming languages including but not limited to C, Fortran, Java, R and Julia. So that models could also be translated to these languages automatically. A possible application of this ability would be to automatically compile different land-carbon models from `bgc_md2`'s collection into an earth system model without the need to reimplement them in the targeted language. Remarkably `ComputabilityGraphs`could also easily be extended to use collections of 'computers' in other languages. It acts much like a compiler and could use the graphs to automatically create code calling the 'computers' recursively in the right order. [7] This is extremely useful for the next possible extension.

5.1.6 Benchmarking Testbed and Stepping Stone for the Implementations of Diagnostics in Other Projects

Our packages have hundreds of unit test cases. If the integration of transit time computations into an existing earth system model was desired, these test could be adapted much faster than created from scratch. For nonlinear and nonautonomous systems, where analytical tests are hard to come by, `bgc_md` can be used as a benchmark for any desired model (which could be implemented much more quickly there and then used as a reference in a set of tests for the new implementation.

5.1.7 Documentation, Error Messages

Our software tools are written in `Python`. This has advantages:

- The availability of hundreds of libraries for pre- or postprocessing of model building blocks or results.
- The flexibility of a very permissive dynamically typed language which allows for rapid prototyping and interactive use of the tools we created.

but also disadvantages: The same permissiveness that allows for rapid prototyping and interactive use can also make the detection of unintended misuse harder. In strictly typed languages like Haskell many unintended usecases are already detected at compile time and reported as errors. This 'fail early and hard' philosophy, facilitates the creation of 'fool proof' 'software', whereas interactive environments, notably R and python follow almost the opposite approach. They try their best to make sense of anything thrown at them which can delay the detection of an error and thereby remove the effects far from the cause. Complex libraries like `bgc_md2` which sits on top of `CompartmentalSystems` which sits on top of `LAPM` which sits on top of `numpy`, `SciPy` and `SymPy` need extra care

[6] R does allow polymorphism in all arguments by means of `S4` classes, but does not allow algebraic types. So in SoilR the combinatoric explosion also lead to longer and longer type names.

[7] At the moment this is not necessary since the computations traversing the graph in reverse are all performed immediately in python with with a growing intermediate result dictionary

to filter out user input that could cause trouble further down the call stack. [8] This becomes more important the less obvious the computation in question is. While an example makes the user aware of her responsibility for correct adaptation, the same user will expect the API of a 'black box' to be 'fool proof'. This is an ongoing task: While `LAPM` and `CompartmentalSystems` are pretty well documented comment:

Big Thanks to Holger!!! `bgc_md` and `ComputabilityGraphs` are not yet and none of the packages is 'fool proof'.

5.1.8 Polished UI, Web Interface

We implemented some widgets to use in `Jupyter` notebooks (e.g. to show a subset of the models in an interactive table or to explore the computability graphs). These widgets are absolutely basic proofs of concept, mainly intended to icite a skilled web developer (or enthusiastic student in need of a project) to transform them into something much more useful (a click on an edge in a computability graph could generate and print code for this path) and shiny. This is especially true for the graph visualizations. Plotting (mathematical) graphs is a surprisingly hard problem and standard plotting libraries like `matplotlib` or `igrap` support only basic graph layout algorithms. Widgets or web applications using `javascript` would allow to use more specialized libraries like `Cytoscape.js` and make the interactive exploration or publications- ready printing of the compartmental model (as well as the computability graphs) much more beautiful. It would also be relatively easy to host `bgc_md2` based on a server, preferably with access to many datasets. A `Jupyter` server would be the most flexible option for scientific users, but specialized and less interactive websites for model or result presentations could easily be build using the existing example widgets as a starting points.

Appendix A Derivation of Matrix equations

Mathematically Compartmental Models are most economically described as graphs, where the set of compartments $\mathcal{P}$ and the set of non-negative fluxes $\mathcal{F}$ form the the nodes and edges respectively. Choosing one of $n!$ possible ways to enumerate the set of pools $\mathcal{P} = \{p_0, \dots, p_n\}$ where p_0 is the outside world, we write the contents of the pools as x_i for $i \in \{1, \dots, n\}$ and the fluxes as

$$\begin{aligned}\mathcal{F} = &\{I_{0\to j} > 0 \text{ for } j \in \{1,\dots,n\}\}\\ &\cup \{F_{i\to j} > 0 \text{ for } i \in \{1,\dots,n\} \text{ and } j \in \{1,\dots,n\} j \neq i\} \text{ with } F_{i\to j} = 0 \text{ for } x_i = 0\}\\ &\cup \{F_{i\to 0} \text{ for } i \in \{1,\dots,n\} \text{ with } F_{i\to 0} = 0 \text{ for } x_i = 0\}\end{aligned}$$

where $I_{0\to j}$ are influxes from the outside into the system $F_{i\to j}$ are fluxes between pools and $F_{i\to 0}$ are fluxes out of the system.

In general all fluxes can depend on all the x_i and time t (trough environmental factors like Temperature $T(t)$ and moisture $W(t)$.

The influxes don't have to depend on the x_i but internal and out fluxes must at least depend on their source pool content to guarantee the condition that there is no outflux from an empty pool. For every pool we have a mass balance equation.

$$\frac{d}{dt}x_i = \sum_{j\neq i}(-F_{i\to j}(x_1,\dots,x_n,t) + F_{j\to j}(x_1,\dots,x_n,t)) + I_{0\to i}(x_1,\dots,x_n,t) - F_{i\to 0} \quad \forall i \in \{1,\dots n\} \tag{A1}$$

[8] E.g. in some corner cases the underlying libraries might treat a matrix of dimensions n,1 differently than an array of dimension (n,), while users usually do not anticipate this.

Assuming continuity of the fluxes with respect to their source pool $F \in \mathcal{C}^1$ we can write them in product form. $F_{i\to j} = r_{j,i}x_i$ for $i \in \{1,\dots n\}, j \in \{0,1,\dots,n\}$ and $j \neq i$

$$\frac{d}{dt}x_i = -\underbrace{\left(r_{i\to} + \sum_{j\neq i} r_{j,i}(x_1,\dots,x_n,t)\right)}_{=m_{i,i}(x_1,\dots,x_n,t)} x_i + \sum_{j\neq i} \underbrace{r_{i,j}(x_1,\dots,x_n,t)}_{-m_{i,j}(x_1,\dots,x_n,t)} x_j + \underbrace{F_{\to i}(x_1,\dots,x_n,t)}_{I_{\to i}} \tag{A2}$$

$$= -\sum_j m_{i,j}(x_1,\dots,x_n,t)x_j + I_{\to i}(x_1,\dots,x_n,t) \tag{A3}$$

Writing $\mathbf{X} = (x_1,\dots x_n)^T$ for the ordered tuple of all pool contents, and $\mathbf{I} = (I_{\to 1},\dots I_{\to n})^T$ for the ordered tuple of all influxes, we get

$$\frac{d}{dt}\mathbf{X} = \mathbf{I}(\mathbf{X},t) - M(\mathbf{X},t)\mathbf{X} \tag{A4}$$

$-M$ is called the Compartmental Matrix. [9]

A01 Matrix decomposition

Together with a start-value $\mathbf{X}_0$ (A4) constitutes an "initial value problem" (ivp) which can be solved numerically by moving step by step forward in time.

Without further assumptions the system is "nonautonomous" (since either of $\mathbf{I}$ or M can depend on time t) and "nonlinear" since either M can depend on $\mathbf{X}$ or $\mathbf{I}$ can depend on $\mathbf{X}$ in a way that cannot be expressed in the form $\mathbf{I}(\mathbf{X},t) = \tilde{\mathbf{I}}(t) + I_{mat}(t)\mathbf{X}$ with the matrix $I_{mat}(t)$ independent of X.

If $m_{i,i}(\mathbf{X},t) \neq 0$ [10] it is possible to factorize $M(X,t)$ into a product $M = A(\mathbf{X},t)K(\mathbf{X},t)$ where K is a diagonal matrix and the matrix A has only ones on it's main diagonal.

If $u = \sum_{k=1\dots n}\mathbf{I}_k \neq 0$ it is possible to determine the dimensionless vector $\boldsymbol{\beta} = \mathbf{I}/u$ where $\sum_{k=1\dots n}\beta_k = 1$ and write $\mathbf{I}(\mathbf{X},t) = \boldsymbol{\beta}(\mathbf{X},t)u(\mathbf{X},t)$ Using these terms we arrive at

$$\frac{d\mathbf{X}}{dt} = B(\mathbf{X},t)u(\mathbf{X},t) - A(\mathbf{X},t)K(\mathbf{X},t)\mathbf{X}$$

with:

$$k_{i,i}(x_1,\dots,x_n,t) = \left(r_{i\to}(x_1,\dots,x_n,t) + \sum_{l\neq i} r_{l,i}(x_1,\dots,x_n,t)\right)$$

$$a_{j,i}(x_1,\dots,x_n,t) = \frac{r_{j,i}(x_1,\dots,x_n,t)}{k_{i,i}(x_1,\dots,x_n,t)} = \begin{cases} = \frac{r_{i,j}(x_1,\dots,x_n,t)}{\left(r_{i\to}(x_1,\dots,x_n,t) + \sum_{l\neq i} r_{l,i}(x_1,\dots,x_n,t)\right)} \text{ for } j \neq i \\ 1 \text{ for } j = i \end{cases} \tag{A5}$$

[9] Because the enumeration of the set of pools is arbitrary there are, for a model with n pools actually $n!$ such matrix equations, that all describe the same model.

[10] If $m_{i,i}(\mathbf{X},t) = 0$ for some i then some elements of A become undefined. However, this does not mean that we could not write M as a product, just that A cannot be inferred and we have to pretend to know the $a_{j,i}(x_1,\dots,x_n,t)$ for $j \neq i$ and $\forall(x_1,\dots,x_n,t)$ with $k_{ii}(x_1,\dots,x_n,t) = 0$ although we could never learn them from any observed fluxes. The same arguments holds for $\boldsymbol{\beta}$.

The $k_{i,i}$ can be interpreted as the rate of the total flux out of pool i. The elements of column i of A describe then which fractions of this total outflux is transferred to pool j.

A02 Assumption of Linearity

If we assume the model to be linear and nonautonomous the dependency on $\mathbf{X}$ vanishes and we have either

$$\begin{aligned}\frac{d\mathbf{X}}{dt} &= \tilde{\mathbf{I}}(t) + I_{mat}(t)\mathbf{X} - M(t)\mathbf{X} \\ &= \underbrace{\tilde{\mathbf{I}}(t) + (I_{mat}(t) - M(t))}_{L(t)}\mathbf{X} \\ &= \tilde{\mathbf{I}}(t) + L(t)\mathbf{X}\end{aligned} \tag{A6}$$

or if we insist on a non-state-dependent inputs

$$\begin{aligned}\frac{d\mathbf{X}}{dt} &= \mathbf{I}(t) - M(t)\mathbf{X} \\ &= \boldsymbol{\beta}(t)u(t) - A(t)K(t)\mathbf{X}.\end{aligned} \tag{A7}$$

Eq. (**??**) allows for influxes to be dependent on the receiving pool, e.g. for the influx of carbon through photosynthesis to depend on the the size of the leaf pool. Note that L does not have to be compartmental and therefore not factorizable into A and K. Eq. (A7) is that Both exclude certain compartmental models e.g. some with interactions between chemical species. Imagine that some of the pools contain Nitrogen and others Carbon. It is likely that some fluxes out of carbon pools are controlled by the available Nitrogen. Imagine a compartmental system where the startvector consist of Carbon and Nitrogen pool contents: $\mathbf{X} = (c_1, c_2, \dots, n_1, n_2, \dots)^T$, then a flux between carbon pools a and b that depends of the content of nitrogen pool c depends on (a part of) the statevector, which makes it nonlinear.

$$\begin{aligned}F_{a\to b}(\mathbf{X}, t) &= r_{c_i\to *}(n_c, t)\mathbf{X}_a \\ &= r_{c_i\to *}(\mathbf{X}, t)\mathbf{X}_a\end{aligned}$$

A03 Assumption of Factorizability, substrate centered versus flux centered description

For many published models the nonautonomous part can be further localized into a diagonal matrix $\xi(t)$ so that we can achieve constant A and K. It is important to realize two points here:

1. This is not possible for all compartmental matrices.
2. In the cases where it is possible it does not uniquely define ξ.

We can discuss (1) from a mathematical and a modeling viewpoint: The linear version of (A8) is:

$$\begin{aligned}k_{i,i}(t) &= \left(r_{i\to}(t) + \sum_{l\neq i} r_{l,i}(t)\right) \\ a_{j,i}(t) &= \begin{cases}\frac{r_{j,i}(t)}{\left(r_{i\to}(t) + \sum_{l\neq i} r_{l,i}(t)\right)} \text{ for } j \neq i \\ 1 \text{ for } j = i\end{cases}\end{aligned} \tag{A8}$$

From this representation it is clear that the $a_{i,j}$ are only constant if all rates $r_{j,i}$ for $j \in \{0, \ldots, n\}$ contain the *same* time dependent factor $\xi(t)$, which makes the existence of constant A and K an *assumption*. From a modeling point of view the $\xi_{i,i}$ can be seen as a "substrate" dependent rate modifier since it affects everything that leaves the same pool in the same way, whereas $r_{i,*}(t)$ is specific to a single flux an so could be different for different "processes" even if they use the same substrate.

In order to discuss (2) we note that the assumption that we can write $M = A\xi K$ implies that we can also write it as $M = A\tilde{\xi}\tilde{K}$ where $\tilde{\xi} = d\xi$, $\tilde{K} = d^{-1}K$ for any diagonal matrix d. This implies that without further assumptions it is not possible to compute ξ for a given model without a gauge condition like $\xi(T_0, W_0) = 1$ for a some specific temperature T_0 and moisture W_0, which in turn implies that the *baseline residence time* $(AK)^{-1}$ is only defined up to the above mentioned diagonal matrix d. This fact becomes very important when certain properties of models are attributed to either ξ or the *baseline residence time*. Any sensible attribution of this kind has to be shown to be robust to changes of d. **??**

Open Research Section

The data used for the example could be obtained from several servers that offer trendy9 data. The example is however not intended to answer a 'real' research question, but to show how a model comparison could look like, if we *had* access to the parameter sets used in the original trendy9 model runs. We chose 'reasonable' parameters for the models as *if* we knew them for a pixel. All code is available on GitHub and the example notebooks can be interactively explored on binder.

Acknowledgments

Enter acknowledgments here. This section is to acknowledge funding, thank colleagues, enter any secondary affiliations, and so on.

References

Anderson, D. H. (1983). *Compartmental modeling and tracer kinetics* (Vol. 50). Springer Science & Business Media.

Bolin, B., & Rodhe, H. (1973). A note on the concepts of age distribution and transit time in natural reservoirs. *Tellus*, *25*(1), 58–62.

Botter, G., Bertuzzo, E., & Rinaldo, A. (2011). Catchment residence and travel time distributions: The master equation. *Geophysical Research Letters*, *38*(11).

Ceballos-Núñez, V., Richardson, A. D., & Sierra, C. A. (2018). Ages and transit times as important diagnostics of model performance for predicting carbon dynamics in terrestrial vegetation models. *Biogeosciences*, *15*(5), 1607–1625. Retrieved from `https://www.biogeosciences.net/15/1607/2018/` doi: 10.5194/bg-15-1607-2018

Ebeling, W., Freund, J., & Schweitzer, F. (1998). *Komplexe Strukturen: Entropie und Information*. Teubner-Verlag, Stuttgart–Leipzig.

Eriksson, E. (1971). Compartment models and reservoir theory. *Annual Review of Ecology and Systematics*, *2*, 67–84.

Haddad, W. M., Chellaboina, V., & Hui, Q. (2010). *Nonnegative and compartmental dynamical systems*. Princeton University Press.

Harman, C. J., & Kim, M. (2014). An efficient tracer test for time-variable transit time distributions in periodic hydrodynamic systems. *Geophysical Research Letters*, *41*(5), 1567–1575.

Jacquez, J. A., & Simon, C. P. (1993). Qualitative theory of compartmental systems. *Siam Review*, *35*(1), 43–79.

Luo, Y., Shi, Z., Lu, X., Xia, J., Liang, J., Jiang, J., ... Wang, Y.-P. (2017). Transient dynamics of terrestrial carbon storage: Mathematical foundation and its applications. *Biogeosciences*, *14*(1), 145–161. Retrieved from http://www.biogeosciences.net/14/145/2017/ doi: 10.5194/bg-14-145-2017

Manzoni, S., & Porporato, A. (2009). Soil carbon and nitrogen mineralization: Theory and models across scales. *Soil Biology and Biochemistry*, *41*(7), 1355–1379. doi: 10.1016/j.soilbio.2009.02.031

Matis, J. H., Patten, B. C., & White, G. C. (1979). *Compartmental analysis of ecosystem models* (Vol. 10). Intl Cooperative Pub House.

Metzler, H. (2020). *Compartmental systems as markov chains : age, transit time, and entropy* (Doctoral dissertation, Friedrich-Schiller-Universiät Jena, Jena). Retrieved from https://suche.thulb.uni-jena.de/Record/1726091651

Metzler, H., Müller, M., & Sierra, C. A. (2018). Transit-time and age distributions for nonlinear time-dependent compartmental systems. *Proceedings of the National Academy of Sciences*, *115*(6), 1150–1155. Retrieved from http://www.pnas.org/content/115/6/1150 doi: 10.1073/pnas.1705296115

Metzler, H., & Sierra, C. A. (2018, Jul 04). Linear autonomous compartmental models as continuous-time Markov chains: Transit-time and age distributions. *Mathematical Geosciences*, *50*(1), 1–34. Retrieved from http://dx.doi.org/10.1007/s11004-017-9690-1 doi: 10.1007/s11004-017-9690-1

Nash, J. (1957). The form of the instantaneous unit hydrograph. *International Association of Scientific Hydrology*, *3*(45), 114–121.

Rodhe, H., & Björkström, A. (1979). Some consequences of non-proportionality between fluxes and reservoir contents in natural systems. *Tellus*, *31*(3), 269–278.

Sierra, C. A., & Müller, M. (2015). A general mathematical framework for representing soil organic matter dynamics. *Ecological Monographs*, *85*, 505–524. Retrieved from http://dx.doi.org/10.1890/15-0361.1 doi: 10.1890/15-0361.1

Sierra, C. A., Müller, M., Metzler, H., Manzoni, S., & Trumbore, S. E. (2017). The muddle of ages, turnover, transit, and residence times in the carbon cycle. *Global Change Biology*, *23*(5), 1763–1773. Retrieved from http://dx.doi.org/10.1111/gcb.13556 doi: 10.1111/gcb.13556

Sierra, C. A., Müller, M., & Trumbore, S. E. (2012). Models of soil organic matter decomposition: The SoilR package, version 1.0. *Geoscientific Model Development*, *5*(4), 1045-1060. doi: 10.5194/gmd-5-1045-2012

Walter, G. G., & Contreras, M. (1999). *Compartmental Modeling with Networks*. Birkhäuser.

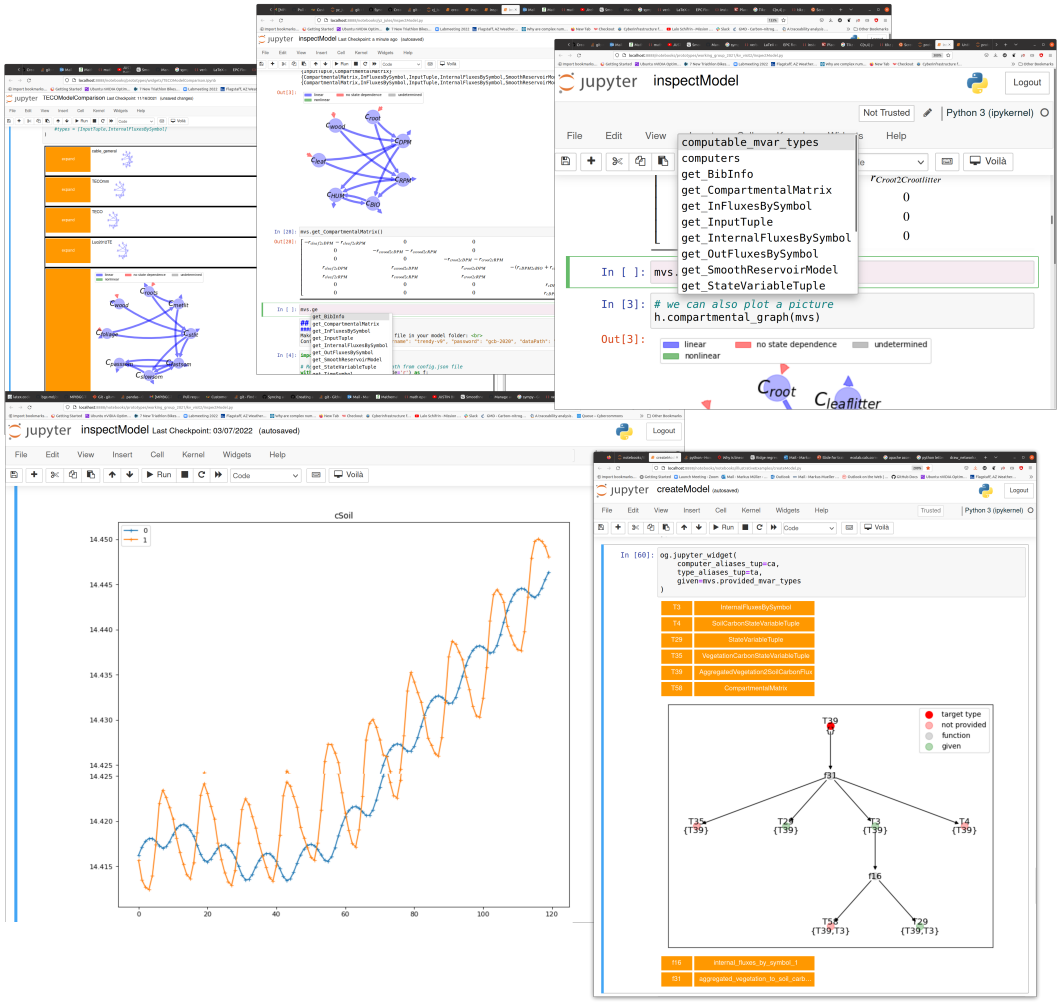

Figure 1: Figure description, top row, left to right: Interactive Jupiter widget with a table of models (orange buttons can be clicked to expand or collapse a more detailed view of the particular model), Model inspection with pool connection graph, which can be derived from the symbolic description along other symbolic properties as flux equations and the compartmental matrix, Zoom into IPython/Jupyter UI, showing methods automatically added by the computability graph library.
Bottom row: Data assimilation with an automatically created numeric model (from symbolic description), Computability graph for a desired diagnostic (aggregated Flux from the vegetation to soil part, showing the additionally needed information to compute the desired result)

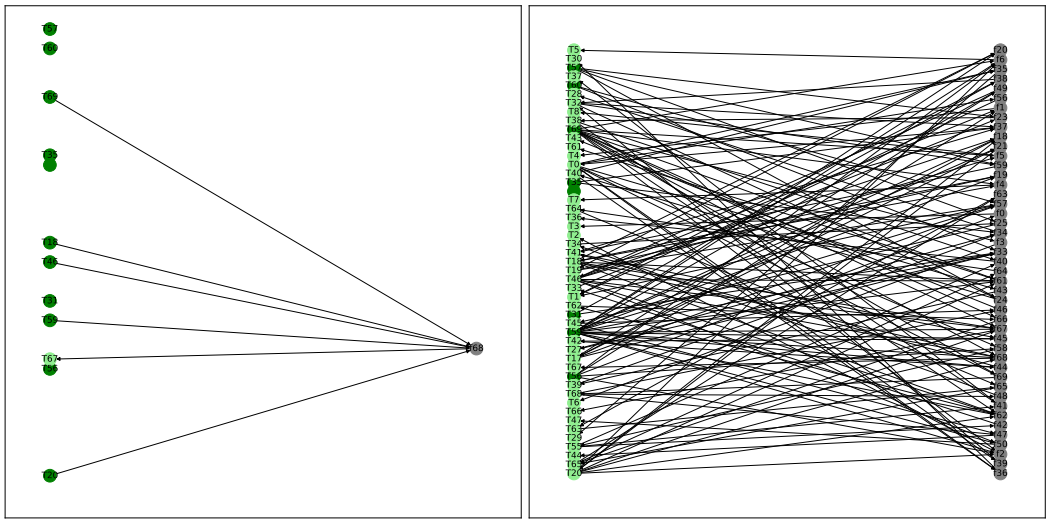

Figure 2: Closure under computability

The left hand side picture shows a first step in the computation of the computable types. The dark green nodes on the left represent the types of the given variables. (in this case describing the YIBS model) We find a first applicable (i.e. all its arguments are given) function (grey node on the right), infer the result type from its type annotation and add it to the set of given types (new light green node). Now we have more arguments and thus more possibly applicable functions.
The right hand side plot shows the result of the recursive application of this procedure to a `CMTVS` instance. Without performing any actual computation we know which results we can compute (30 light green nodes) from the 11 provided variables (dark green), some of them in different ways via intermediate results (as explained in Fig. 3 below). This information is used to automatically add methods to an `CMTVS` instance, so that interactive python environments suggest computable results to the user. In this case 30 new methods appear automatically, including very complex results like the pool specific transient mean age solution for the vegetation carbon sub system witch appears as `get_NumericVegetationCarbonMeanAgeSolutionArray()` It is also the basis for queries, e.g. "for which models can we compute the mean transit time through the the vegetation subsystem?" Note that the applicable functions (in this example 45 represented by the grey nodes) can also be used independently of the `CMTVS` class, explicitly by the user. A lack of the automatic combination would however make it much more difficult to guide a user through the often purely technical conversions to the targeted result and massively increase the amount of necessary handwritten documentation. In this sense `ComputabilityGraphs` can be used as computable documentation or a much simpler API.

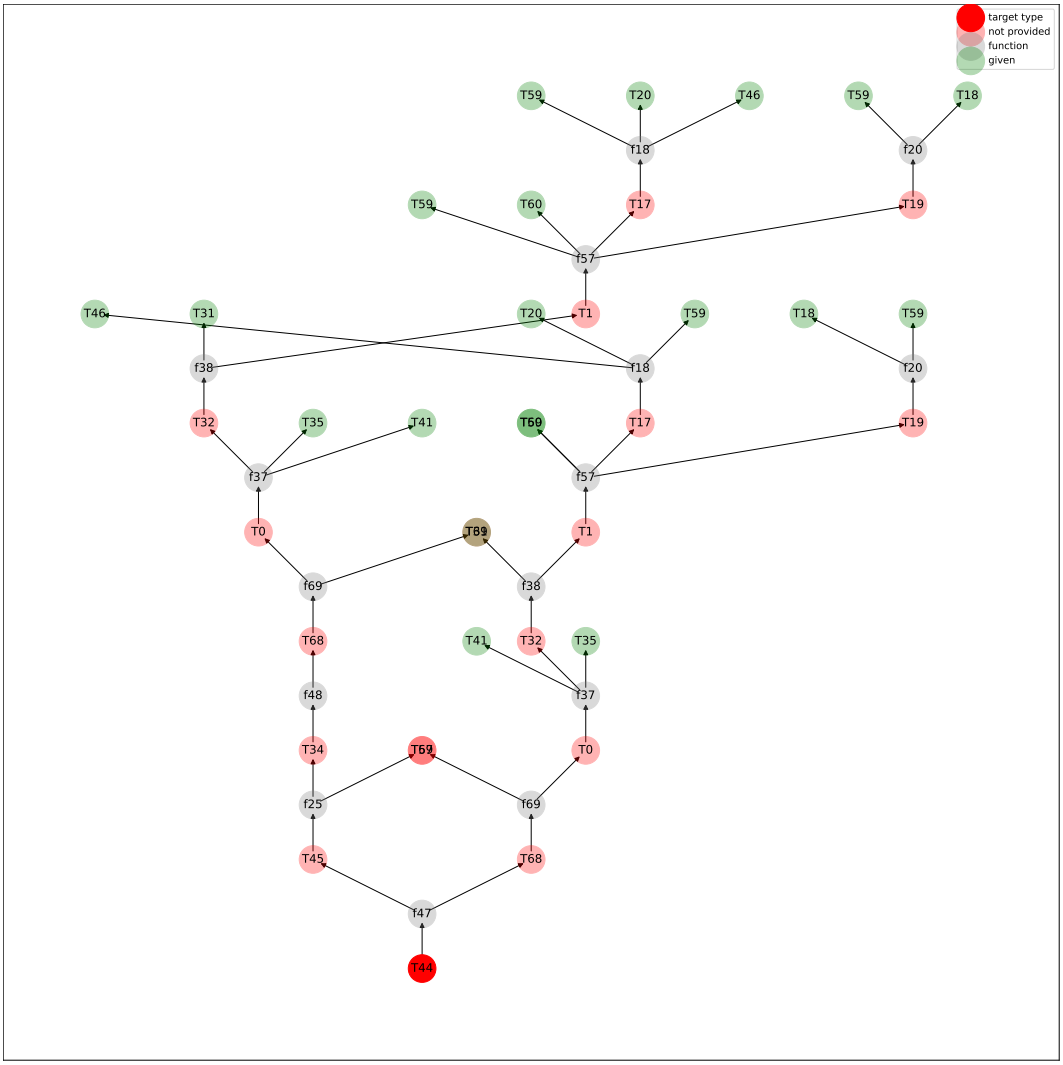

Figure 3: Dependency graph

T0	SmoothModelRun
T1	SmoothReservoirModel
T17	CompartmentalMatrix
T18	InFluxesBySymbol
T19	InputTuple
T20	InternalFluxesBySymbol
T31	NumericParameterization
T32	NumericParameterizedSmoothReservoirModel
T34	NumericParameterizedVegetationCarbonSmoothReservoirModel
T35	NumericSimulationTimes
T41	NumericStartValueArray
T44	NumericVegetationCarbonMeanBackwardTransitTimeSolution
T45	NumericVegetationCarbonStartMeanAgeTuple
T46	OutFluxesBySymbol
T57	StartConditionMaker
T59	StateVariableTuple
T60	TimeSymbol
T68	VegetationCarbonSmoothModelRun
T69	VegetationCarbonStateVariableTuple

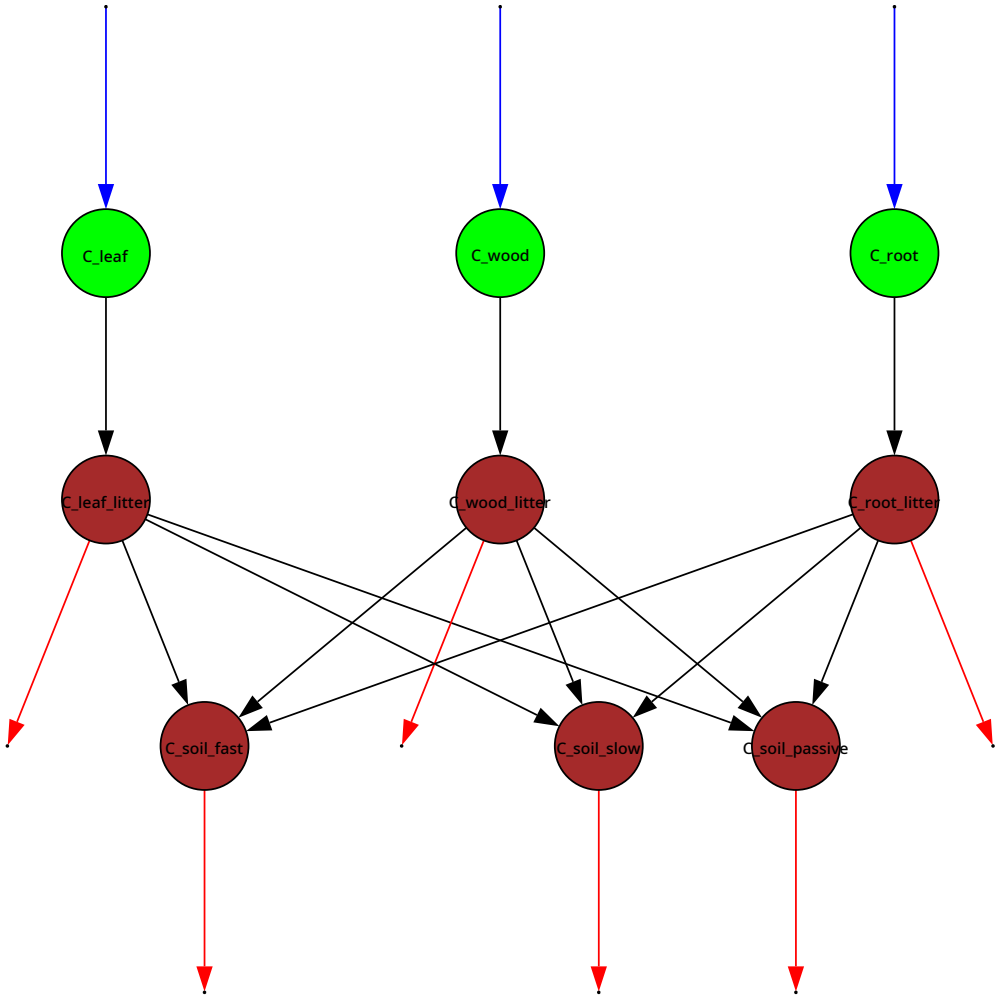

Figure 4: Automatic decomposition into subsystems

Assuming that a compartmental system has been defined symbolically e.g. by providing expressions for input, output and internal fluxes different flow diagrams can be created automatically. The additional information for the decomposition into subsystems just consists of a set of pool names for each subsystem. (Here vegetation and soil part for the visit model reconstruction). For special subsystems like vegetation or soil that frequently occur in Carbon cycle models a declaration of a set of e.g. soil pools has far reaching consequences. Apart from the graph shown here, such a statement immediately makes several new diagnostics available, including matrix descriptions for the vegetation subsystem as in (Ceballos-Núñez et al., 2018) or their soil equivalent, cumulative fluxes between the two subsystems as well as transient age and transit time distributions w.r.t. the vegetation or soil subsystems.

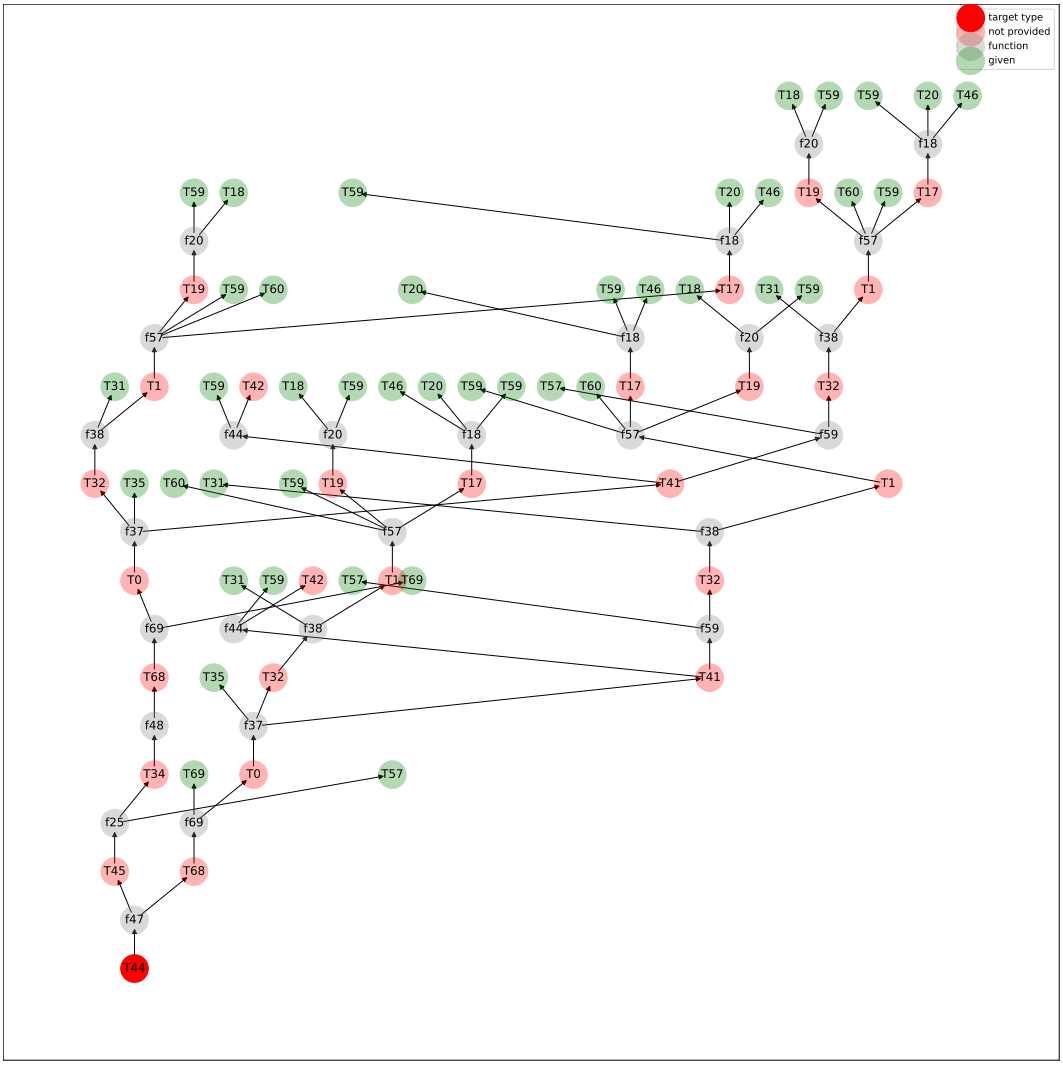

Figure 5: Possible ways to compute the backward transit time for the vegetation subsystem, from the few variables given in the description of the YIBS model (Fig. 2 in dark green).

- Many *labels* (e.g. T59) appear more than once in the graph, while of course a node itself cannot. The reason is that the nodes in this search tree are actually tuples of which we only show the first part in the label. The second part, *which is invisible here*, is a set of types that, in the branch of the node, have already been used and are not allowed anymore. This is a standard technique to transform search *graphs* (with possible circular dependencies) into search *trees* which can be traversed much faster.
- There are actually only nine variables necessary (Fig. 2 dark green) (T18: InFluxesBySymbol, T20: InternalFluxesBySymbol, T31: NumericParameterization, T35: NumericSimulationTimes, T46: OutFluxesBySymbol, T57: StartConditionMaker, T59: StateVariableTuple, T60: TimeSymbol, T69: VegetationCarbonStateVariableTuple) to compute the result.
- There are actually different ways to compute the result. Some of the red nodes (e.g. T41) could be computed by different functions (here f44 and f59) which have different arguments which are compute by different functions . . . so the tree branches out. At the moment the algorithm chooses the first path but we could use this feature to remove hidden duplication in the model description (remove some of the given variables since they are computable) and if it is unavoidable to automatically test its consistency.

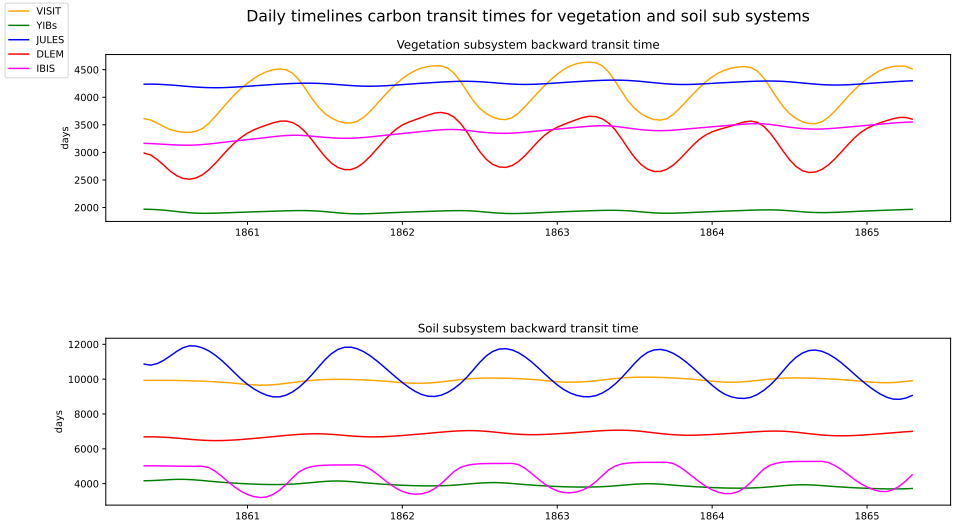

Figure 6: Figure above: Comparing the transient backward transit time through subsystems, across different models.

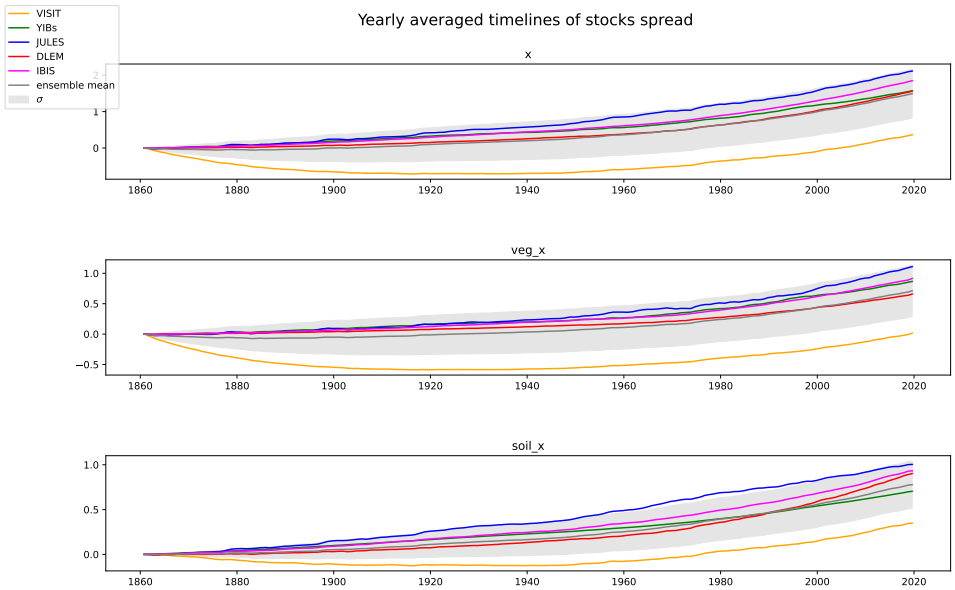

Figure 7: Figure above: Total Carbon stock and subsystem stock development for different models. `bgc_md2` only needs to be told which pools belong to which subsystem.