BMC Bioinformatics

SOFTWARE

Open Access

Constructing and analysing dynamic models with modelbase v1.2.3: a software update

Marvin van Aalst[1], Oliver Ebenhöh[1,2] and Anna Matuszyńska[1,2*]

*Correspondence: anna.matuszynska@hhu.de
[2] CEPLAS - Cluster of Excellence on Plant Sciences, Universitätsstr. 1, 40225 Düsseldorf, Germany
Full list of author information is available at the end of the article

Abstract

Background: Computational mathematical models of biological and biomedical systems have been successfully applied to advance our understanding of various regulatory processes, metabolic fluxes, effects of drug therapies, and disease evolution and transmission. Unfortunately, despite community efforts leading to the development of SBML and the BioModels database, many published models have not been fully exploited, largely due to a lack of proper documentation or the dependence on proprietary software. To facilitate the reuse and further development of systems biology and systems medicine models, an open-source toolbox that makes the overall process of model construction more consistent, understandable, transparent, and reproducible is desired.

Results and discussion: We provide an update on the development of `modelbase`, a free, expandable Python package for constructing and analysing ordinary differential equation-based mathematical models of dynamic systems. It provides intuitive and unified methods to construct and solve these systems. Significantly expanded visualisation methods allow for convenient analysis of the structural and dynamic properties of models. After specifying reaction stoichiometries and rate equations modelbase can automatically assemble the associated system of differential equations. A newly provided library of common kinetic rate laws reduces the repetitiveness of the computer programming code. `modelbase` is also fully compatible with SBML. Previous versions provided functions for the automatic construction of networks for isotope labelling studies. Now, using user-provided label maps, `modelbase` v1.2.3 streamlines the expansion of classic models to their isotope-specific versions. Finally, the library of previously published models implemented in `modelbase` is growing continuously. Ranging from photosynthesis to tumour cell growth to viral infection evolution, all these models are now available in a transparent, reusable and unified format through `modelbase`.

Conclusion: With this new Python software package, which is written in currently one of the most popular programming languages, the user can develop new models and actively profit from the work of others. `modelbase` enables reproducing and replicating models in a consistent, tractable and expandable manner. Moreover, the expansion of models to their isotopic label-specific versions enables simulating label propagation, thus providing quantitative information regarding network topology and metabolic fluxes.

Keywords: Research software, Mathematical modelling, ODE, Metabolic networks, Systems biology, Systems medicine, Biomedical systems, Flux analysis, Labelling, Isotope tracing

Background

Mathematical models are accepted as valuable tools for advancing biological and medical research [1, 2]. In particular, models based on ordinary differential equations (ODEs) have found application in a variety of fields. Most recently, deterministic models simulating the dynamics of infectious diseases gained the interest of the general public during the Covid-19 pandemic. Consequently, a large number of ODE based mathematical models were developed and discussed, even in nonscientific journals [3–5]. Such focus on mathematical modelling is not surprising, because computational models allow for methodical investigations of complex systems under fixed, controlled and reproducible conditions. Hence, the effect of various perturbations of the systems can be inspected systematically in silico.

The model building process itself plays an important role in integrating and systematising vast amounts of available information [6]. Properly designed and verified computational models serve various purposes. They are used to develop hypotheses to guide the design of new research experiments (e.g., in immunology to study lymphoid tissue formation [7]). Models can also support metabolic engineering efforts (e.g., identification of enzymes to enhance essential oil production in peppermint [8]). More recently, models contribute to tailoring medical treatment to individual patient in the spirit of precision medicine (e.g., in oncology [2]). Finally, modelling results guide political decision making and governmental strategies (see the review on the impact of modelling for European Union Policy [9]). Considering their potential impact, models must be openly accessible so that they can be verified and corrected, if necessary.

In many publications, modelling efforts are justified by the emergence of extraordinary amounts of data provided by new experimental techniques. However, arguing for the necessity of model construction only because a certain type or amount of data exists, ignores several important aspects. Computational models are generally a result of months, if not years, of intense research, which involves gathering and sorting information, simplifying numerous details and distilling out the essentials, implementing the mathematical description in computer code, carrying out performance tests, and, finally, validating the simulation results. Our understanding of many phenomena could be deepened if, instead of constructing yet another first-generation model, we could efficiently build on the knowledge that was systematically collected in previously developed models. Moreover, the knowledge generated during the model construction process is often lost, e.g. because the main developer left the research team.

Preservation of the information collected in the form of a computational model has become an important quest in systems biology, and has, to some extent, been addressed by the community. Development of the Systems Biology Markup Language (SBML) [10] for unified communication and storage of biomedical computational models, and the existence of the BioModels repository [11] already ensures the survival of models beyond the academic life of their developers, or the lifetime of the software used to create them. However, a model in SBML format rarely reveals the logic of model construction. The structure of modelbase code promotes consistent and transparent description of the model components (such as reaction rates), hence the logic of construction becomes inherently clear. Such knowledge loss can be prevented by providing simple-to-use toolboxes that enforce a universally readable model construction format.

For these reasons we developed modelbase[12], a Python package that allows the user to easily document the model building process. On the one hand, we defined the core of the model construction process, while on the other hand, the software does not make these definitions too strict, and fully integrates the model construction process into the Python programming language. This differentiates modelbase from many other Python-based modelling tools (such as ScrumPy [13], PySCeS [14], PySB [15] or tellurium [16, 17]) and other mathematical modelling languages (recently reviewed from a software engineering perspective by Schölzel and colleagues [18]). We would in particular like to stress a fundamental difference in the philosophy of modelbase, which distinguishes it from the other Python-based tools. In ScrumPy, PySCeS and tellurium, models are objects that are constructed by either SBML import or by a human-readable string (e.g. the Antimony representation chosen in tellurium [17]), which have methods for their numeric simulations and analysis. However, once constructed, the objects are not designed to be further modified. A modular design of different, but similar models, which all depend on sets of analogous modules, is thus difficult to represent. PySB aims at providing systematic construction methods, adding e.g. 'monomers' and 'rules' how these are converted. However, PySB deliberately ignores and overrides standard Python behaviour, making it difficult to keep multiple models in one namespace. In modelbase, models and simulations are two different types of objects. In analogy to experiments, a model corresponds to the biological entity, such as a cell, whereas a simulation corresponds to a particular experiment that is performed on the entity. A model object can be arbitrarily modified by numerous methods. Typically, a model is systematically constructed by instantiating an empty model object, to which components are added by dedicated methods. In this way, the model construction process remains maximally transparent, is fully integrated into the Python programming language, and is completely reproducible. Flexibility to modify and alter the model structure (incl. parameters) is ensured in this way.

Here we report new features in modelbase v1.2.3, developed over the last two years. We have significantly improved the interface to make model construction easier and more intuitive. The accompanying repository of replicated, published models, available from our GitLab project, has been considerably expanded, and now includes a diverse selection of biomedical models (see Additional file 1: Table 2). This diversity highlights the general applicability of our software. Essentially, every dynamic process that can be described by a system of ODEs can be implemented with modelbase.

Implementation

modelbase is a Python package to facilitate construction and analysis of ODE based mathematical models of biological systems. Version 1.2.3 introduces changes not compatible with the previous official release, version 0.2.5 [12]. All API changes are summarised in the official documentation hosted by ReadTheDocs and the differences between the versions are summarised in Table 1.

Table 1 Key changes between the first published version of `modelbase` [12] and the current update

Functionality	`modelbase` 0.2.5	`modelbase` 1.2.3
Initialization	Model takes only parameters as an argument	Model takes as arguments all other model components as dictionaries
Parameters	Hidden as a private attribute	Replaced with a vanilla dictionary
Derived parameters	No function to calculate from other model parameters	Called on initialization and prior to any numerical operations
Handling of time-dependent reactions	Time given in kwargs	Modifiers argument is introduced
Simulation	Integration via `timeCourse` that takes an array of time points for Simulation	Integration via `simulate` that takes only the endpoint of the simulation, default starting point $t = 0$, otherwise starts where the last simulation ended
Labelling features	Focused on carbon labelling problems	Reference of the word carbon was changed to label
Method to calculate relative label distribution in steady-state	None available	Via `LinearLabelModel`
Scan steady-state concentrations depending on parameter values	None available	Via `parameter_scan`
SBML support	Export of model stoichiometries	Import of models that match the capabilities of `modelbase` and full export of models using `ratelaw`
Metabolic Control Analysis support	None available	A full suite of methods to calculate and plot elasticities via `mca` module
Predefined kinetic laws	None available	Via `ratelaw` module
Plotting support	Time course plots	Phase-plane analysis

The model building process starts by creating a modelling object of the dedicated Python class `Model` and adding to it the chemical compounds of the system. Then, following the intuition of connecting the compounds, the reaction network is constructed by adding the reactions one by one. Each reaction requires stoichiometric coefficients and a kinetic rate law. The latter can be provided either as a custom function or by selecting one from the newly provided library of rate laws. The usage of this library (`ratelaws`) reduces the repetitiveness by avoiding boilerplate code. It requires the user to explicitly define reaction properties, such as directionality. This contributes to a systematic and understandable construction process, following the second guideline from the Zen of Python, the guiding principles for Python's design[1]: "Explicit is better than implicit".

From this, `modelbase` automatically assembles the system of ODEs. It also provides numerous methods to conveniently retrieve information about the model. In particular, the `get_*` methods can be used to inspect all the components of the model, and calculate reaction rates for given concentration values. These functions have multiple variants that return all the common data structures (array [19], dictionary, data frames [20]).

[1] python-c "import this".

After the model building process is completed, simulation and analyses of the model are performed with the `Simulator` class. Currently, we offer interfaces to two integrators to solve stiff and non-stiff ODE systems. Provided the Assimulo package [21] is available, as recommended in our installation guide, `modelbase` will use CVode, a variable-order, variable-step multi-step algorithm. The CVode class provides a direct connection to Sundials [22] which is a powerful industrial solver and robust time integrator, with a high computing performance. If Assimulo is not available, `modelbase` will automatically use the SciPy library [23]. Specifically lsoda will be used to integrate the model, which in our experience resulted in lower computing performance [24]. The whole process of assembling a model has been summarised in Fig. 1.

Metabolic control analysis

Sensitivity analysis provides a theoretical foundation to systematically quantify the effects of small parameter perturbations on global system behaviour. In particular, Metabolic Control Analysis (MCA), initially developed to study metabolic systems, is an important and widely used framework providing quantitative information about the response of the system to perturbations [25, 26]. The new version of `modelbase` has a full suite of methods to calculate elasticities. These can be plotted as a heat-map, giving a clear and intuitive colour-coded visualisation of the results. An example of such visualisation, for a re-implemented toy model of the upper part of glycolysis (Section 3.1.2 [27]), can be found in Fig. 2.

Visualisation support

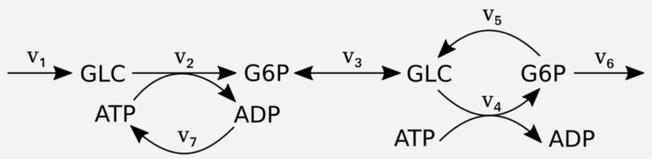
Many of the existing software packages for building computational models restrict the users by providing unmodifiable plotting routines with predefined settings that may not suit their personal preferences. In `modelbase` v1.2.3 plotting functions allow the user to pass optional keyword-arguments (often abbreviated as **kwargs), similar to Tellurium [17]. All plot elements are accessible and available for change, providing a transparent and flexible interface to the commonly used matplotlib library [28]. The easy access functions that visualise the results of simulations were expanded from the previous version. They now include funcitonality to plot selections of compounds or fluxes, phase-plane analysis and the results of MCA. An example of the latter is included in Fig. 2.

Models for isotope tracing

`modelbase` has also been developed to aid the in silico analyses of label propagation during isotopic studies. To simulate the dynamic distribution of isotopes, all possible labelling patterns for all intermediates need to be created. By providing an atom transition map in the form of a list or a tuple, all 2^N isotope-specific versions of a chemical compound are created automatically, where N denotes the number of possibly labelled atoms. Changing the name of the previous function `carbonmap` to `labelmap` in v1.2.3 acknowledges the diversity of possible labelling experiments that can be reproduced with models built using our software.

1. How to build and analyse a model with `modelbase`

We use `modelbase` to simulate a toy model of glycolysis proposed in [27], and presented on the scheme below.



First we import modules and classes from the `modelbase` , `numpy` and `matplotlib` packages

```
from modelbase.ode import Model, LabelModel, LinearLabelModel, Simulator, mca
from modelbase.ode import ratelaws as rl
from modelbase.ode import ratefunctions as rf
import numpy as np
import matplotlib.pyplot as plt
```

and instantiate the `modelbase` object

```
m = Model()
```

Next we add a dictionary of parameters and a list of state variables (compounds):

```
m.add_parameters({"k1":0.25,"k2":1,"k3":1,"k3m":1,"k4":1,"k5":1,"k6":1,"k7":2.5})
m.add_compounds(["GLC", "G6P", "F6P", "FBP", "ATP", "ADP"])
```

In the next step we define reaction rate functions. We use the build-in ratelaws function with predefined kinetic laws. As all reaction rates in the toy model are simple mass action kinetics we add 7 reactions to the model m

```
m.add_reaction(rate_name="v1", function=rf.constant, stoichiometry={"GLC": 1}, parameters=["k1"])
m.add_reaction(rate_name="v2", function=rf.mass_action_2,
    stoichiometry={"GLC": -1, "ATP": -1, "G6P": 1, "ADP": 1}, parameters=["k2"])
m.add_reaction(rate_name="v3", function=rf.reversible_mass_action_1_1,
    stoichiometry={"G6P": -1, "F6P": 1}, parameters=["k3", "k3m"],
    reversible=True)
m.add_reaction(rate_name="v4", function=rf.mass_action_2,
    stoichiometry={"F6P": -1, "ATP": -1, "ADP": 1, "FBP": 1}, parameters=["k4"])
m.add_reaction(rate_name="v5", function=rf.mass_action_1,
    stoichiometry={"FBP": -1, "F6P": 1}, parameters=["k5"])
m.add_reaction(rate_name="v6", function=rf.mass_action_1,
    stoichiometry={"FBP": -1}, parameters=["k6"])

m.add_reaction(rate_name="v7", function=rf.mass_action_1,
    stoichiometry={"ADP": -1, "ATP": 1}, parameters=["k7"])
```

To perform the computation we generate an instance of the Simulator class

```
s = Simulator(m)
```

and provide the dictionary with initial concentrations to start integration

```
y0 = {"GLC": 0, "G6P": 0,"F6P": 0, "FBP": 0,"ATP": 0.5,"ADP": 0.5}
s.initialise(y0)
t, y = s.simulate(100)
```

The results of above simulation will be displayed automatically using the plot() function of the method. Do not forget to label your axes.

```
fig, ax = s.plot()
plt.show()
```

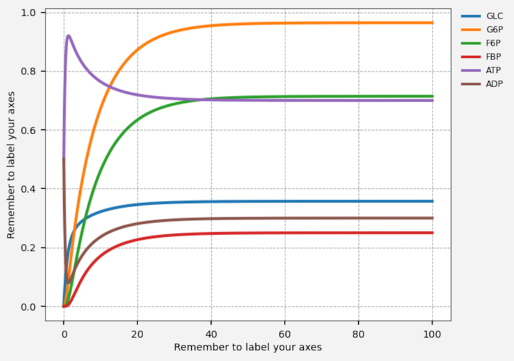
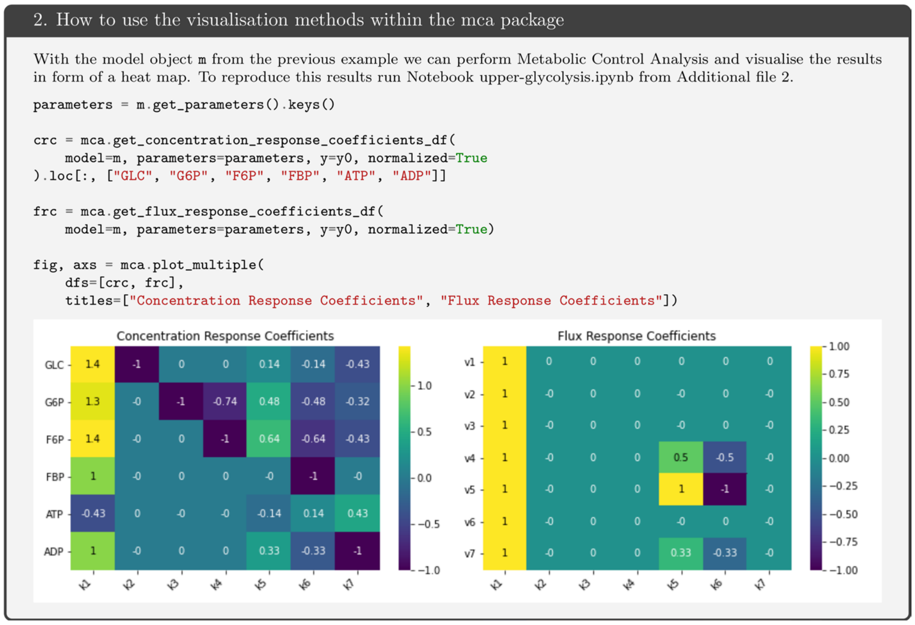


Fig. 1 An example of how to build and analyse a model with `modelbase`

Isotope tracing under stationary conditions

Sokol and Portais derived the theory of dynamic label propagation under the stationary assumption [29]. In steady-state, the space of possible solutions is reduced and the labelling dynamics can be represented by a set of linear differential equations. We have used this theory and implemented an additional class `LinearLabelModel` that



2. How to use the visualisation methods within the mca package

With the model object m from the previous example we can perform Metabolic Control Analysis and visualise the results in form of a heat map. To reproduce this results run Notebook upper-glycolysis.ipynb from Additional file 2.

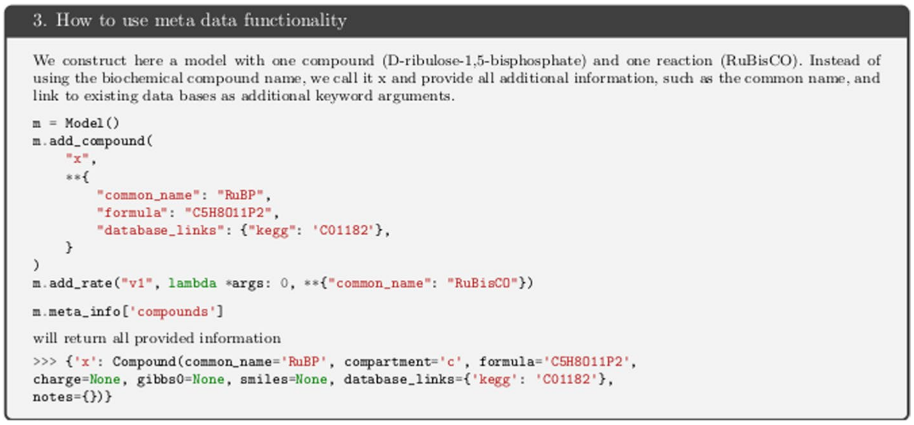
```
parameters = m.get_parameters().keys()

crc = mca.get_concentration_response_coefficients_df(
    model=m, parameters=parameters, y=y0, normalized=True
).loc[:, ["GLC", "G6P", "F6P", "FBP", "ATP", "ADP"]]

frc = mca.get_flux_response_coefficients_df(
    model=m, parameters=parameters, y=y0, normalized=True)

fig, axs = mca.plot_multiple(
    dfs=[crc, frc],
    titles=["Concentration Response Coefficients", "Flux Response Coefficients"])
```

Fig. 2 An example of how to use the visualisation methods within the mca package



3. How to use meta data functionality

We construct here a model with one compound (D-ribulose-1,5-bisphosphate) and one reaction (RuBisCO). Instead of using the biochemical compound name, we call it x and provide all additional information, such as the common name, and link to existing data bases as additional keyword arguments.

```
m = Model()
m.add_compound(
    "x",
    **{
        "common_name": "RuBP",
        "formula": "C5H8O11P2",
        "database_links": {"kegg": 'C01182'},
    }
)
m.add_rate("v1", lambda *args: 0, **{"common_name": "RuBisCO"})
m.meta_info['compounds']
```

will return all provided information

```
>>> {'x': Compound(common_name='RuBP', compartment='c', formula='C5H8O11P2',
charge=None, gibbs0=None, smiles=None, database_links={'kegg': 'C01182'},
notes={})}
```

Fig. 3 An example of how to use metadata functionality

allows rapid calculation of the label propagation given the steady-state concentrations and fluxes of the metabolites [29]. `modelbase` can automatically build the linear label model from user provided label maps. An example of such a model is provided in Fig. 3, where we simulate label propagation in a linear non-reversible pathway, see Fig. 1 in [29] for comparison. The linear label models are constructed using `modelbase` rate laws, and hence can be fully exported as an SBML file.

Model metadata

Many models lose their readability due to the inconsistent, intractable or misguided naming conventions. An example is a model with reactions named v1-v10, without referencing them properly. By providing metadata fields for all `modelbase` objects, the

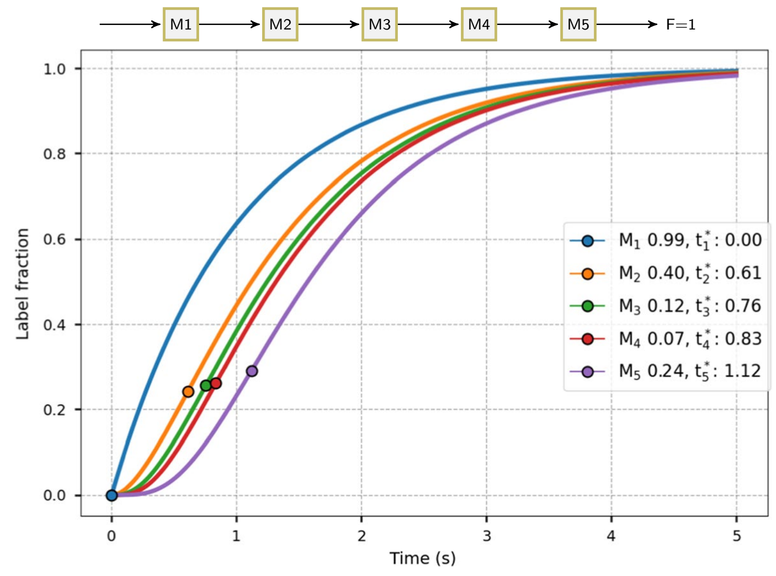


Fig. 4 Labelling curves in a linear non-reversible pathway. Example of label propagation curves for a linear non-reversible pathway of five randomly sized metabolite pools, as proposed in the paper by Sokol and Portais [29]. Circles mark the position at which the first derivative of each labelling curve reaches maximum. In the original paper, this information has been used to analyse the label shock wave (LSW) propagation. To reproduce these results run the label-propagation-2015.ipynb Jupyter Notebook from the Additional file 4: Jupyter Notebook label-propagation-2015.ipynb

user can abbreviate component names in a personally meaningful manner and supply additional annotation information in accordance with recognised standards, such as MIRIAM [30]. An example of how to use metadata functionality is provided in Fig. 4. This interface can also be used to supply additional information, such as the unit of a parameter.

SBML support

In contrast to the previous `modelbase` version, where we only supported the export of stoichiometric models to SBML format, we now support both import and export of kinetic models. The full summary of the SBML concepts supported by `modelbase` is documented in the official SBML test suite, where the output of our tests is stored. Examples where SBML models are imported and exported, using our `build_model_from_sbml` and `write_sbml_model` functions, are supplied in the `modelbase` documentation.

Results and discussion

With the newly implemented changes, `modelbase` is more versatile and user friendly. As argued before, its strength lies in its flexibility and applicability to virtually any biological system with dynamics that can be described using an ODE system. There exist countless mathematical models of biological and biomedical systems derived using ODEs. Many of these models are rarely re-used, at least not to the extent that could be reached, if models were shared in a readable, understandable and reusable way [18]. Our package can be used efficiently both for the development of new models, as well as the

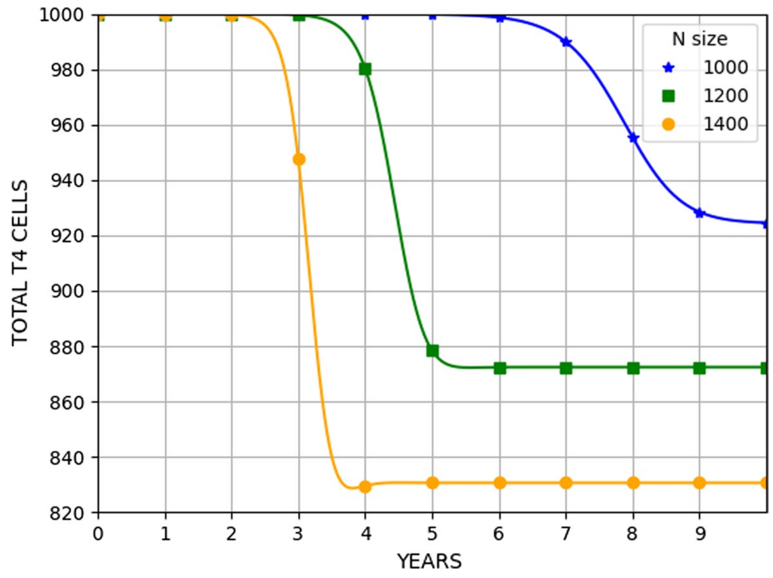


Fig. 5 The total CD4+ T-cell population versus time after the infection. We have reproduced the results from Fig. 4 from the original paper [35] showing the decrease in the overall population of CD4+ T-cell (uninfected + latently infected + actively infected CD4+) over time, depending on the number of infectious particles produced per actively infected cell (N). To reproduce these results run the hiv-t4cell.ipynb Jupyter Notebook from the Additional file 5: Jupyter Notebook hiv-t4cell.ipynb

reconstruction of existing ones. We are confident that `modelbase` will in particular support users with limited modelling experience in re-constructing already published work, serving as a starting point for their further exploration and development. We have previously demonstrated the versatility of `modelbase` by re-implementing mathematical models previously published without the source code: two models of biochemical processes in plants [31, 32], and a model of the non-oxidative pentose phosphate pathway of human erythrocytes [33, 34]. To present how the software can be applied to study medical systems, we used `modelbase` to re-implement various models, not published by our group, and reproduced key results of the original manuscripts. It was beyond our focus to verify the scientific accuracy of the corresponding model assumptions. We selected these examples to show that despite describing different processes, they all share a unified construct. This highlights that by learning how to build a dynamic model with `modelbase`, the user do not learn how to build a one-purpose model, but in fact expands the toolbox to be capable of replicating any given ODE based model. All examples are available as Jupyter notebooks and listed in the Additional file 3: Jupyter Notebook upper-glycolysis.ipynb.

Compartment model for disease evolution

For this paper, we surveyed available computational models and selected a relatively old publication of significant impact, that was published without providing the computational source code, nor details regarding the numerical integration. We chose a four-compartment model of HIV immunology that investigates the interaction of a single virus population with the immune system described only by the CD4$^+$ T cells, commonly known as T helper cells [35]. We implemented the four ODEs describing the dynamics of uninfected (T), latently infected (L), actively infected CD4$^+$ T cells (A), and infectious HIV population (V). In Fig. 5, we reproduce the results from Fig. 4 in the original paper, whereby changing the number of infectious particles produced per actively infected cell (N) we follow the dynamics of the overall T cell population (T+L+A) over a period of 10 years. The model was also used to explore the effect of azidothymidine, an antiretroviral medication, by decreasing the value of N after 3 years by 25% or 75%, mimicking the blocking of the viral replication of HIV. A more detailed description of the time-dependent drug concentration in the body is often achieved with pharmacokinetic models. Mathematical models based on a system of differential equations that link the dosing regimen with the dynamics of a disease are called pharmacokinetic-pharmacodynamic (PK-PD) models [36]. The next example explores how `modelbase` can be used to develop such models.

PK-PD models and precision medicine

Technological advances forced a paradigm shift in many fields, including medicine, making more personalised healthcare not only a possibility but a necessity. A pivotal role in the success of precision medicine will be to correctly determine dosing regimes for drugs [37]. PK-PD models provide a quantitative tool to support this [38]. PK-PD models have proven successful in many fields, including oncology [39], here we use the classical tumour growth model developed by Simeoni and colleagues, originally implemented using the industry-standard software WinNonlin [40]. As the full pharmacokinetic model is not fully described, we reproduced only the highly simplified case, where we assume a single drug administration and investigate the effect of drug potency (k_2) on simulated tumour growth curves. In Fig. 6 we plot the simulation results of the `modelbase` implementation of the system of four ODEs over a period of 18 days, where we systematically changed the value of k_2, assuming a single drug administration on Day 9. With the MCA suite available in our software, we can calculate the response to perturbation of all other system parameters. Such a quantitative description of the system's dynamics to local parameter perturbation provides support for further studies of the rational design of combined drug therapy and the discovery of new drug targets, as described in the review by Cascante and colleagues [41].

Modelling of infectious diseases with SIR models

Finally, compartmental models based on ODE systems have a long history of application in mathematical epidemiology [42]. Many of them, including numerous recent publications studying the spread of coronavirus, are based on the classic epidemic Susceptible-Infected-Recovered (SIR) model, originating from the theories developed by Kermack and McKendrick at the beginning of the last century [43]. One of the most important insights gained from simulating the dynamics of infectious disease is the existence of disease-free or endemic equilibrium, and assessment of its stability [44]. Interestingly, periodic oscillations have been observed for several infectious diseases, including measles, influenza and smallpox [42]. To provide an overview of more `modelbase` functionalities we have implemented a relatively simple SIR model based on the recently published autonomous model for smallpox [45]. We have generated damped oscillations and visualised them using the built-in function `plot_phase_plane` (Fig. 7). In the

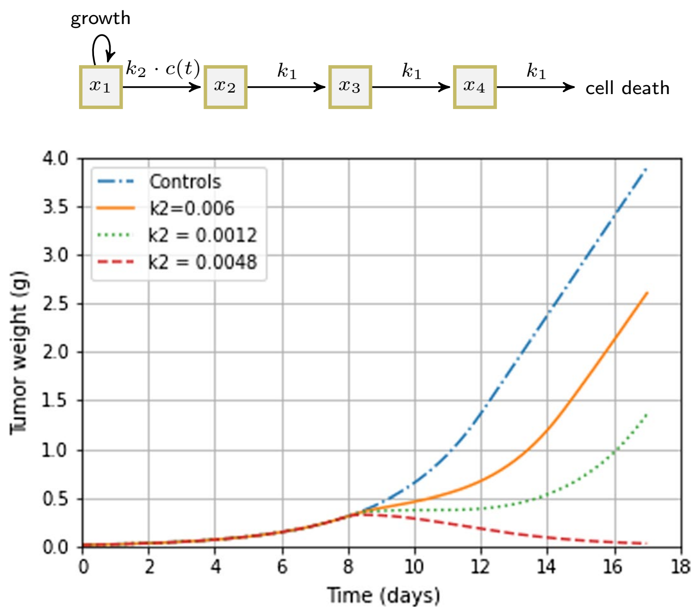


Fig. 6 Compartmental pharmacokinetic-pharmacodynamic model of tumour growth after anticancer therapy. We have reproduced the simplified version of the PK-PD model of tumour growth, where the PK part is reduced to a single input and simulated the effect of drug potency (k_2) on tumour growth curves. The system of four ODEs describing the dynamics of the system visualised on a scheme above is integrated over a period of 18 days. We systematically changed the value of k_2, assuming a single drug administration on Day 9. We have obtained the same results as in Fig. 4 in the original paper [40]. To reproduce these results run the tumour-growth-2004.ipynb Jupyter Notebook from the Additional file 6: Jupyter Notebook tumour-growth-2004.ipynb

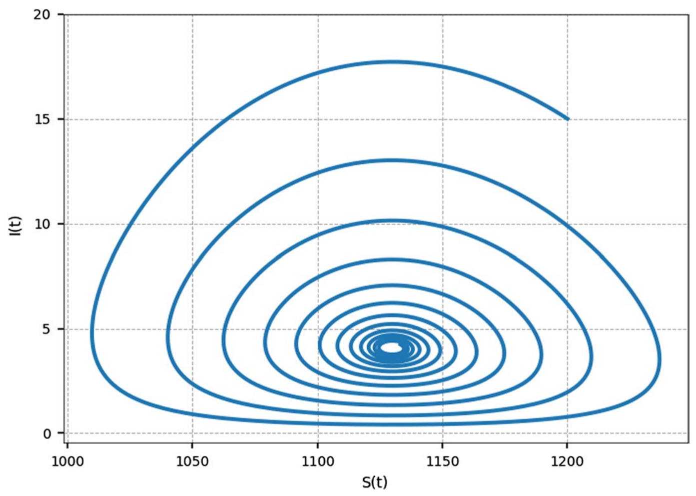


Fig. 7 Sample phase portrait obtained with SIR model with oscillations. SIR model with vital dynamics including birth rate has been adapted based on the autonomous model to simulate periodicity of chickenpox outbreak in Hida, Japan [45]. To reproduce these results run the sir-model.ipynb Jupyter Notebook from the Additional file 7: Jupyter Notebook sir-model.ipynb

accompanying Jupyter notebook we demonstrate using `modelbase`, how simply the SIR model can be built and how to modify it to construct more variants, such as the SEIR (E-exposed) or SIRD (D-deceased) models.

Conclusions

Here, we are presenting an update of our modelling software that simplifies the process of building mathematical models based on ODEs. `modelbase` is fully embedded in the Python programming language. It facilities a systematic construction of new models, and replication of models in a consistent, tractable and expandable manner. As ODEs are a core method to describe the dynamical systems, we hope that our software will serve as the base for deterministic modelling, hence its name. With the smoothed interface and clearer description of how the software can be used for medical purposes, such as simulation of possible drug regimens for precision medicine, we expect to broaden our user community. We envisage that by providing the MCA functionality, users new to mathematical modelling will adopt a working scheme where such sensitivity analyses are an integral part of model development and analysis. The value of sensitivity analyses is demonstrated by considering how the results of such analyses have given rise to new potential targets for drug discovery [41]. We anticipate that the capability of `modelbase` to automatically generate isotopic label-specific models will prove useful in predicting fluxes and label propagation dynamics through various metabolic networks. In emerging fields such as computational oncology, such models will be useful to, e.g., predict the appearance of labels in cancer cells.

Availability and requirements

Project name: `modelbase`
Project home page: https://pypi.org/project/modelbase/
Code repository: https://gitlab.com/qtb-hhu/modelbase-software
Version published: 1.2.3
Date published: 14 Jan 2021
Documentation: https://modelbase.readthedocs.io/en/latest
Operating system(s): Platform independent
Programming language: Python
Other requirements: None
Licence: GNU General Public License (GPL), version 3
Any restrictions to use by non-academics: None.

Abbreviations
MCA: Metabolic control analysis; ODE: Ordinary differential equations; PK-PD: Pharmacokinetic-pharmacodynamic; SBML: Systems biology markup language.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-021-04122-7.

Additional file 1. Table with the list of available models in the repository

Additional file 2. README.md with the Installation instructions

Additional file 3. Jupyter Notebook upper-glycolysis.ipynb: toy model of the upper path of glycolysis asintroduced in [27]

Additional file 4. Jupyter Notebook label-propagation-2015.ipynb: an example of a linear non-reversible path-wayof five randomly sized metabolites and label propagation experiments, as proposed in the paper by Sokol and Portais [29]

Additional file 5. Jupyter Notebook hiv-t4cell.ipynb: a model of the dynamics of HIV infection of CD4+ cells,considering three populations of T cells and free virus, proposed by Perelson, Kirschner and de Boer [35]

Additional file 6. Jupyter Notebook tumour-growth-2004.ipynb: minimal pharmacokinetic-pharmacodynamic(PK-PD)model linking that linking the dosing regimen of an anticancer agent to the tumour growth, proposed bySimeoni and colleagues [40]

Additional file 7. Jupyter Notebook sir-model.ipynb: classic epidemic Susceptible-Infected-Recovered (SIR) model-parameterised as the autonomous model used to simulate periodicity of chickenpox outbreak in Hida, Japan [45]

Acknowledgements
We would like to thank all members of the Institute of Quantitative and Theoretical Biology for their continuous involvement in the software testing and expansion of the repository of novel and replicated computational models.

Authors' contributions
MvA implemented, tested, bug fixed and maintained the `modelbase` software. OE conceptualised `modelbase`. AM specified requirements, designed and tested the implemented changes and provided all biomedical examples used in this paper. All authors have read and approved the manuscript.

Funding
Open Access funding enabled and organized by Projekt DEAL. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germanys Excellence Strategy—EXC-2048/1–project ID 390686111 (OE, AM), EU's Horizon 2020 research and innovation programme under the Grant Agreement 862087 (MvA), and Deutsche Forschungsgemeinschaft Research Grant-project ID 420069095 (AM). The funding bodies did not play any role in the design of the study nor collection, analysis, and interpretation of data nor in writing the manuscript.

Availability of data and materials
All data generated or analysed during this study are included in this published article. Additionally, code documentation can be found here https://modelbase.readthedocs.io/en/latest and more Jupyter Notebooks with teaching materials and models here https://gitlab.com/qtb-hhu/models. If you have any questions regarding `modelbase`, you are very welcome to ask them on our issues page or by contacting any of the authors. It is our mission to enable reproducible science and to help to put the theory into action.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests

Author details
[1] Institute of Quantitative and Theoretical Biology, Heinrich Heine University, Universitätsstr. 1, 40225 Düsseldorf, Germany. [2] CEPLAS - Cluster of Excellence on Plant Sciences, Universitätsstr. 1, 40225 Düsseldorf, Germany.

Received: 9 September 2020 Accepted: 7 April 2021
Published online: 20 April 2021

References

1. Kitano H. Computational systems biology. Nature. 2002;420(6912):206–10. https://doi.org/10.1038/nature01254.
2. Barbolosi D, Ciccolini J, Lacarelle B, Barlési F, André N. Computational oncology-mathematical modelling of drug regimens for precision medicine. Nat Rev Clin Oncol. 2016. https://doi.org/10.1038/nrclinonc.2015.204.
3. Maier BF, Brockmann D. Effective containment explains subexponential growth in recent confirmed COVID-19 cases in China. Science. 2020;368(6492):742–6. https://doi.org/10.1126/science.abb4557.
4. Tang B, Bragazzi NL, Li Q, Tang S, Xiao Y, Wu J. An updated estimation of the risk of transmission of the novel coronavirus (2019-nCov). Infect Dis Modell. 2020;5:248–55. https://doi.org/10.1016/j.idm.2020.02.001.
5. Rocklöv J, Sjödin H, Wilder-Smith A. COVID-19 outbreak on the Diamond Princess cruise ship: estimating the epidemic potential and effectiveness of public health countermeasures. J Travel Med. 2020;27(3):taa030.
6. Brodland GW. How computational models can help unlock biological systems. Semin Cell Dev Biol. 2015. https://doi.org/10.1016/j.semcdb.2015.07.001.
7. Butler JA, Cosgrove J, Alden K, Timmis J, Coles MC. Model-driven experimentation: a new approach to understand mechanisms of tertiary lymphoid tissue formation, function, and therapeutic resolution. Front Immunol. 2017. https://doi.org/10.3389/fimmu.2016.00658.
8. Rios-Estepa R, Lange I, Lee JM, Lange BM. Mathematical modeling-guided evaluation of biochemical, developmental, environmental, and genotypic determinants of essential oil composition and yield in peppermint leaves. Plant Physiol. 2010;152:2105–19. https://doi.org/10.1104/pp.109.152256.
9. Acs S, Ostlaender N, Listorti G, Hradec J, Hardy M, Smits P, Hordijk L. Modelling for EU Policy support: impact assessments. Technical report. Publications Office of the European Union, Luxembourg (2019). https://doi.org/10.2760/748720
10. Keating S, Waltemath D, König M, Zhang F, Dräger A, Chaouiya C, Bergmann F, Finney A, Gillespie C, Helikar T, Hoops S, Malik-Sheriff R, Moodie S, Moraru I, Myers C, Naldi A, Olivier B, Sahle S, Schaff J, Smith L, Swat M, Thieffry D, Watanabe L, Wilkinson D, Blinov M, Begley K, Faeder J, Gómez H, Hamm T, Inagaki Y, Liebermeister W, Lister A, Lucio D, Mjolsness E, Proctor C, Raman K, Rodriguez N, Shaffer C, Shapiro B, Stelling J, Swainston N, Tanimura N, Wagner J, Meier-Schellersheim M, Sauro H, Palsson B, Bolouri H, Kitano H, Funahashi A, Hermjakob H, Doyle J. SBML level 3 Community members: SBML level 3: an extensible format for the exchange and reuse of biological models. Mol Syst Biol. 2020. https://doi.org/10.15252/msb.20199110.
11. Malik-Sheriff RS, Glont M, Nguyen TVN, Tiwari K, Roberts MG, Xavier A, Vu MT, Men J, Maire M, Kananathan S, Fairbanks EL, Meyer JP, Arankalle C, Varusai TM, Knight-Schrijver V, Li L, Dueñas-Roca C, Dass G, Keating SM, Park YM, Buso N, Rodriguez N, Hucka M, Hermjakob H. BioModels-15 years of sharing computational models in life science. Nucleic Acids Res. 2020;48(D1):407–15. https://doi.org/10.1093/nar/gkz1055.
12. Ebenhöh O, Aalst MV, Saadat NP, Nies T, Matuszyńska A. Building mathematical models of biological systems with modelbase. J Open Res Softw. 2018. https://doi.org/10.5334/jors.236.
13. Poolman MG, Scrum PY. Metabolic modelling with python. IEE Proc Syst Biol. 2006;153(5):375–8. https://doi.org/10.1049/ip-syb.
14. Olivier BG, Rohwer JM, Hofmeyr JHS. Modelling cellular systems with PySCeS. Bioinformatics. 2005;21(4):560–1. https://doi.org/10.1093/bioinformatics/bti046.
15. Lopez CF, Muhlich JL, Bachman JA, Sorger PK. Programming biological models in python using PySB. Mol Syst Biol. 2013;9(1):646. https://doi.org/10.1038/msb.2013.1.
16. Choi K, Medley JK, König M, Stocking K, Smith L, Gu S, Sauro HM. Tellurium: an extensible python-based modeling environment for systems and synthetic biology. Biosystems. 2018;171:74–9. https://doi.org/10.1016/j.biosystems.2018.07.006.
17. Medley JK, Choi K, König M, Smith L, Gu S, Hellerstein J, Sealfon SC, Sauro HM. Tellurium notebooks—an environment for reproducible dynamical modeling in systems biology. PLoS Comput Biol. 2018;14(6):1006220. https://doi.org/10.1371/journal.pcbi.1006220.
18. Schölzel C, Blesius V, Ernst G, Dominik A. Characteristics of mathematical modeling languages that facilitate model reuse in systems biology: a software engineering perspective. BioRxiv. 2020. https://doi.org/10.1101/2019.12.16.875260.
19. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, Rio JF, Wiebe M, Peterson P, Gerard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE. Array programming with NumPy. Nature. 2020;585(7825):357–62. https://doi.org/10.1038/s41586-020-2649-2.
20. The pandas development team: pandas-dev/pandas: Pandas (2020). https://doi.org/10.5281/zenodo.3509134.
21. Andersson C, Claus F, Akesson J. ScienceDirect assimulo: a unified framework for ODE solvers. Math Comput Simul. 2015;116:26–43. https://doi.org/10.1016/j.matcom.2015.04.007.
22. Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DANE, Woodward CS. Sundials?: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Softw. 2005;31(3):363–96.
23. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Jarrod Millman K, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey C, Polat I, Feng Y, Moore EW, Van der Plas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P. SciPy 1.0 contributors: SciPy 1.0: fundamental algorithms for scientific computing in python. Nat Methods. 2020;17:261–72. https://doi.org/10.1038/s41592-019-0686-2.
24. Malengier B, Kišon P, Tocknell J, Abert C, Bruckner F, Bisotti MA. ODES: a high level interface to ODE and DAE solvers. J Open Source Softw. 2018;3(22):165. https://doi.org/10.21105/joss.00165.
25. Kacser H, Burns JAAA. The control of flux: 21. Symp Soc Exp Biol. 1973;27:65–104.
26. Heinrich R, Rapoport TA. A linear steady-state treatment of enzymatic chains. General properties, control and effector strength. Eur J Biochem. 1974;42(1):89–95. https://doi.org/10.1111/j.1432-1033.1974.tb03318.x.
27. Klipp E, Liebermeister W, Wierling C, Kowald A, Lehrach H, Herwig R. Systems biology: a textbook. Weinheim: Wiley; 2013.

28. Hunter JD. Matplotlib: a 2d graphics environment. Comput Sci Eng. 2007;9(3):90–5. https://doi.org/10.1109/MCSE.2007.55.
29. Sokol S, Portais J-C. Theoretical basis for dynamic label propagation in stationary metabolic networks under step and periodic inputs. PLoS ONE. 2015;10(12):0144652. https://doi.org/10.1371/journal.pone.0144652.
30. Le Novère N, Finney A, Hucka M, Bhalla US, Campagne F, Collado-Vides J, Crampin EJ, Halstead M, Klipp E, Mendes P, Nielsen P, Sauro H, Shapiro B, Snoep JL, Spence HD, Wanner BL. Minimum information requested in the annotation of biochemical models (MIRIAM). Nat Publ Gr. 2005. https://doi.org/10.1038/nbt1156.
31. Ebenhöh O, Fucile G, Finazzi G, Rochaix JD, Goldschmidt-Clermont M. Short-term acclimation of the photosynthetic electron transfer chain to changing light: a mathematical model. Philos Trans R Soc Lond B Biol Sci. 2014;369(1640):20130223. https://doi.org/10.1098/rstb.2013.0223.
32. Pettersson G, Ryde-Pettersson U. A mathematical model of the Calvin photosynthesis cycle. Eur J Biochem. 1988;175(3):661–72. https://doi.org/10.1111/j.1432-1033.1988.tb14242.x.
33. McIntyre LM, Thorburn DR, Bubb WA, Kuchel PW. Comparison of computer simulations of the F-type and L-type non-oxidative hexose monophosphate shunts with 31P-NMR experimental data from human erythrocytes. Eur J Biochem. 1989;180(2):399–420.
34. Berthon HA, Bubb WA, Kuchel PW. 13C n.m.r. isotopomer and computer-simulation studies of the non-oxidative pentose phosphate pathway of human erythrocytes. Biochem J. 1993;296:379–87. https://doi.org/10.1042/bj2960379.
35. Perelson AS, Kirschner DE, Boer RD. Dynamics of hiv infection of cd4+ t cells. Math Biosci. 1993;114(1):81–125. https://doi.org/10.1016/0025-5564(93)90043-A.
36. Derendorf H, Meibohm B. Modeling of pharmacokinetic/pharmacodynamic (PK/PD) relationships: concepts and perspectives. Pharm Res. 1999;16(2):176–85. https://doi.org/10.1023/A:1011907920641.
37. Lloyd KCK, Khanna C, Hendricks W, Trent J, Kotlikoff M. Precision medicine: an opportunity for a paradigm shift in veterinary medicine HHS public access. J Am Vet Med Assoc. 2016;248(1):45–8. https://doi.org/10.2460/javma.248.1.45.
38. Polasek TM, Shakib S, Rostami-Hodjegan A. Precision dosing in clinical medicine: present and future. Miton Park: Taylor and Francis Ltd.; 2018. https://doi.org/10.1080/17512433.2018.1501271.
39. Koziol JA, Falls TJ, Schnitzer JE. Different ODE models of tumor growth can deliver similar results. BMC Cancer. 2020;20(1):226. https://doi.org/10.1186/s12885-020-6703-0.
40. Simeoni M, Magni P, Cammia C, De Nicolao G, Croci V, Pesenti E, Germani M, Poggesi I, Rocchetti M. Predictive pharmacokinetic-pharmacodynamic modeling of tumor growth kinetics in xenograft models after administration of anticancer agents. Cancer Res. 2004;64(3):1094–101. https://doi.org/10.1158/0008-5472.CAN-03-2524.
41. Cascante M, Boros LG, Comin-Anduix B, de Atauri P, Centelles JJ, Lee P. Metabolic control analysis in drug discovery and disease. Nat Biotechnol. 2002;20:243–9. https://doi.org/10.1038/nbt0302-243.
42. Hethcote HW. The mathematics of infectious diseases. SIAM Rev. 2000;42(4):599–653. https://doi.org/10.1137/S0036144500371907.
43. Kermack WO, McKendrick AG. A contribution to the mathematical theory of epidemics. Proc R Soc Lond Ser A Contain Pap Math Phys Charact. 1927;115(772):700–21. https://doi.org/10.1098/rspa.1927.0118.
44. Wang J, Liu S, Zheng B, Takeuchi Y. Qualitative and bifurcation analysis using an SIR model with a saturated treatment function. Math Comput Modell. 2012;55(3–4):710–22. https://doi.org/10.1016/j.mcm.2011.08.045.
45. Greer M, Saha R, Gogliettino A, Yu C, Zollo-Venecek K. Emergence of oscillations in a simple epidemic model with demographic data. R Soc Open Sci. 2020. https://doi.org/10.1098/rsos.191187.

Publisher's Note