Available online at www.sciencedirect.com

ScienceDirect

Current Opinion in Systems Biology

Programmatic modeling for biological systems

Alexander L. R. Lubbock[1,2] and Carlos F. Lopez[1,2,3]

Abstract

Computational modeling has become an established technique to encode mathematical representations of cellular processes and gain mechanistic insights that drive testable predictions. These models are often constructed using graphical user interfaces or domain-specific languages, with community standards used for interchange. Models undergo steady-state or dynamic analysis, which can include simulation and calibration within a single application, or transfer across various tools. Here, we describe a novel programmatic modeling paradigm, whereby modeling is augmented with software engineering best practices. We focus on Python—a popular programming language with a large scientific package ecosystem. Models can be encoded as programs, adding benefits such as modularity, testing, and automated documentation generators, while still being extensible and exportable to standardized formats for use with external tools if desired. Programmatic modeling is a key technology to enable collaborative model development and enhance dissemination, transparency, and reproducibility.

Addresses

[1]Department of Biochemistry, Vanderbilt University, Nashville, TN 37232, United States

[2]Vanderbilt-Ingram Cancer Center, Vanderbilt University, Nashville, TN 37232, United States

[3]Department of Biomedical Informatics, Vanderbilt University, Nashville, TN 37203, United States

Corresponding author: Lopez, Carlos F (c.lopez@vanderbilt.edu)

Current Opinion in Systems Biology 2021, **27**:100343

This review comes from a themed issue on **Mathematical Modelling (2021)**

Edited by **Stacey Deleria Finley** and **Vassily Hatzimanikatis**

For complete overview of the section, please refer the article collection - Mathematical Modelling (2021)

Available online 24 May 2021

https://doi.org/10.1016/j.coisb.2021.05.004

Keywords

Programmatic modeling, Python, Network models, Parameter optimization, Visualization, Analysis, Model maintenance, Model sharing, Model revision, Model building, Simulation methods.

Introduction

Mathematical modeling of cellular processes for mechanism exploration is becoming widespread [1–6], but challenges remain as to how models should be built, calibrated, analyzed, and interpreted to extract much-needed mechanistic knowledge from experimental data. Historically, methods and techniques from other fields have been directly imported to systems biology with varying success. For example, early interpretations of cellular processes as circuits provided insights about basic regulatory motifs that could explain cellular behaviors [7], and insights from chemical engineering provided the foundations of flux analysis and metabolic modeling [8–11]. Similarly, techniques from chemistry [12], physics [13,14], and various engineering disciplines [15,16] have been used to model cellular processes, but because of their associated spatiotemporal complexity, from femtosecond/nanometer electron transfer reactions to years and meter scales in tumor growth, no established paradigm has emerged to capture the full complexity of a cell. Multiple tools have been developed with differing modeling objectives, fields, and target audiences. For example, COPASI [5], RuleMonkey [17], Simmune [18], and StochSS [19] all provide graphical user interfaces that cater to nonexpert modelers wishing to encode mechanistic representations of biological processes. More abstract approaches, such as BioNetGen [20] and Kappa [21], use a domain-specific language (DSL) to describe and simulate models. However, most tools are self-contained platforms with a small set of included methods and analyses, limiting access to other standalone simulation tools such as StochKit [4], SciML [22], URDME [23], and SmolDyn [24] to name a few. Similarly, optimization techniques ranging from vector-based optimization methods [25,26] to probabilistic-based methods [27,28] exist in yet another isolated domain. Therefore, the current modeling and simulation ecosystem is compartmentalized and fractured, and thus, unification and intercompatibility efforts are sorely needed.

Valuable efforts toward unification have been put forth to create standards for model instantiation, simulation, analysis, and dissemination [29–35]. Of these, the Systems Biology Markup Language (SBML) is perhaps the most successful to date. Although there are many useful models and frameworks that do not use or predate SBML (for a kinetic metabolic modeling review, see the study by Foster et al. [36]), SBML compatibility has emerged as a valuable standard for reproducibility and interchange. Despite these efforts, mathematical modeling for cell biology remains challenging to scale, both vertically

(larger, more complex models) and horizontally (more active collaborators). Although mathematical tools are the obvious way forward to describe cellular processes, the community knowledge base is highly domain specific because of the substantial complexity involved, with some notable exceptions [37].

A novel, more flexible approach to encode knowledge about biological processes as computer programs is slowly emerging and gaining momentum [3,38,39]. In this approach, biological models are no longer static information, but computer code that is executable, is modular, and aggregates community knowledge toward crowd-driven mathematical models. Although computer languages like Lisp [40] and proprietary packages such as MATLAB have been proposed toward this goal, we believe Python provides the largest open-source ecosystem, myriad learning resources, and large applicable base to unify modeling practices in the field. Adopting a programmatic modeling paradigm for systems biology automatically accrues decades of computer science practices including structured documentation, integrated development environments, (model) version control, code-sharing platforms, code testing frameworks, and importantly, literate programming/computational notebook dissemination. Here, we review the recent developments in programming-based approaches for systems biology. The structure of the article is motivated by the model specification, simulation, calibration, analysis, and visualization paradigm/pipeline, commonly practiced in systems biology. To guide the reader, we provide a list of major Python modeling frameworks and their features (Table 1) and useful Python packages and services to support programmatic modeling (Table 2). Throughout the article, we note how this approach could be supplemented and improved by incorporating best practices from software engineering (Figure 1).

Model specification

Traditionally, encoding a model of biochemical reactions would require the user to write each equation by hand, encode these into a solver, and run the simulations [41]. Although this is still common practice for smaller model systems, these lists of equations often lead to a model dead end as the biological context is completely lost in the mathematical representation, which hinders model reuse. Reaction-based modeling formats add one layer of abstraction where the user instead writes chemical reactions of the form $A + B \leftrightarrow C$ in a program-specific notation, and the computer parses this information into a mathematical representation [42]. These DSLs can operate either through a graphical user interface that generates the code in the background or directly through a text editor [20,21,42]. However, signaling pathways often comprise a large number of molecular complexes, which can assemble in multiple orders, leading to a large number of reactions and intermediate species during complex assembly and degradation. Therefore, traditional approaches become unwieldy as model systems become larger, leading to model dead ends. A level of abstraction to alleviate the problem of manual reaction enumeration is presented by rule-based modeling formalisms, whereby reaction rules rather than explicit reactions (or equations) are used to encode the system [3,20,21]. A reaction rule is a template for reaction patterns to be enumerated and instantiated recursively, starting from a defined list of initial species, thereby saving the user time and reducing error-prone

Table 1

List of major modeling frameworks in Python.

Framework	License	Field/Focus	Model format(s)	Programmatic modeling	ODE simulations	Spatial modeling	Stochastic modeling	GPU acceleration	Steady-state analysis
PySB [3]	BSD	Cell signaling	PySB[a] SBML BNGL[b]	✓	✓	✓[c]	✓	✓	✖
Tellurium [39]	Apache	Cell signaling	Antimony[d] SBML	✓[e]	✓	✓[c]	✓	✖	✓
PySCeS [86]	BSD	Metabolic	Custom DSL, SBML	✖	✓	✓[c]	✖	✖	✓
ScrumPy [87]	GPL	Metabolic	Custom DSL	✖	✓	✖	✖	✖	✓
COBRAPy [6]	GPL	Metabolic	COBRAPy SBML	✓	✖	✓[c]	✖	✖	✓

BSD and MIT are permissive software licenses. GPL is a 'copyleft' software license.

[a] Python-embedded, general purpose programming language (GPPL) approach.

[b] BioNetGen language.

[c] Compartment-based only.

[d] Domain-specific language (DSL).

[e] Function support within DSL.

Table 2

List of key tools and services to support programmatic modeling in Python.

Tool or service	License	Notes
Testing		
PyTest	MIT	Testing framework; pytest.org
GitHub Actions	Free[a] service	Continuous Integration; github.com/features/actions
Circle CI	Free[a] service	Continuous Integration; circleci.com
Calibration		
PyBioNetFit [61]	BSD	BNGL and SBML models
PyPESTO [62]	BSD	SBML and PEtab support
PyDREAM [27]	GPL	PySB interface
Analysis and visualization		
Matplotlib	PSF	Plotting library; matplotlib.org
Jupyter Notebooks	BSD	Computational notebooks; jupyter-notebook.readthedocs.io
PyVIPR [73]	MIT	PySB, Tellurium interfaces
Sharing and modification		
GitHub	Free[a] service	Code hosting and collaboration suite; github.com
Sphinx	BSD	Documentation framework; sphinx-doc.org
Readthedocs	Free[a] service	Automated documentation compiler and hosting; readthedocs.io

BSD, MIT, and PSF are permissive software licenses. GPL is a 'copyleft' software license.
[a] Services are free for open-source projects.

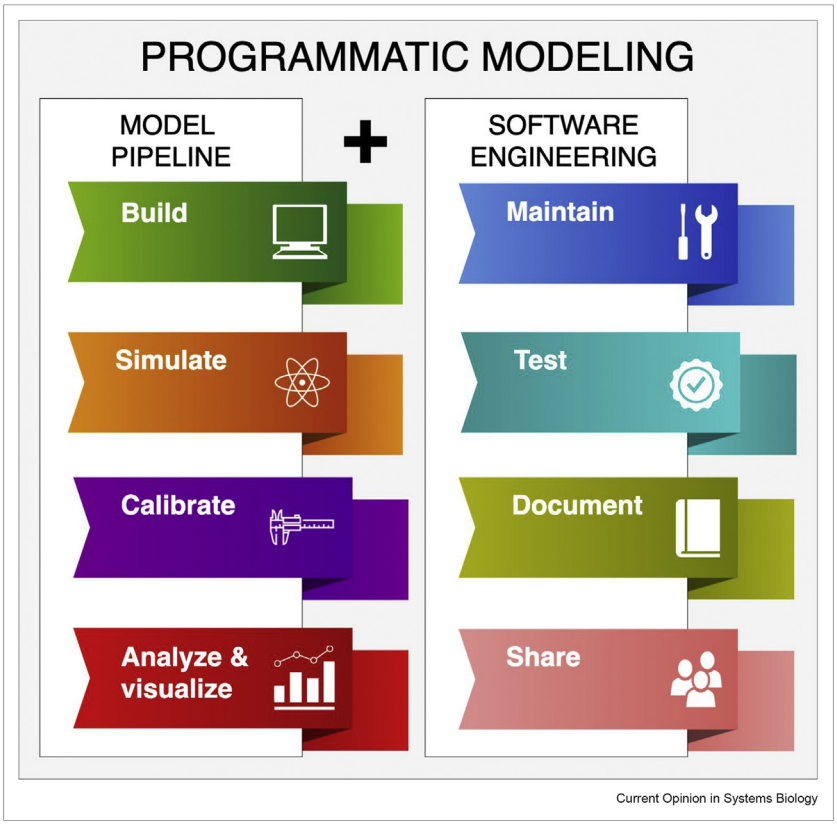
repetition. In rule-based approaches, the reaction center (the relevant molecular components for a given reaction) is separated from the context (attached molecular components which have minimal or no effect on the reaction). These approaches often require a preprocessing step to generate the network of nodes (chemical species) and edges (chemical reactions) from the initial pool of chemical species and a set of reaction rules. However, network-free methodologies have been proposed to bypass the network generation step [43].

Model specification has also been embedded into general purpose programming languages to provide a more powerful approach to biological modeling. In this programmatic modeling paradigm, the model is encoded as executable computer code, thereby offering all the advantages of a full-fledged computer programming language (Figure 2) for all aspects of model manipulation. Modularity, in which a model can be split into smaller, reusable code objects, is perhaps the most useful aspect for cell biology modeling. For example, PySB currently includes a library of 25 macros (small modules or functions) that encode reaction patterns commonly found in biology such as catalysis, inhibition, or oligomerization. From a user perspective, general purpose programming languages have greater integration with integrated development environments than DSLs, thus allowing syntax highlighting and checking, as well as navigation between functions. The model is also inspectable at runtime, allowing searching and filtering of model components. For example, a user could check whether certain species or reactions are present before simulation commences. Currently, the most used modeling frameworks to model nonequilibrium processes that use the programmatic modeling paradigm in Python are PySB [3], written in and using Python, and Tellurium [39], which is written in Python but uses Antimony [42], a DSL with function support, for model specification. For constraint-based metabolic modeling, COBRAPy [6] models are encoded in Python with a broad array of analysis types, including access to the COBRA Toolbox [44].

Model simulation

Model simulation involves numerically solving the model equations to obtain trajectories for dynamically controlled species. Concentrations or molecule counts of chemical species in the model are the most commonly simulated quantities. Integration of systems of ordinary differential equations (ODEs) for deterministic simulations is the most common model simulation approach. Many ODE integrators are available, and the best choice depends on model stiffness, desired integrator tolerances, and other requirements. In Python, a family of integrators is available through SciPy [45] including VODE and LSODA, but many other solvers have been proposed. Other commonly used solver suites include StochKit (stochastic simulation algorithm) [4,46], BioNetGen (CVODE, SSA, tau-leaping algorithm, partition-leaping algorithm) [20], cupSODA (GPU ODE) [47], GPU_SSA (GPU SSA) [48], SUNDIALS [49], and Libroadrunner (CVODE, SSA) [50]. Within the Python ecosystem, PySB provides a simulation class that enables users to use many of these simulation tools or to connect new tools as needed. In addition, users of other Python-based tools such as

Figure 1



The traditional modeling paradigm in systems biology entails model building, simulation, calibration, and analysis (left column), which is carried out with myriad tools and practices. Software engineering practices can add much needed structure to the practice through maintenance, testing, documentation, and sharing paradigms (right column), vetted by the software community.

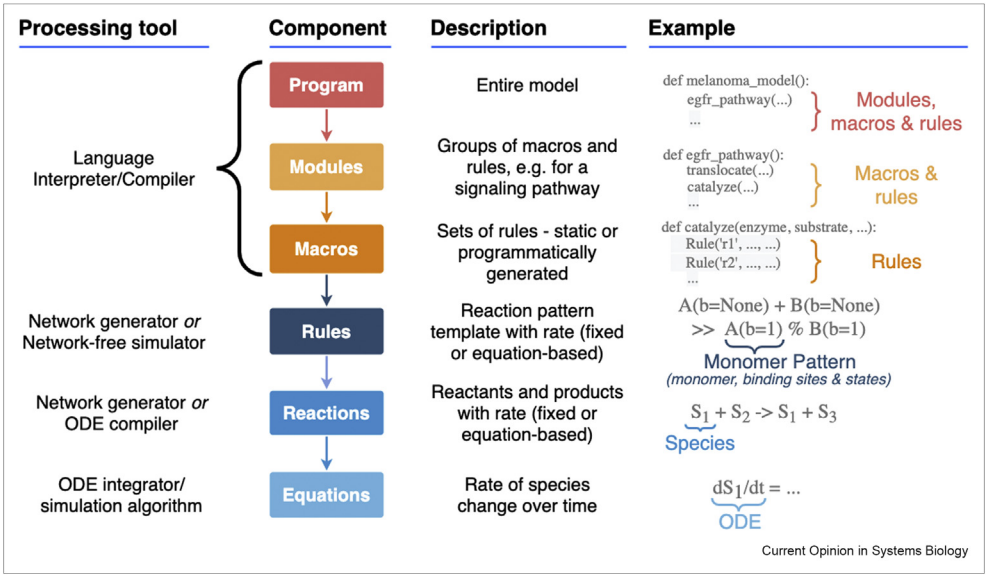
Tellurium can also take advantage of these resources. We note that steady-state simulation tools can also be used, such as flux analysis using COBRApy [6] and pyTFA [51].

Model calibration

Model calibration is the process of adjusting model parameters to match experimental data, also known as parameter estimation/optimization when applied to parametric models. The most common form of model calibration involves a process of running many simulations (thousands to millions or more) and checking the distance between model and experimental data error using an objective function, which gives a measure of the model's simulation 'error' versus experiment; for a review, see the study by Mitra and Hlavacek [52]. Because dynamic data for signaling models are hard to come by, the modeler often only has data for a few species, and thus, model calibration often leaves a model underdetermined—multiple parameter sets fit the data equally well [53]. The concept of parameter 'sloppiness' states that only a few 'stiff' combinations of parameters are important in determining model outcomes, and others are 'sloppy' and have little effect. Thus, an undetermined model can still be useful in predicting biological properties [54]. However, the interpretation of large, underdetermined models in the context of limited data is still up for debate. Lessons from, for example, hydrology and climate modeling have been highly influential toward addressing these issues [27,55,56].

The landscape of model parameters is often envisioned as a multidimensional surface with 'height' representing the objective function, where the (ideally global) minimum or minima (representing the best fit(s)) must be found. SciPy [45], for example, includes gradient descent and simplex-based methods. However, the curse of dimensionality means that local optimization can give far-from-globally optimal results as the number of model parameters increases. Finding the global minimum of a multivariate nonlinear model is NP-hard [57]; however, several methods can make statistically good approximations. Markov chain Monte Carlo sampling methods are among the most popular algorithms [58]. General purpose optimization toolkits for Python include SciPy.optimize [45] and Pyomo [59]. We have

Figure 2



Levels of abstraction in programmatic modeling. Models are composed of modules and macros, which are handled by the programming language interpreter/compiler; rules encode sets of reactions using structured pattern templates; reactions specify biochemical species' transformations, and finally, equations are handled by an ODE integrator or simulation algorithm directly.

found that DEAP [26] provides excellent support for particle swarm optimization and genetic algorithm–based optimization.

Given the dearth of data available for biological model calibration, conditional probability (Bayesian) approaches are gaining traction. These approaches provide a probabilistic interpretation of model parameters [60], including uncertainty quantification, at the cost of increased computer time. However, new GPU-based integrators mitigate this problem. Excellent tools for Bayesian parameter inference include PyDREAM (which can readily take PySB models) [27], PyBioNetFit [61], PyPESTO [62], PyMC3 [63], and PySTAN [64], although popular data-science tools such as TensorFlow [65] and PyTorch [66] also provide Bayesian inference capabilities. ABC-SysBio [67] provides a hybrid solution using approximate Bayesian computation, which has also been applied to metabolic models [68].

Model analysis and visualization

Model analysis and visualization are likely the least developed areas in systems biology as no clear standards have been proposed. In general, modelers explore the chemical species concentration trajectories in their model to infer mechanistic behaviors and properties. Although well-defined methods exist for solving biochemical flux in steady-state systems, calculating flux through nonequilibrium reactions is highly challenging with some notable attempts toward this goal in the literature [13,60]. For visualization, perhaps the most useful tool in Python is matplotlib [69], which provides flexible graphing capabilities. Other Python tools include Seaborn (https://seaborn.pydata.org/), Plotly [70], Bokeh [71], and Mayavi [72]. Network visualization is perhaps the other major area of model analysis that is addressed in various ways in Python. For example, PyVIPR [73] is a visualization tool built on Cytoscape.js [74] for rule- and reaction-based models which animates model dynamics over time, overlaid on a graph. MASSPy [75] also provides some visualization capabilities for metabolic models. Escher [76]visualizes pathways in a web interface and has a Python package. In addition, we note that excellent tools for graph manipulation in Python exist, such as NetworkX [77].

Model sharing and modification

Perhaps the most appealing benefit for the systems biology community from program-based paradigm is the use of literate programming for model and results dissemination. Introduced by Donald Knuth [78], literate programming is a paradigm whereby the code and the document coexist in an interactive format. Jupyter Notebook, a popular format, has been described as 'data scientists' computational notebook of choice' [79]. Jupyter Notebooks allow analyses to be run in a web browser, checked into version control, and include

documentation alongside analyses, in turn improving transparency and reproducibility. We believe that Jupyter notebooks are a highly desirable step forward in systems biology as it greatly contributes to model transparency, revision, and dissemination and should be included in article submissions where computational simulation and analysis are involved.

Programmatic models' code can be managed using existing version control tools. Git has emerged as the *de facto* standard for version control, providing powerful capabilities for decentralized editing, branching, and merging, with online platforms such as GitHub adding a collaborative interface for change management, commenting, and other functions. In PySB, models are Python programs and so can be imported like other Python modules and extended or modified. The code can be inspected, for example, the model can be searched for species or reactions using pattern matching. Tellurium's Antimony language has an import function, but previous model definitions are currently not programmatically searchable or modifiable.

Good documentation can be vital to ensure model reproducibility and interpretability by others. Sphinx (sphinx-doc.org) is a *de facto* documentation standard for Python code, which allows code comments to be compiled into multiple formats including website (HTML) and PDF. The former can be combined with continuous integration (CI), for always up-to-date documentation (readthedocs.io).

Model checking and testing

Complex biochemical models present challenges in both ensuring they are correctly encoded and ensuring their dynamics meet a given specification. In software engineering, it has become common practice to build an accompanying test suite while developing code, which runs the code under scrutiny to test that works as expected. Subtle errors can be introduced as models grow larger. In our opinion, the field should establish minimum standards to ensure software is runnable, is reproducible, and meets basic quality standards [80]. In the context of models-as-programs, unit and integration tests can be borrowed from software engineering to ensure code correctness. Unit tests refer to code which checks the functionality of other, minimal units of code; integration tests check that units work as expected when combined. Python has several frameworks for testing such as PyTests, a popular option with a plugin for Jupyter Notebooks [81]. PySB introduces a framework for testing static properties of rule-based models after network generation, for example, checking that certain species are produced by the reaction network or that certain reactions are present. MEMOTE [34] provides testing and difference-checking capabilities for metabolic models. Using CI, these tests can be run automatically when changes are made and checked into version control and/or on a regular basis. Running tests regularly is recommended because, even if a model itself does not change, changes to software dependencies could lead to unexpected errors. The importance of this is emphasized by a recent review, which found a majority of Jupyter Notebooks were not automatically reproducible, often because of dependency errors [82]. For open-source models, these tests can be run for free using services such as Github Actions, Travis, and Circle CI (Table 2). Finally, we recommend containerization technologies such as Docker [83] and Singularity [84], which bundle model and software dependencies together in a self-contained environment, further aiding reproducibility and cross-platform compatibility.

Conclusions

Python has recently turned 30 years old and is now one of the most popular programming languages in the world. There are many reasons for its success, but a key insight of its creator is that code is read much more often than it is written [85]. The same principle applies to biological models, which emphasizes the importance of clear documentation, transparency of approach, and the separation of model specification from simulation and downstream analysis code. These efforts are central to improving reproducibility, code maintenance, and model extensions, by original authors and third parties. We believe the programmatic modeling approach provides a foundation to address the challenges of growing model complexity via well-tested, documented, modular models. Models should ideally be compatible with a range of simulation engines and analysis types; Python-based frameworks enable extensibility with new methods, and standards such as SBML additionally enable cross-framework compatibility. We additionally envisage large, composite models may require multimodal simulations, for example spatial and nonspatial components, or connected gene regulatory, signaling, and metabolic modules. Further development on programmatic modeling frameworks will be needed to fully address these challenges.

For beginners interested in modeling cell signaling, we recommend either the PySB or Tellurium frameworks, both of which have high-quality documentation and active communities for support. We expect the Python modeling ecosystem will continue to grow and efforts for framework and package interoperability to increase.

Conflict of interest statement

Nothing declared.

Acknowledgements

Funding was provided by the National Science Foundation (1411482 and 1942255 to C.F.L.) and the National Cancer Institute (U01CA215845 to C.F.L.).

References

Papers of particular interest, published within the period of review, have been highlighted as:

* of special interest
** of outstanding interest

1. Albert R, Thakar J: **Boolean modeling: a logic-based dynamic approach for understanding signaling and regulatory networks and for making useful predictions**. *Wiley Interdiscip Rev Syst Biol Med* 2014, **6**:353–369.
2. Ghaffarizadeh A, Heiland R, Friedman SH, Mumenthaler SM, Macklin P, PhysiCell: **An open source physics-based cell simulator for 3-D multicellular systems**. *PLoS Comput Biol* 2018, **14**, e1005991.
3. Lopez CF, Muhlich JL, Bachman JA, Sorger PK: **Programming biological models in Python using PySB**. *Mol Syst Biol* 2013, **9**.
4. Sanft KR, Wu S, Roh M, Fu J, Lim RK, Petzold LR: **StochKit2: software for discrete stochastic simulation of biochemical systems with events**. *Bioinformatics* 2011, **27**:2457–2458.
5. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U: **COPASI—a COmplex PAthway SImulator**. *Bioinformatics* 2006, **22**:3067–3074.
6. Ebrahim A, Lerman JA, Palsson BO, Hyduke DR: **COBRApy: COnstraints-Based Reconstruction and Analysis for Python**. *BMC Syst Biol* 2013, **7**:74.
7. Tyson JJ: **Modeling the cell division cycle: cdc2 and cyclin interactions**. *Proc Natl Acad Sci U S A* 1991, **88**:7328–7332.
8. Kacser H, Burns JA: **The control of flux**. *Symp Soc Exp Biol* 1973, **27**:65–104.
9. Heinrich R, Rapoport SM, Rapoport TA: **Metabolic regulation and mathematical models**. *Prog Biophys Mol Biol* 1977, **32**:1–82.
10. Heinrich R, Rapoport TA: **A linear steady-state treatment of enzymatic chains. General properties, control and effector strength**. *Eur J Biochem* 1974, **42**:89–95.
11. Heinrich R, Reder C: **Metabolic control analysis of relaxation processes**. *J Theor Biol* 1991, **151**:343–350.
12. Vermeulen R, Schymanski EL, Barabási A-L, Miller GW: **The exposome and health: where chemistry meets biology**. *Science* 2020, **367**:392–396.
13. Mallela A, Nariya MK, Deeds EJ: **Crosstalk and ultrasensitivity in protein degradation pathways**. *PLoS Comput Biol* 2020, **16**, e1008492.
14. Tripathi S, Levine H, Jolly MK: **The physics of cellular decision making during epithelial–mesenchymal transition**. *Annu Rev Biophys* 2020, **49**:1–18.
15. Lander AD, Nie Q, Sanchez-Tapia C, Simonyan A, Wan FYM: **Regulatory feedback on receptor and non-receptor synthesis for robust signaling**. *Dev Dyn Off Publ Am Assoc Anat* 2020, **249**:383–409.
16. Clarke R, Tyson JJ, Tan M, Baumann WT, Jin L, Xuan J, Wang Y: **Systems biology: perspectives on multiscale modeling in research on endocrine-related cancers**. *Endocr Relat Cancer* 2019, **26**:R345–R368.
17. Colvin J, Monine MI, Gutenkunst RN, Hlavacek WS, Von Hoff DD, Posner RG: **RuleMonkey: software for stochastic simulation of rule-based models**. *BMC Bioinformatics* 2010, **11**:404.
18. Angermann BR, Klauschen F, Garcia AD, Prustel T, Zhang F, Germain RN, Meier-Schellersheim M: **Computational modeling of cellular signaling processes embedded into dynamic spatial contexts**. *Nat Methods* 2012, **9**:283–289.
19. Drawert B, Hellander A, Bales B, Banerjee D, Bellesia G, Daigle Jr BJ, Douglas G, Gu M, Gupta A, Hellander S, *et al.*: **Stochastic simulation service: bridging the gap between the computational expert and the biologist**. *PLoS Comput Biol* 2016, **12**, e1005220.
20. Harris LA, Hogg JS, Tapia J-J, Sekar JAP, Gupta S, Korsunsky I, Arora A, Barua D, Sheehan RP, Faeder JR: **BioNetGen 2.2: advances in rule-based modeling**. *Bioinformatics* 2016, **32**:3366–3368.
21. Boutillier P, Maasha M, Li X, Medina-Abarca HF, Krivine J, Feret J, Cristescu I, Forbes AG, Fontana W: **The Kappa platform for rule-based modeling**. *Bioinformatics* 2018, **34**:i583–i592.
22. Rackauckas C, Nie Q: **DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia**. *J Open Res Softw* 2017, **5**:15.
23. Drawert B, Trogdon M, Toor S, Petzold L, Hellander A: **MOLNs: a cloud platform for interactive, reproducible, and scalable spatial stochastic computational experiments in systems biology using PyURDME**. *SIAM J Sci Comput Publ Soc Ind Appl Math* 2016, **38**:C179–C202.
24. Andrews SS: **Smoldyn: particle-based simulation with rule-based modeling, improved molecular interaction and a library interface**. *Bioinformatics* 2017, **33**:710–717.
25. Kennedy J, Eberhart R: **Particle swarm optimization**. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4; 1995:1942–1948.
26. Fortin F-A, De Rainville F-M, Gardner M-AG, Parizeau M, Gagné C: **DEAP: evolutionary algorithms made easy**. *J Mach Learn Res* 2012, **13**:2171–2175.
27. Shockley EM, Vrugt JA, Lopez CF: **PyDREAM: high-dimensional parameter inference for biological models in python**. *Bioinformatics* 2018, **34**:695–697.
28. Feroz F, Hobson MP, Bridges M: **MultiNest: an efficient and robust Bayesian inference tool for cosmology and particle physics**. *Mon Not R Astron Soc* 2009, **398**:1601–1614.
29. Keating SM, Waltemath D, König M, Zhang F, Dräger A, Chaouiya C, Bergmann FT, Finney A, Gillespie CS, Helikar T, *et al.*: **SBML Level 3: an extensible format for the exchange and reuse of biological models**. *Mol Syst Biol* 2020, **16**, e9110.
30. ** Zhang F, Smith LP, Blinov ML, Faeder J, Hlavacek WS, Juan Tapia J, Keating SM, Rodriguez N, Dräger A, Harris LA, *et al.*: **Systems biology markup language (SBML) level 3 package: multistate, multicomponent and multicompartment species, version 1, release 2**. *J Integr Bioinform* 2020:17.
SBML Multi is a standard to include multistate, multicomponent species within SBML, bringing closer compatibility with rules-based modeling platforms.
31. Clerx M, Cooling MT, Cooper J, Garny A, Moyle K, Nickerson DP, Nielsen PMF, Sorby H: **CellML 2.0**. *J Integr Bioinform* 2020:17.
32. Agapito G, Pastrello C, Guzzi PH, Jurisica I, Cannataro M: **BioPAX-Parser: parsing and enrichment analysis of BioPAX pathways**. *Bioinformatics* 2020, **36**:4377–4378.
33. Bergmann FT, Cooper J, König M, Moraru I, Nickerson D, Le Novère N, Olivier BG, Sahle S, Smith L, Waltemath D: **Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 3 (L1V3)**. *J Integr Bioinform* 2018, **15**.
34. Lieven C, Beber ME, Olivier BG, Bergmann FT, Ataman M, Babaei P, Bartell JA, Blank LM, Chauhan S, Correia K, *et al.*: **MEMOTE for standardized genome-scale metabolic model testing**. *Nat Biotechnol* 2020, **38**:272–276.
35. * Porubsky VL, Goldberg AP, Rampadarath AK, Nickerson DP, Karr JR, Sauro HM: **Best practices for making reproducible biochemical models**. *Cell Syst* 2020, **11**:109–120.
Porubsky et al. present best practices and discussion toward improved reproducibility of biochemical models.
36. Foster CJ, Wang L, Dinh HV, Suthers PF, Maranas CD: **Building kinetic models for metabolic engineering**. *Curr Opin Biotechnol* 2021, **67**:35–41.
37. * Szigeti B, Roth YD, Sekar JAP, Goldberg AP, Pochiraju SC, Karr JR: **A blueprint for human whole-cell modeling**. *Curr Opin Syst Biol* 2018, **7**:8–15.
Szigeti et al. propose a roadmap toward whole-cell dynamical models, including discussion on technology and standards development.

38. Gyori BM, Bachman JA, Subramanian K, Muhlich JL, Galescu L, Sorger PK: **From word models to executable models of signaling networks using automated assembly**. *Mol Syst Biol* 2017, **13**:954.

39. Choi K, Medley JK, Cannistra C, Konig M, Smith L, Stocking K, Sauro HM: **Tellurium: a Python based modeling and reproducibility platform for systems biology**. *bioRxiv* 2016, https://doi.org/10.1101/054601.

40. Mallavarapu A, Thomson M, Ullian B, Gunawardena J: **Programming with models: modularity and abstraction provide powerful capabilities for systems biology**. *J R Soc Interface* 2009, **6**:257–270.

41. Chen WW, Schoeberl B, Jasper PJ, Niepel M, Nielsen UB, Lauffenburger DA, Sorger PK: **Input–output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data**. *Mol Syst Biol* 2009, **5**.

42. Smith LP, Bergmann FT, Chandran D, Sauro HM: **Antimony: a modular model definition language**. *Bioinformatics* 2009, **25**: 2452–2454.

43. Sneddon MW, Faeder JR, Emonet T: **Efficient modeling, simulation and coarse-graining of biological complexity with NFsim**. *Nat Methods* 2011, **8**:177–183.

44. Heirendt L, Arreckx S, Pfau T, Mendoza SN, Richelle A, Heinken A, Haraldsdóttir HS, Wachowiak J, Keating SM, Vlasov V, *et al.*: **Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0**. *Nat Protoc* 2019, **14**:639–702.

45. ** Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, *et al.*: **SciPy 1.0: fundamental algorithms for scientific computing in Python**. *Nat Methods* 2020, **17**:261–272.
SciPy 1.0 is an open-source library at the core of the scientific Python ecosystem, including optimization, integration, interpolation, matrix algebra and much more.

46. Gillespie DT: **Stochastic simulation of chemical kinetics**. *Annu Rev Phys Chem* 2007, **58**:35–55.

47. Harris LA, Nobile MS, Pino JC, Lubbock ALR, Besozzi D, Mauri G, Cazzaniga P, Lopez CF: **GPU-powered model analysis with PySB/cupSODA**. *Bioinformatics* 2017, **33**:3492–3494.

48. Pino JC, Prugger M, Lubbock ALR, Harris LA, Lopez CF: **Accelerated simulations of chemical reaction systems using the stochastic simulation algorithm on GPUs**. *bioRxiv* 2020, https://doi.org/10.1101/2020.02.14.948612.

49. Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS: **SUNDIALS: suite of nonlinear and differential/algebraic equation solvers**. *ACM Trans Math Softw* 2005, **31**:363–396.

50. Somogyi ET, Bouteiller J-M, Glazier JA, König M, Medley JK, Swat MH, Sauro HM: **libRoadRunner: a high performance SBML simulation and analysis library**. *Bioinformatics* 2015, **31**: 3315–3321.

51. Salvy P, Fengos G, Ataman M, Pathier T, Soh KC, Hatzimanikatis V: **pyTFA and matTFA: a Python package and a Matlab toolbox for thermodynamics-based flux analysis**. *Bioinformatics* 2019, **35**:167–169.

52. Mitra ED, Hlavacek WS: **Parameter estimation and uncertainty quantification for systems biology models**. *Curr Opin Syst Biol* 2019, **18**:9–18.

53. Daniels BC, Chen Y-J, Sethna JP, Gutenkunst RN, Myers CR: **Sloppiness, robustness, and evolvability in systems biology**. *Curr Opin Biotechnol* 2008, **19**:389–395.

54. Gutenkunst RN, Waterfall JJ, Casey FP, Brown KS, Myers CR, Sethna JP: **Universally sloppy parameter sensitivities in systems biology models**. *PLoS Comput Biol* 2007, **3**:e189.

55. Kochen MA, Lopez CF: **A probabilistic approach to explore signal execution mechanisms with limited experimental data**. *Front Genet* 2020:11.

56. Vrugt JA, ter Braak CJF, Clark MP, Hyman JM, Robinson BA: **Treatment of input uncertainty in hydrologic modeling: doing hydrology backward with Markov chain Monte Carlo simulation**. *Water Resour Res* 2008, **44**. n/a–n/a.

57. Floudas CA, Pardalos PM: *State of the art in global optimization: computational methods and applications*. Springer Science & Business Media; 2013.

58. Valderrama-Bahamóndez GI, Fröhlich H: **MCMC techniques for parameter estimation of ODE based models in systems biology**. *Front Appl Math Stat* 2019, **5**.

59. Hart WE: **Python optimization modeling objects (Pyomo)**. In *Operations Research and Cyber-Infrastructure*. Edited by Chinneck JW, Kristjansson B, Saltzman MJ, Springer US; 2009: 3–19.

60. * Shockley EM, Rouzer CA, Marnett LJ, Deeds EJ, Lopez CF: **Signal integration and information transfer in an allosterically regulated network**. *Npj Syst Biol Appl* 2019, **5**:1–9.
Shockley et al. present a Python implementation of the DREAM algorithm, an efficient Monte Carlo method for parameter estimation, with PySB compatibility compatible with High-Performance Computing architectures.

61. * Mitra ED, Suderman R, Colvin J, Ionkov A, Hu A, Sauro HM, Posner RG, Hlavacek WS: **PyBioNetFit and the biological property specification language**. *iScience* 2019, **19**: 1012–1036.
PyBioNetFit is a tool for model parameter estimation, uncertainty characterization, and agreement with experimental data.

62. Schmiester L, Weindl D, Hasenauer J: **Efficient gradient-based parameter estimation for dynamic models using qualitative data**. *bioRxiv* 2021, https://doi.org/10.1101/2021.02.06.430039.

63. Salvatier J, Wiecki TV, Fonnesbeck C: **Probabilistic programming in Python using PyMC3**. *PeerJ Comput Sci* 2016, **2**:e55.

64. Van Hoey S, van der Kwast J, Nopens I, Seuntjens P: **Python package for model STructure ANalysis (pySTAN)** 2013, **15**. EGU2013-10059.

65. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, *et al.*: *TensorFlow: a system for large-scale machine learning*. 2016:265–283.

66. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, *et al.*: *PyTorch: an imperative style*. High-Performance Deep Learning Library; 2019.

67. Liepe J, Barnes C, Cule E, Erguler K, Kirk P, Toni T, Stumpf MPH: **ABC-SysBio—approximate Bayesian computation in Python with GPU support**. *Bioinformatics* 2010, **26**:1797–1799.

68. Saa PA, Nielsen LK: **Construction of feasible and accurate kinetic models of metabolism: a Bayesian approach**. *Sci Rep* 2016, **6**:29635.

69. Hunter JD: **Matplotlib: a 2D graphics environment**. *Comput Sci Eng* 2007, **9**:90–95.

70. Plotly Technologies Inc: *Collaborative data science*. 2015.

71. Jolly K: *Hands-on data visualization with Bokeh: interactive web plotting for Python using Bokeh*. Packt Publishing Ltd; 2018.

72. Ramachandran P, Varoquaux G: **Mayavi: 3D visualization of scientific data**. *Comput Sci Eng* 2011, **13**:40–51.

73. * Ortega OO, Lopez CF: **Interactive multiresolution visualization of cellular network processes**. *iScience* 2020, **23**: 100748.
PyVIPR is a visualization tool that aims to address network visualization problems with automated community detection algorithms.

74. Franz M, Lopes CT, Huck G, Dong Y, Sumer O, Bader GD: **Cytoscape.js: a graph theory library for visualisation and analysis**. *Bioinformatics* 2016, **32**:309–311.

75. Haiman ZB, Zielinski DC, Koike Y, Yurkovich JT, Palsson BO: **MASSpy: building, simulating, and visualizing dynamic biological models in Python using mass action kinetics**. *PLoS Comput Biol* 2021, **17**, e1008208.

76. King ZA, Dräger A, Ebrahim A, Sonnenschein N, Lewis NE, Palsson BO, Escher: **A web application for building,**

sharing, and embedding data-rich visualizations of biological pathways. *PLoS Comput Biol* 2015, **11**, e1004321.

77. Hagberg A, Swart P, S Chult D: *Exploring network structure, dynamics, and function using networkx*. Los Alamos, NM (United States): Los Alamos National Lab. (LANL); 2008.

78. Knuth DE: **Literate programming**. *Comput J* 1984, **27**:97–111.

79. Perkel JM: **Why Jupyter is data scientists' computational notebook of choice**. *Nature* 2018, **563**:145–146.

80. Lubbock ALR: **Accredit scientific software for sustainability**. *Nature* 2019, **572**:586.

81. Fangohr H, Fauske V, Kluyver T, Albert M, Laslett O, Cortés-Ortuño D, Beg M, Ragan-Kelly M: *Testing with Jupyter notebooks: NoteBook VALidation (nbval) plug-in for pytest*. 2020.

82. ** Pimentel JF, Murta L, Braganholo V, Freire J: **A large-scale study about quality and reproducibility of Jupyter notebooks**. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*; 2019:507–517.
Pimentel et al. conduct a large scale reproducibility analysis of Jupyter Notebooks. They identify common issues affecting reproducibility and suggest best practices.

83. Boettiger C: **An introduction to Docker for reproducible research**. *ACM SIGOPS Oper Syst Rev* 2015, **49**:71–79.

84. Kurtzer GM, Sochat V, Bauer MW: **Singularity: scientific containers for mobility of compute**. *PLoS One* 2017, **12**, e0177459.

85. PEP 8 – Style Guide for Python Code. Python.org. Retrieved 16th April 2021, https://www.python.org/dev/peps/pep-0008.

86. Olivier BG, Rohwer JM, Hofmeyr J-HS: **Modelling cellular systems with PySCeS**. *Bioinformatics* 2005, **21**:560–561.

87. Poolman MG: **ScrumPy: metabolic modelling with Python**. *Syst Biol* 2006, **153**:375–378.