SBMLtoODEjax: efficient simulation and optimization of ODE SBML models in JAX

Mayalen Etcheverry[1,2,*] Michael Levin[3] Clément Moulin-Frier[1]
Pierre-Yves Oudeyer[1]

[1] INRIA, University of Bordeaux, Talence 33405, France
[2] Poietis, Pessac 33600, France
[3] Allen Discovery Center, Tufts University, Medford, MA, USA

[*] Correspondence: Mayalen Etcheverry <mayalen.etcheverry@inria.fr>

Keywords: SBML, Biological Network Analysis, Python, JAX, high performance computing, parallel computing

Summary

Developing methods to explore, predict and control the dynamic behavior of biological systems, from protein pathways to complex cellular processes, is an essential frontier of research for bioengineering and biomedicine [1]. Thus, significant effort has gone in computational inference and mathematical modeling of biological systems [2], [3]. This effort has resulted in the development of large collections of publicly-available models, typically stored and exchanged on online platforms (such as the BioModels Database [4], [5]) using the Systems Biology Markup Language (SBML), a standard format for representing mathematical models of biological systems [6], [7].

SBMLtoODEjax is a lightweight library that allows to automatically parse and convert SBML models into python models written end-to-end in JAX, a high-performance numerical computing library with automatic differentiation capabilities [8]. SBMLtoODEjax is targeted at researchers that aim to incorporate SBML-specified ordinary differential equation (ODE) models into their python projects and machine learning pipelines, in order to perform efficient numerical simulation and optimization with only a few lines of code. Taking advantage of JAX's core transformation features, one can easily boost the speed of ODE models time-course simulations and perform efficient search and optimization by running simulations in parallel and/or using automatic differentiation to find derivatives. SBMLtoODEjax extends SBMLtoODEpy, a python library developed in 2019 for converting SBML files into python files written in Numpy/Scipy [9]. The chosen conventions for the generated variables and modules are slightly different from the standard SBML conventions (used in the SBMLtoODEpy library) with the aim to accommodate for more flexible manipulations while preserving JAX-like functional programming style.

SBMLtoODEjax is available at https://github.com/flowersteam/sbmltoodejax.

Statement of Need

Despite the wealth of available SBML models, scientists still lack an in-depth understanding of the range of possible behaviors that these models can exhibit under different initial data and environmental stimuli, and lack effective ways to search and optimize those behaviors via external interventions. Except for a subset of simple networks where system behavior and response to stimuli can be well understood analytically (or with exhaustive enumeration methods), onerous sampling of the parameter space and time-consuming numerical simulations are often needed which remains a major roadblock for progress in biological network analysis.

On the other hand, recent progress in machine learning (ML) has led to the development of novel computational tools that leverage high-performance computation, parallel execution and differentiable programming and that promise to accelerate research across multiple areas of science, including biological network analysis [10] and applications in drug discovery and molecular medicine [11], [12]. However, to our knowledge, there is no software tool that allows seamless integration of existing mathematical models of cellular molecular pathways (SBML files constructed by biologists) with ML-supported pipelines and programming frameworks. Whereas there exists many software tools for manipulation and numerical simulation of SBML models, they typically rely either on specialized simulation platforms limiting the flexibility for customization and scripting (such as COPASI [13], [14], Virtual Cell [15], [16] and Cell Designer [17], [18]) or provide scripting interfaces in Python or Matlab but rely on backend engines that do not support hardware acceleration or automatic differentiation (like Tellurium [19], [20] and SBMLtoODEpy [9] python packages, or the Systems Biology Format Converter (SBFC) which generates MATLAB and OCTAVE code [21]).

SBMLtoODEjax seeks to bridge that gap by bringing SBML simulation to the JAX ecosystem, a thriving community of JAX libraries that aim to accelerate research in machine learning and beyond, with diverse applications spanning molecular dynamics [22], protein engineering [23], quantum physics [24], cosmology [25], ocean modeling [26], photovoltaic research [27], acoustic simulations [28] and fluid dynamics [29]. SBMLtoODEjax aims to integrate this ecosystem and provide tools to accelerate research in biological network analysis.

Why use SBMLtoODEjax?

Simplicity and extensibility SBMLtoODEjax retains the simplicity of the SBMLtoODEPy library to facilitate incorporation and refactoring of the ODE models into one's own python projects. As shown in Figure 1, with only a few lines of python code one can load and simulate existing SBML files.

JAX-friendly The generated python models are tailored to take advantage of JAX main features. Model rollouts use `jit` transformation and `scan` primitive to reduce compilation and execution time of the recursive ODE integration steps, which is particularly useful when running large numbers of steps (long reaction times). Models also inherit from the Equinox module abstraction [31] and are registered as PyTree containers, which facilitates the application of JAX core transformations to any SBMLtoODEjax object.

Efficiency simulation and optimization The application of JAX core transformations, such as just-in-time compilation (`jit`), automatic vectorization (`vmap`) and automatic differentiation (`grad`), to the generated models make it very easy (and seamless) to efficiently run simulations in parallel. For instance, as shown in Figure 2, with only a few lines of python code one can vectorize calls to model rollout and perform batched computations efficiently, which is particularly useful

```python
from sbmltoodejax.utils import load_biomodel

# load and simulate model
model, _, _, _ = load_biomodel(10)
n_secs = 150*60
n_steps = int(n_secs / model.deltaT)
ys, ws, ts = model(n_steps)

# plot time-course simulation
...
```

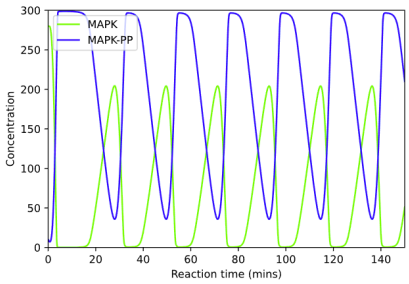

Figure 1: Example code (left) and output snapshot (right) reproducing original simulation results of Kholodenko 2000's paper [30] hosted on BioModels website.

when considering large batch sizes.

```python
# vector of initial conditions
batched_y0 = ...

# batch model
batched_model = vmap(model,
                     in_axes=(None, 0),
                     out_axes=(0, 0, None))

# run simulation in batch mode
batched_ys, batched_ws, ts = batched_model(n_steps,
                                           batched_y0)
```

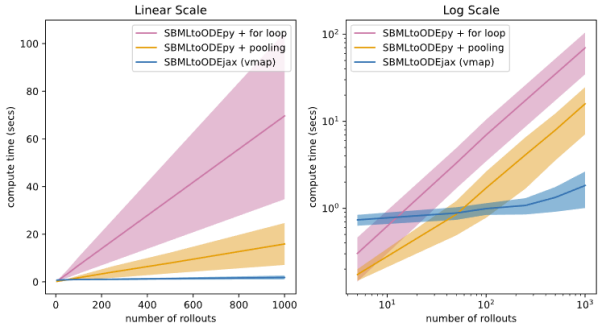

Figure 2: (left) Example code to vectorize calls to model rollout (right) Results of a (rudimentary) benchmark comparing the average simulation time of models implemented with SBMLtoODEpy versus SBMLtoODEjax (for different number of rollouts i.e. batch size). For additional details on the comparison, please refer to our Benchmarking notebook.

As shown in Figure 3, SBMLtoODEjax models can also be integrated within Optax pipelines, a gradient processing and optimization library for JAX [32], allowing to optimize model parameters and/or external interventions with stochastic gradient descent.

Altogether, the parallel execution capabilities and the differentiability of the generated models opens interesting possibilities to design and optimize intervention strategies.

Current limitations SBMLtoODEjax is still in its early phase and does not yet handle all possible cases of SBML files, but we welcome contributions and aim to handle more cases in future releases. Similarly, SBMLtoODEjax only integrates one ODE solver for now (`jax.experimental.odeint`), but could benefit from more [33]. Finally, whereas SBMLtoODEjax is to our knowledge the first software tool enabling gradient backpropagation through the SBML model rollout, applying it in practice can be hard and other optimization methods, such as evolutionary strategies, might be more adapted.

Documentation Please refer to https://developmentalsystems.org/sbmltoodejax/ for additional details on SBMLtoODEjax's main design principles, advantages and limitations, and API docs as well as for various hands-on tutorials for loading and simulating biomodels, parallel execution and gradient descent.

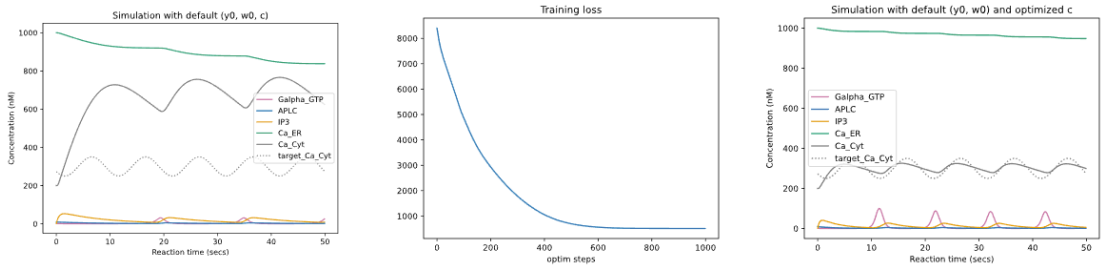

Figure 3: (left) Default simulation results of biomodel #145 which models ATP-induced intracellular calcium oscillations, and (arbitrary) target sine-wave pattern for Ca_Cyt concentration. (middle) Training loss obtained when running the Optax optimization loop, with Adam optimizer, over the model kinematic parameters c. (right) Simulation results obtained after optimization. The full example is available at our Gradient Descent tutorial.

Software requirements

SBMLtoODEjax is developed under the MIT license and available on `PyPI` via `pip install sbmltoodejax`. It is written on top of SBMLtoODEpy [9], JAX (cpu) [8] and Equinox [31], which are the main requirements.

Acknowledgements

SBMLtoODEjax builds on SBMLtoODEpy's parsing and conversion of SBML files [9], JAX's composable transformations [8], Equinox's module abstraction [31] and BasiCO's access to the BioModels REST api [14].

This work has been funded by the biotechnology company Poietis and the French National Association of Research and Technology (ANRT), as well as by the French National Research Agency (ANR, project DeepCuriosity). It also benefited from two mobility research scholarships given by the French Academy (Jean Walter Zellidja scholarship) and the University of Bordeaux (UBGRS-Mob scholarship). Finally, this work benefited from the use of the Jean Zay supercomputer associated with the Genci grant A0091011996.

References

[1] H. Kitano, "Systems Biology: A Brief Overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, Mar. 2002, doi: 10.1126/science.1069492.

[2] H. de Jong, "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, Jan. 2002, doi: 10.1089/10665270252833208.

[3] F. M. Delgado and F. Gómez-Vela, "Computational methods for Gene Regulatory Networks reconstruction and analysis: A review," *Artificial Intelligence in Medicine*, vol. 95, pp. 133–145, Apr. 2019, doi: 10.1016/j.artmed.2018.10.006.

[4] M. Glont *et al.*, "BioModels: Expanding horizons to include more modelling approaches and formats," *Nucleic Acids Research*, vol. 46, no. D1, pp. D1248–D1253, Jan. 2018, doi: 10.1093/nar/gkx1023.

[5] R. S. Malik-Sheriff *et al.*, "BioModels—15 years of sharing computational models in life science," *Nucleic Acids Research*, vol. 48, no. D1, pp. D407–D415, Jan. 2020, doi: 10.1093/nar/gkz1055.

[6] M. Hucka *et al.*, "The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, Mar. 2003, doi: 10.1093/bioinformatics/btg015.

[7] M. Hucka *et al.*, "The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core Release 2," *Journal of Integrative Bioinformatics*, vol. 16, no. 2, p. 20190021, Jun. 2019, doi: 10.1515/jib-2019-0021.

[8] J. Bradbury *et al.*, "JAX: Composable transformations of Python+NumPy programs." 2018.

[9] S. M. Ruggiero and A. N. F. Versypt, "SBMLtoODEpy: A software program for converting SBML models into ODE models in Python," *Journal of Open Source Software*, vol. 5, no. 53, p. 1643, Sep. 2019, doi: 10.21105/joss.01643.

[10] G. Muzio, L. O'Bray, and K. Borgwardt, "Biological network analysis with deep learning," *Briefings in Bioinformatics*, vol. 22, no. 2, pp. 1515–1530, Mar. 2021, doi: 10.1093/bib/bbaa257.

[11] D. M. Camacho, K. M. Collins, R. K. Powers, J. C. Costello, and J. J. Collins, "Next-Generation Machine Learning for Biological Networks," *Cell*, vol. 173, no. 7, pp. 1581–1592, Jun. 2018, doi: 10.1016/j.cell.2018.05.015.

[12] M. AlQuraishi and P. K. Sorger, "Differentiable biology: Using deep learning for biophysics-based and data-driven modeling of molecular mechanisms," *Nature Methods*, vol. 18, no. 10, pp. 1169–1180, Oct. 2021, doi: 10.1038/s41592-021-01283-4.

[13] S. Hoops *et al.*, "COPASI—a COmplex PAthway SImulator," *Bioinformatics*, vol. 22, no. 24, pp. 3067–3074, Dec. 2006, doi: 10.1093/bioinformatics/btl485.

[14] F. T. Bergmann, "Copasi/basico: Release 0.48." Mar. 2023.

[15] L. M. Loew and J. C. Schaff, "The Virtual Cell: A software environment for computational cell biology," *Trends in Biotechnology*, vol. 19, no. 10, pp. 401–406, Oct. 2001, doi: 10.1016/S0167-7799(01)01740-1.

[16] B. M. Slepchenko, J. C. Schaff, I. Macara, and L. M. Loew, "Quantitative cell biology with the Virtual Cell," *Trends in Cell Biology*, vol. 13, no. 11, pp. 570–576, Nov. 2003, doi: 10.1016/j.tcb.2003.09.002.

[17] A. Funahashi, M. Morohashi, H. Kitano, and N. Tanimura, "CellDesigner: A process diagram editor for gene-regulatory and biochemical networks," *BIOSILICO*, vol. 1, no. 5, pp. 159–162, Nov. 2003, doi: 10.1016/S1478-5382(03)02370-9.

[18] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano, "CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks," *Proceedings of the IEEE*, vol. 96, no. 8, pp. 1254–1265, Aug. 2008, doi: 10.1109/JPROC.2008.925458.

[19] K. Choi *et al.*, "Tellurium: An extensible python-based modeling environment for systems and synthetic biology," *Biosystems*, vol. 171, pp. 74–79, Sep. 2018, doi: 10.1016/j.biosystems.2018.07.006.

[20] J. K. Medley *et al.*, "Tellurium notebooks—An environment for reproducible dynamical modeling in systems biology," *PLOS Computational Biology*, vol. 14, no. 6, p. e1006220, Jun. 2018, doi: 10.1371/journal.pcbi.1006220.

[21] N. Rodriguez *et al.*, "The systems biology format converter," *BMC Bioinformatics*, vol. 17, no. 1, p. 154, Apr. 2016, doi: 10.1186/s12859-016-1000-2.

[22] S. S. Schoenholz and E. D. Cubuk, "JAX, M.D.: A Framework for Differentiable Physics." arXiv, Dec. 2020. Accessed: Jun. 02, 2023. [Online]. Available: https://arxiv.org/abs/1912.04232

[23] E. J. Ma and A. Kummer, "Reimplementing Unirep in JAX." bioRxiv, p. 2020.05.11.088344, May 2020. doi: 10.1101/2020.05.11.088344.

[24] G. Carleo *et al.*, "NetKet: A machine learning toolkit for many-body quantum systems," *SoftwareX*, vol. 10, p. 100311, Jul. 2019, doi: 10.1016/j.softx.2019.100311.

[25] J.-E. Campagne *et al.*, "JAX-COSMO: An End-to-End Differentiable and GPU Accelerated Cosmology Library," *The Open Journal of Astrophysics*, vol. 6, p. 10.21105/astro.2302.05163, Apr. 2023, doi: 10.21105/astro.2302.05163.

[26] D. Häfner, R. Nuterman, and M. Jochum, "Fast, Cheap, and Turbulent—Global Ocean Modeling With GPU Acceleration in Python," *Journal of Advances in Modeling Earth Systems*, vol. 13, no. 12, p. e2021MS002717, 2021, doi: 10.1029/2021MS002717.

[27] S. Mann, E. Fadel, S. S. Schoenholz, E. D. Cubuk, S. G. Johnson, and G. Romano, "∂PV: An end-to-end differentiable solar-cell simulator," *Computer Physics Communications*, vol. 272, p. 108232, Mar. 2022, doi: 10.1016/j.cpc.2021.108232.

[28] A. Stanziola, S. R. Arridge, B. T. Cox, and B. E. Treeby, "J-Wave: An open-source differentiable wave simulator," *SoftwareX*, vol. 22, p. 101338, May 2023, doi: 10.1016/j.softx.2023.101338.

[29] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams, "JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows," *Computer Physics Communications*, vol. 282, p. 108527, Jan. 2023, doi: 10.1016/j.cpc.2022.108527.

[30] B. N. Kholodenko, "Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades," *European Journal of Biochemistry*, vol. 267, no. 6, pp. 1583–1588, Mar. 2000, doi: 10.1046/j.1432-1327.2000.01197.x.

[31] P. Kidger and C. Garcia, "Equinox: Neural networks in JAX via callable PyTrees and filtered transformations," *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[32] I. Babuschkin *et al.*, "The DeepMind JAX Ecosystem." 2020.

[33] P. Städter, Y. Schälte, L. Schmiester, J. Hasenauer, and P. L. Stapor, "Benchmarking of numerical integration methods for ODE models of biological systems," *Scientific Reports*, vol. 11, no. 1, p. 2696, Jan. 2021, doi: 10.1038/s41598-021-82196-2.