SymPy: Improving Foundational Open Source Symbolic Mathematics for Science: Proposal for CZI EOSS Cycle 4

This proposal was accepted for funding. See https://chanzuckerberg.com/eoss/proposals/sympy-improving-foundational-open-source-symbolic-mathematics-for-science/

1. Did you previously apply for funding for this or a related proposal under the CZI EOSS program?

   No

2. Have you previously received funding for this proposal under the CZI EOSS program?

   No

3. Proposal Purpose

   To improve the SymPy Python symbolic mathematics library in the key areas of performance, code generation, and documentation.

4. Amount Requested

   Year 1: 178737
   Year 2: 173474
   Total All Years: 352211

5. Proposal Summary

   Symbolic mathematics is an essential pillar in scientific computing. For over a decade, SymPy has been the symbolic mathematics pillar in Python scientific computing and also plays strong roles in the Sage, Octave, Julia, and R communities. Github reports that some 30 thousand repositories utilize SymPy and over 120 academic papers cite SymPy per year. We aim to sustain and grow SymPy's foundational role, including for many

biomedical applications, by reducing performance barriers, expanding the popular numerical code generation features, and overhauling our documentation. We have identified these three areas as the ones most likely to gain more users and developers.

SymPy will be adopted more broadly in science if its performance is drastically improved, as it is a limiting factor for intensive use cases. The library is written in pure Python, which makes it easier to maintain and extend, but this can limit its performance. Thus it is essential that the core algorithms are written for efficiency with optimized data structures. We will improve the performance of our core algorithms by an order of magnitude, with strong focus on the polynomial and matrix submodules because these submodules will provide wide reaching performance benefits across SymPy. The performance improvements will be achieved in two ways: firstly, by using faster algorithms and data structures, and secondly, by optionally interfacing with libraries such as FLINT (via python-flint) and SymEngine that are written in C and C++ and are designed to be as fast as possible. We will hire one (1) core SymPy developer (Oscar Benjamin) part time to implement these improvements.

Numerical code generation is SymPy's most widely used feature in scientific applications. SymPy expressions can be automatically transformed into efficient numerical codes in tens of programming languages and packages (C, Fortran, Julia, Rust, Tensorflow, PyTorch, etc.). This makes advanced, performance critical numerical calculations accessible to domain scientists. We will hire one (1) postdoc part time who will improve the numerical stability of generated code and expand code generation to more complex expressions. We will demonstrate the efficacy of the improvements on one of SymPy's biological applications: biomechanical simulation codes with PyDy.

The SymPy codebase is quite large, consisting of over 700k lines of Python code and over 1000 public functions and methods. However, a large fraction of this functionality is undocumented or under-documented leaving users without an entry point. SymPy has strong API documentation but is missing high level user guides and tutorials. API documentation helps users who already know what they are looking for, but user guides are needed to assist newer users who are not yet expert enough in the library to know where to look in the API. Such documentation requires expert knowledge of the library to write effectively. This proposal is to hire one (1) core SymPy developer (Aaron Meurer) part time to write high level documentation.

6. Work Plan

This plan describes three roadmap projects that will address a large number of SymPy's outstanding issues, make SymPy more accessible, and increase performance for biomedical research codes. Each project was selected because it needs sustained, focused development to establish a foundation for future incremental improvements.

Performance has a wide reaching role in SymPy's roadmap that is critical for retaining adoption over competitors. This project will focus on improving performance in linear algebra, calculus, equation solvers, polynomials, and the core of SymPy, which contains key routines affecting performance bottlenecks. A core developer (Oscar Benjamin) will perform the work at 0.4 FTE over 2 years.

There are three primary foci:

1. Rewrite the internals of matrices and linear algebra routines and adapt relevant APIs to leverage the new routines. Symbolic computation cannot use floating point libraries, e.g. LAPACK, so algorithms need to be built from scratch. The new matrix classes will implement efficient, sparse, division-free algorithms. A preliminary implementation gives speedups ranging from 10x to over 1000x for common bottlenecks. Although aimed at matrices, this work will also improve many fundamental algorithms, extending performance gains widely across the codebase.
2. Introduce safe limits in low-level functions that have unbounded computational cost. This approach will be applied in many parts of the codebase, e.g. numerical evaluation and construction of algebraic extensions, to eliminate extremely slow cases and make SymPy's performance more predictable.
3. Expand SymPy's existing benchmarks repository and fully integrate the benchmark execution into SymPy's continuous integration (CI). Benchmarking and targeted profiling will be used to identify and fix the worst performance issues in the codebase; starting with uncovering bottlenecks in popular user APIs such as integration and equation solving.

Code generation plays a central role in SymPy's roadmap for supporting scientific codes. The code generation roadmap prioritizes usability, generating high performance code, and supporting many languages. This project will focus on improving usability and performance. The 1 FTE postdoctoral researcher (to be hired) and 0.05 FTE Assistant Professor (Jason Moore) will develop a performance critical musculoskeletal model for benchmarking purposes. This model will be adopted from a bicycle vehicle model (Moore 2012) and a human arm model in (Chadwick 2014 & 2008). This human-vehicle model includes stiff dynamics, holonomic and nonholonomic constraints, thousands of algebraic operations, and complex force definitions; all features that strain SymPy's ability to best competitors in advanced biomechanical simulations.

The researcher will then incrementally improve a number of SymPy's submodules while testing against the human-vehicle model. Documentation and benchmarking will accompany each increment. They will improve the Autolev parsing module for importing models. They will enable common subexpression elimination (CSE) in the code generators; benefiting most output languages, as well as the very popular lambdify() function. They will develop code printers for matrix operations that leverage assumptions to print appropriate BLAS/LAPACK calls. They will create symbolic objects for muscle forces in the mechanics package, along with associated optimized code printers. The

improvements will make SymPy very attractive for high performance musculoskeletal simulations, such as those used in machine learning and optimal predictive control. These changes will also broadly benefit many other general and biomedical domain uses due to SymPy's code generation's ubiquitous use.

Improvements to SymPy's documentation is also a central part of SymPy's roadmap, as it is key to both new and advanced users' ability to effectively use the library. The documentation project aims to vastly increase and improve documentation. The project will focus on writing user guides and tutorials, which are currently lacking. The developer will begin by adding much needed guides for solvers, integration, and assumptions. Additionally, areas for improvement in the documentation will be identified from a systematic review of questions asked on StackOverflow and our forums, a survey sent out to active users, and by leveraging the developers' years of Q&A experience. The survey will collect feedback regarding the strengths and weaknesses in SymPy's documentation. It will be disseminated in our forums, on social media, and through NumFocus's newsletter. The data will be used to prioritize additional new guides to write.

During development of the guides, SymPy's extensive API reference documentation will be updated by applying our new style guide and adding missing API references. We will work closely with technical writers hired to work on SymPy as part of Google Season of Docs, as well as the larger SymPy community to address any documentation related challenges that arise during the work period. A core developer (Aaron Meurer) will do the documentation work 0.5 FTE over 2 years.

7. Milestones and Deliverables

Performance Milestones and Deliverables:

The key metric for the performance project will be measurable improvements in benchmark timings across the codebase with focus on matrix and linear algebra operations. With the CI-integrated benchmarking, speed gains will be reported with each pull request. The other metric will be the percentage closed of the approximately 100 long-standing GitHub issues related to performance and operations that "hang". The summary of performance results will be shared online and at international conferences for scientific Python.

This project will involve many continuous small fixes but the timeline for the main chunks of work will be:

- September - December, 2021: Extend the benchmark suite and review open performance related issues on GitHub.
- January - August, 2022: Complete the implementation of the new matrix classes and make use of them in other APIs.

- September 2022 - August 2023: Identify and fix/improve cases of unbounded computation and other cases where significant optimisation is possible in common operations.

Code Generation Milestones and Deliverables:

The key metrics for the code generation project will be usability (measured by reduction in boilerplate code for typical problems) and performance improvements (measured by benchmark computation speed gains from the benchmark suite and the new advanced model). The following list provides a bimonthly set of milestones for the 1 year project period:

- Months 1–2: Develop the benchmark model; Add improvements to the Autolev parser.
- Months 3–4: Add CSE to code generators and lambdify().
- Months 5–6: Add matrix operation code printers.
- Months 7–8: Develop muscle force models in sympy.physics.mechanics.
- Months 9–10: Develop muscle force code printers.
- Months 11–12: Finalize documentation; write an academic paper on the results, post an open access preprint, and submit to a journal.

Documentation Milestones and Deliverables:

The key metrics for the documentation project will be the number of new pages and number of new guides added to the documentation. We will also analyze the number of user questions for key topics on SymPy user forums and StackOverflow, which will be identified at the beginning of the project. The goal is that the number of questions on these topics decline as their documentation improves.

The timeline for this project will be:

- September 1 – 30, 2021: Develop and deploy a user survey, to find where documentation is lacking. Anonymized results will be published publicly to the SymPy community.
- September, 2021 – August, 2023: For each two-week period of the program, one of the following tasks will be accomplished on the SymPy documentation:
    - Write a new high-level page.
    - Edit and improve an existing high-level page.
    - Write missing reference documentation for a SymPy submodule.
    - Copyedit a page for consistency.

  Each of these tasks will be done as small incremental changes so that the results can be published to the documentation site immediately.

The two core developers, postdoctoral researcher, and assistant professor will have progress check in meetings every six months to validate the accompisments against the milestones detailed above.

8. Existing Support

In 2019 and 2020, Google Summer of Code gave in-kind support for 14 students to work on SymPy and paid SymPy a total of $11,700 through mentor stipends. In 2019, Google Season of docs gave in-kind support for 1 technical writer and paid SymPy a total of $1,500 as a mentor stipend. SymPy has received 2 small development grants from NumFOCUS in 2019 and 2020, totalling $6,000 of support. The 2019 and 2020 grants were to improve SymPy's ODE module and to improve SymPy Live and SymPy Gamma, respectively, and each took place in its respective year. Lastly, SymPy has received about $1000 in small donations over the two year period.

9. Landscape Analysis

SymPy is the most widely used general purpose open source computer algebra system (CAS). SymPy is over 16 years old and has over 800k downloads per month on PyPI. It is a full featured CAS, with over 700k lines of Python code. The most popular CAS competitors to SymPy are Mathematica and Maple, which are both proprietary. Among the alternative open source CASes, one of the most popular is Sage, which is a collection of various open source mathematics software packages, including SymPy itself, into a single product. Outside of Sage, SymPy stands as the only mature CAS usable from Python. The advantages of SymPy over Sage include the fact that it is standalone, its liberal BSD license, and that it is designed to be used as a library. For this reason, SymPy is typically preferred for usage by other libraries higher up in the scientific software stack.

Mpmath is a library for arbitrary precision numerics, and is a core dependency of SymPy. Mpmath is virtually the only Python library with such capabilities. Python-flint is a Python wrapper to FLINT, a C library for polynomial manipulation with speeds comparable to the best proprietary alternatives such as Mathematica. SymEngine is a C++ CAS designed to be performant, with an aim to be usable as an alternative fast backend in SymPy. SymPy Mechanics and PyDy provide flexible and feature rich competition to less accessible toolboxes in Maple and Matlab as well as standalone dynamics software such as MotionGenesis and SDFast.

10. Value to Biomedical Users

Much like NumPy, SciPy, and Matplotlib, SymPy is a foundational library at the bottom of the scientific Python dependency stack, supporting a large number of users across many

scientific fields. Supporting SymPy has a multiplicative effect by improving downstream domain science software tools in Python and other programming languages.

SymPy is used by several Python biological modelling packages, including Brian 2, PySB, MASSpy, and AMICI, which is in turn used by packages such as PESTO. SymPy finds its use in these packages as a tool for representing models symbolically, as well as other tasks such as parsing and simplifying calculations. The SymPy paper (Meurer, Aaron, et al. "SymPy: Symbolic computing in Python." PeerJ Computer Science 3 (2017): e103.) has been cited by nearly a hundred different papers relating to biology and the life sciences.

SymPy supports code generation to the tools PyTorch and Tensorflow, among others. These two tools are rapidly being adopted for machine learning efforts in the biological sciences. Code generation plays an essential role in the widely used Brian 2 spiking neural network simulator. Maybe more importantly, code generation is used in an ad hoc manner to develop essential elements of algorithms in thousands of biological software packages and one-off analyses. One specific example the performance and code generation improvements will enhance is biomechanical musculoskeletal modeling through the sympy.physics.mechanics sub-package and the downstream multibody dynamics projects PyDy.

11. Diversity, Equity, and Inclusion Statement

SymPy's leadership is committed to fostering a diverse, equitable, and inclusive community. We adopted a Code of Conduct (Contributor Covenant) in 2016 to establish a baseline for acceptable and desired behavior, based on our 10 years of experience in community interactions and best practices among leading inclusive software projects.

Throughout SymPy's 15 year history, the community has a record of attracting a wide range of diverse contributors (i.e. geographical location, background). Our dedicated participation in Google Summer of Code (GSoC) 14 times with around 80 participants plays a significant role in introducing newcomers to open source software development that may have not participated otherwise. Also, we use a large percentage of organization funds to sponsor disadvantaged GSoC students to present at software development conferences, which in the long-term can have a significant impact on their careers.

The project has introductory contribution and development guides aimed at improving the new contributor onboarding experience. It is our standard practice to give extra attention and mentorship to first time contributors in code reviews. We highly value bringing in new community members and try to minimize any overburdensome gatekeeping that discourages new contributors.

Although SymPy excels at international diversity, we struggle with gender diversity, as does much of the open source software world. We have recently brought in more gender diverse contributors through Google Season of Docs. We will use TU Delft's inclusive hiring practices for the postdoctoral position to create an opportunity to onboard and mentor another non-male contributor.

Number of Open Source Software Projects

How many software projects are involved in this proposal that will be supported by this grant?

1

Open Source Software Project #1: Details

1. Software Project name (required)
SymPy
2. Homepage URL (required)
https://www.sympy.org/
3. Hosting platform (required)
GitHub
4. Main code repository (e.g. GitHub URL) (required)
https://github.com/sympy/sympy
5. DOI of major publication(s) describing software project (if applicable)
https://www.doi.org/10.7717/peerj-cs.103
6. Social media handles (if applicable)
@SymPy on Twitter
7. Do you or software project key personnel have commit rights to the code repositories for this software project? (required)
Yes
8. Short description of software project (200 words maximum) (required)
SymPy is a Python library for symbolic mathematics. It is a full-featured computer algebra system (CAS), keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.

Software Project Metrics: Quality (required):

1. **What is the software project license?**

Permissive license (e.g. BSD 3-Clause, MIT, Apache 2.0)

2. **What is the main programming language?**

   Python

3. **Does the software project have a code of conduct?**

   Yes
   Link (optional; format https://example.com):
   https://github.com/sympy/sympy/blob/master/CODE_OF_CONDUCT.md
4. **Does the software project have end-user documentation?**

   Yes
   Link (optional; format https://example.com):
   https://docs.sympy.org/latest/index.html

5. **Does the software project have an issue tracker?**

   Yes
   Link (optional; format https://example.com):
   https://github.com/sympy/sympy/issues
6. **Does the software project have a community engagement / Q&A forum (self-hosted, on Stack Exchange etc.)?**

   Yes
   Link (optional; format https://example.com):
   http://groups.google.com/group/sympy

7. **Does the software project have contribution / coding guidelines?**

   Yes
   Link (optional; format https://example.com):
   https://github.com/sympy/sympy/wiki/Introduction-to-contributing

8. **Are there examples or demo notebooks, scripts, and datasets?**

   Yes
   Link (optional; format https://example.com):
   https://github.com/sympy/sympy/tree/master/examples

9. **Is there a corresponding package available in a package manager (PyPi, CRAN, etc.)?**

Yes
Link (optional; format https://example.com):
https://pypi.org/project/sympy/

10. **Does the software project support continuous integration for testing?**

Yes
Comment (optional):
SymPy uses the GitHub Actions CI to test all pull requests and the master branch.

Software Project Metrics: Impact (optional):

1. Complete the following table. List the number and explanation for each, if needed:

	Number	**Comment**
Scholarly paper(s) (including preprints) citing or mentioning the software project	4030	Google Scholar returns 4030 results for “SymPy” (the name “sympy” would not reference anything other than the SymPy library). 524 articles cite the paper “SymPy: symbolic computing in Python” (Meurer et al.)
Monthly users, if applicable (based on one or more of the following: monthly downloads from websites, monthly downloads from package managers, monthly unique requests for updates, etc.)	1500000	In April, 2021, there were 1,140,046 downloads from PyPI, according to https://pypistats.org/, plus 338616 downloads from conda (conda-forge and anaconda channels), according to condastats. Note: SymPy 1.8 was apreleased on April 9, 2021.
Software projects that depend on the project (if applicable)	924	924 packages depend on SymPy according to https://github.com/sympy/sympy/network/dependents. Additionally, 30,498 GitHub repositories depend on SymPy.
Monthly visitors to project’s	1000000	The SymPy websites do not

website, discussion forum (e.g. Stack Overflow), or similar		have direct analytics installed, but the Google Search Console estimates 1M impressions each for sympy.org and docs.sympy.org for April, 2021.

2. Size of the largest potential user base:

	Number	**Comment**
Estimate the potential number of unique users who could adopt this project in the relevant field/discipline. Use as guidance the number of users of comparable projects, the number of papers published in the domain to which the project is applicable, number of labs able to adopt the project, etc.	*Multiple Choice* *1-10* *11-100* *101-1,000* *1,001-10,000* *10,001-100,000* *Over 100,000* Over 100,000	SymPy has over 1M monthly downloads from package managers and is a dependency for over 900 upstream packages and 30k GitHub repositories. Its potential fields include any scientific discipline, not just biomedicine.

3. List of upstream, downstream, or related software projects that the team is contributing to or receiving contributions from:

If one line you can use: mpmath, gmpy2, antlr, sphinx, numpydoc, symengine, pydy and over 900 other packages

The performance project may involve integrating SymEngine and python-flint, and may also involve incidental improvements to SymPy's dependency mpmath.

The code generation project involves work with the downstream project PyDy, a multibody dynamics package that uses SymPy's mechanics submodule.

The documentation project may involve upstreaming incidental improvements to various documentation tooling packages, including Sphinx, MyST, and numpydoc.

4. Additional metrics from project code repositories and package managers:

SymPy is a core part of the scientific Python ecosystem, as the premier symbolic manipulation library for Python. Furthermore, SymPy is wrapped in packages for Julia (SymPy.jl) and Octave (Octave Symbolic), which are presently the most popular symbolic packages in those respective open source language communities. Downstream use cases of SymPy span most, if not all, of

the Chan Zuckerberg Initiative mission's fields of focus. SymPy is used in machine learning and data analysis tools such as Google's Tensorflow & Tensorflow Quantum, Apple's coremltools, University of Sheffield's GPy, and TensorTrade. In neuroscience, SymPy powers projects like Brian 2, NineML, and the Brain Modeling Toolkit with code generation. In cell biology, SymPy is used in packages such as AMICI, cameo, cobrapy, and pysb. SymPy's code generation is also used in prominent genomics packages such as GetOrganelle, cobrame, Medusa and millstone. The package devito uses SymPy code generation for imaging purposes and SymPy is used in trimesh, primify, and tributary for visualization.

SymPy has 8.1k stars and 3.4k forks on GitHub. It was started in 2005 and has had over 1000 individual contributors. In the past year, there have been over 150 individual contributors, about 2/3 of whom were first time contributors to SymPy in that time. There were over 700 pull requests merged into SymPy in the past year, which were merged by 19 different core developers. The number of pull requests per year has been stable over the past several years. (This data was computed from the git history.) Most, if not all, developers are volunteers. The core developers span the globe and come from a large diverse set of organizations and backgrounds. Most of them also contribute to related packages in the scientific Python ecosystem. One reason SymPy has such a high rate of contribution is the extensive documentation for new contributors: https://github.com/sympy/sympy/wiki/Introduction-to-contributing.

The SymPy community governs itself via an opt-in consensus model with contentious decisions or Code of Conduct issues settled by the project leaders.

Over 2000 issues and pull requests were opened in the SymPy issue tracker in the past year, 63% of which were opened by outside collaborators (people without push access to the SymPy repository). 61% of the issues and pull requests opened in the past year have been closed or merged. Of these, the median time to close/merge was 3 days.

SymPy's public mailing list (http://groups.google.com/group/sympy) has over 3000 members and has about 100 messages per month. In the past month, the "sympy" tag on StackOverflow has had 83 questions, of which 55 have received an answer. These answers have come from both SymPy's developers as well as the user community.

SymPy's public roadmap is published at https://www.sympy.org/en/roadmap.html. All three projects proposed here (performance, code generation, and documentation) are major parts of the SymPy roadmap.

References

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. "TensorFlow: A System for Large-Scale Machine Learning." In *12th USENIX Symposium on Operating Systems Design and*

Implementation (OSDI 16), 265–83. Savannah, GA: USENIX Association. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

Apple/Coremltools. (2017) 2021. Python. Apple. https://github.com/apple/coremltools.

Broughton, Michael, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, et al. 2020. "TensorFlow Quantum: A Software Framework for Quantum Machine Learning." *ArXiv:2003.02989 [Cond-Mat, Physics:Quant-Ph]*, March. http://arxiv.org/abs/2003.02989.

Cardoso, João G. R., Kristian Jensen, Christian Lieven, Anne Sofie Lærke Hansen, Svetlana Galkina, Moritz Beber, Emre Özdemir, Markus J. Herrgård, Henning Redestig, and Nikolaus Sonnenschein. 2018. "Cameo: A Python Library for Computer Aided Metabolic Engineering and Optimization of Cell Factories." *ACS Synthetic Biology* 7 (4): 1163–66. https://doi.org/10.1021/acssynbio.7b00423.

Chadwick, Edward K., Dimitra Blana, Antonie J. van den Bogert, and Robert F. Kirsch. 2009. "A Real-Time, 3-D Musculoskeletal Model for Dynamic Simulation of Arm Movements." *IEEE Transactions on Biomedical Engineering* 56 (4): 941–48. https://doi.org/10.1109/TBME.2008.2005946.

Chadwick, E.K., D. Blana, R.F. Kirsch, and A.J. van den Bogert. 2014. "Real-Time Simulation of Three-Dimensional Shoulder Girdle and Arm Dynamics." *IEEE Transactions on Biomedical Engineering* Early Access Online. https://doi.org/10.1109/TBME.2014.2309727.

Dai, Kael, Sergey L. Gratiy, Yazan N. Billeh, Richard Xu, Binghuang Cai, Nicholas Cain, Atle E. Rimehaug, et al. 2020. "Brain Modeling ToolKit: An Open Source Software Suite for Multiscale Modeling of Brain Circuits." *PLOS Computational Biology* 16 (11): e1008386. https://doi.org/10.1371/journal.pcbi.1008386.

Dawson-Haggerty, Michael. (2013) 2021. *Mikedh/Trimesh*. Python. https://github.com/mikedh/trimesh.

Ebrahim, Ali, Joshua A. Lerman, Bernhard O. Palsson, and Daniel R. Hyduke. 2013. "COBRApy: COnstraints-Based Reconstruction and Analysis for Python." *BMC Systems Biology* 7 (1): 74. https://doi.org/10.1186/1752-0509-7-74.

Fröhlich, Fabian, Daniel Weindl, Yannik Schälte, Dilan Pathirana, Łukasz Paszkowski, Glenn Terje Lines, Paul Stapor, and Jan Hasenauer. 2020. "AMICI: High-Performance Sensitivity Analysis for Large Ordinary Differential Equation Models." *ArXiv:2012.09122 [q-Bio]*, December. http://arxiv.org/abs/2012.09122.

Goodman, Daniel B., Gleb Kuznetsov, Marc J. Lajoie, Brian W. Ahern, Michael G. Napolitano, Kevin Y. Chen, Changping Chen, and George M. Church. 2017. "Millstone: Software for Multiplex Microbial Genome Analysis and Engineering." *Genome Biology* 18 (1): 101. https://doi.org/10.1186/s13059-017-1223-1.

GPy. 2012. “GPy: A Gaussian Process Framework in Python.” http://github.com/SheffieldML/GPy.

Haiman, Zachary B., Daniel C. Zielinski, Yuko Koike, James T. Yurkovich, and Bernhard O. Palsson. 2021. “MASSpy: Building, Simulating, and Visualizing Dynamic Biological Models in Python Using Mass Action Kinetics.” *PLOS Computational Biology* 17 (1): e1008208. https://doi.org/10.1371/journal.pcbi.1008208.

Jin, Jian-Jun, Wen-Bin Yu, Jun-Bo Yang, Yu Song, Ting-Shuang Yi, and De-Zhu Li. 2018. “GetOrganelle: A Simple and Fast Pipeline for de Novo Assembly of a Complete Circular Chloroplast Genome Using Genome Skimming Data.” *BioRxiv*, March, 256479. https://doi.org/10.1101/256479.

Lange, Michael, Navjot Kukreja, Mathias Louboutin, Fabio Luporini, Felippe Vieira, Vincenzo Pandolfo, Paulius Velesko, Paulius Kazakas, and Gerard Gorman. 2016. “Devito: Towards a Generic Finite Difference DSL Using Symbolic Python.” In *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, 67–75. https://doi.org/10.1109/PyHPC.2016.013.

levi.borodenko@gmail.com. (2019) 2021. *LeviBorodenko/Primify*. Python. https://github.com/LeviBorodenko/primify.

Lloyd, Colton J., Ali Ebrahim, Laurence Yang, Zachary A. King, Edward Catoiu, Edward J. O’Brien, Joanne K. Liu, and Bernhard O. Palsson. 2018. “COBRAme: A Computational Framework for Genome-Scale Models of Metabolism and Gene Expression.” *PLOS Computational Biology* 14 (7): e1006302. https://doi.org/10.1371/journal.pcbi.1006302.

Lopez, Carlos F, Jeremy L Muhlich, John A Bachman, and Peter K Sorger. 2013. “Programming Biological Models in Python Using PySB.” *Molecular Systems Biology* 9 (1): 646. https://doi.org/10.1038/msb.2013.1.

Medlock, Gregory L., Thomas J. Moutinho, and Jason A. Papin. 2020. “Medusa: Software to Build and Analyze Ensembles of Genome-Scale Metabolic Network Reconstructions.” *PLOS Computational Biology* 16 (4): e1007847. https://doi.org/10.1371/journal.pcbi.1007847.

Meurer, Aaron, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, et al. 2017. “SymPy: Symbolic Computing in Python.” *PeerJ Computer Science* 3 (January): e103. https://doi.org/10.7717/peerj-cs.103.

Moore, Jason K. 2012. “Human Control of a Bicycle.” Doctor of Philosophy, Davis, CA: University of California. http://moorepants.github.io/dissertation.

Paine, Tim. (2018) 2021. *Timkpaine/Tributary*. Python. https://github.com/timkpaine/tributary.

Raikov, Ivan, Robert Cannon, Robert Clewley, Hugo Cornelis, Andrew Davison, Erik De Schutter, Mikael Djurfeldt, et al. 2011. “NineML: The Network Interchange for Ne

Uroscience Modeling Language.” *BMC Neuroscience* 12 (1): P330. https://doi.org/10.1186/1471-2202-12-S1-P330.

Stapor, Paul, Daniel Weindl, Benjamin Ballnus, Sabine Hug, Carolin Loos, Anna Fiedler, Sabrina Krause, Sabrina Hroß, Fabian Fröhlich, and Jan Hasenauer. 2018. “PESTO: Parameter EStimation TOolbox.” *Bioinformatics* 34 (4): 705–7. https://doi.org/10.1093/bioinformatics/btx676.

Stimberg, Marcel, Romain Brette, and Dan FM Goodman. 2019. “Brian 2, an Intuitive and Efficient Neural Simulator.” Edited by Frances K Skinner. *ELife* 8 (August): e47314. https://doi.org/10.7554/eLife.47314.

Tensortrade-Org/Tensortrade. (2019) 2021. Python. tensortrade-org. https://github.com/tensortrade-org/tensortrade.