



Python Battery Mathematical Modelling (PyBaMM)

SOFTWARE METAPAPER

VALENTIN SULZER 

SCOTT G. MARQUIS 

ROBERT TIMMS 

MARTIN ROBINSON 

S. JON CHAPMAN 

**Author affiliations can be found in the back matter of this article*

]u[ubiquity press

ABSTRACT

As the UK battery modelling community grows, there is a clear need for software that uses modern software engineering techniques to facilitate cross-institutional collaboration and democratise research progress. The Python package PyBaMM aims to provide a flexible platform for implementation and comparison of new models and numerical methods. This is achieved by implementing models as expression trees and processing them in a modular fashion through a pipeline. Comprehensive testing provides robustness to changes and hence eases the implementation of model extensions. PyBaMM is open source and available on GitHub. For more information visit www.pybamm.org.

CORRESPONDING AUTHOR:

Valentin Sulzer

DPhil Student; Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford OX2 6GG, GB

valentinsulzer@hotmail.com

KEYWORDS:

battery modelling; expression tree; python; symbolic differentiation

TO CITE THIS ARTICLE:

Sulzer V, Marquis SG, Timms R, Robinson M, Chapman SJ 2021 Python Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, 9: 14. DOI: <https://doi.org/10.5334/jors.309>

1 OVERVIEW

INTRODUCTION

With the battery modelling research community growing rapidly in the UK in the last few years [1], it is essential to develop tools that facilitate cross-institutional collaboration. One such tool is battery modelling software that allows new research (e.g. new physics, numerical methods) to be employed with minimal effort by the rest of the community. Modelling software should also facilitate quantitative comparison of different models and numerical methods. To be reusable and extendable, the software should be both modular, so that new models and numerical methods can easily be added, and rigorously tested, in order to be robust to changes.

EXISTING SOFTWARE

Currently, COMSOL [2] is the modelling software most commonly utilised by the the battery modelling community, providing a simple graphical user interface (GUI) to implement and solve standard battery models. However, there are several drawbacks to COMSOL. Firstly, its prohibitively expensive licence fee is a barrier to collaboration and sharing of software. Secondly, it has limited flexibility for adaptation of existing battery models, or investigation of new numerical methods. Thirdly, as the implementation is performed through a GUI, programs cannot be directly scripted without additional software, which inhibits version control, unit testing and combination with other software (for example, for parameter estimation).

Several existing open-source battery modelling software packages provide an alternative option to COMSOL. Examples include DUALFOIL [3], fastDFN [4], LIONSIMBA [5], and M-PET [6]. However, each of these packages is focused on the implementation of one specific battery model under a specific choice of operating conditions. As a result, these software packages lack the flexibility required to allow for easy implementation of reduced-order versions of a model or to include model extensions. This lack of flexibility considerably limits the reuse of such packages across different research projects.

OVERVIEW OF PYBAMM

PyBaMM (Python Battery Mathematical Modelling) is a tool for fast and flexible simulations of battery models. Our mission is to accelerate battery modelling research by providing an open-source framework for multi-institutional, interdisciplinary collaboration. PyBaMM offers improved collaboration and research impact in battery modelling by creating a modular framework through which either existing or new tools can be combined in order to solve continuum models for batteries. To achieve this, PyBaMM separates the models, discretisation and solver, giving ultimate flexibility to the end user, and provides a unified interface through which to incorporate new

models, alternative spatial discretisations, or new time-stepping algorithms. Any such additions can then immediately be used with the existing suite of models already implemented, and comparisons can be made between different models, discretisations, or algorithms with variables such as hardware, software and implementation details held fixed. Similarly, additional physics can be incorporated into the existing models, enabled by the extensible “plug-and-play” submodel structure around which models are constructed. As a result, the need to start from scratch to study each new effect is removed, and the simultaneous study of a range of extensions to the standard battery models, for example by coupling together several degradation mechanisms, is readily achieved. A comprehensive suite of tests provides the robustness necessary to allow the continual addition of new models and solvers in an open-source framework.

PyBaMM is one of the major components of the Faraday Institution’s ‘Common Modelling Framework’, part of the Multi-Scale Modelling Fast Start project, which will act as a central repository for UK battery modelling research. PyBaMM has already been used to develop and compare reduced-order models for lithium-ion [7] and lead-acid [8, 9] batteries, parameterize lithium-ion cells [10], model spirally-wound batteries [11], model two-dimensional distributions in the current collectors [12, 13], and model SEI growth [14]. Further research outcomes are anticipated from continued collaborations with other members of the modelling community, both within and beyond the Faraday Institution. An up-to-date list of papers that use PyBaMM can be found at pybamm.org/publications.

PyBaMM is an Affiliated Project with NumFOCUS, and builds on other tools in the NumFOCUS ecosystem including NumPy [15], SciPy [16], pandas [17], Matplotlib [18], and Project Jupyter [19].

IMPLEMENTATION AND ARCHITECTURE

PyBaMM’s architecture is based around two core components. The first is the expression tree, which encodes mathematical equations symbolically (see [Figure 1](#)). Each expression tree consists of a set of symbols, each of which represents either a variable, parameter, mathematical operation, matrix, or vector. Every battery model in PyBaMM is then defined as a collection of symbolic expression trees. The expression trees in each model are organised within python dictionaries representing the governing equations, boundary equations, and initial conditions of the model.

An example of implementing a simple diffusion model using expression trees is provided in Appendix A. Further examples of creating ODE and PDE models can be found in the [“Creating Models”](#) notebooks hosted online. To clearly demonstrate how to set up multi-domain and

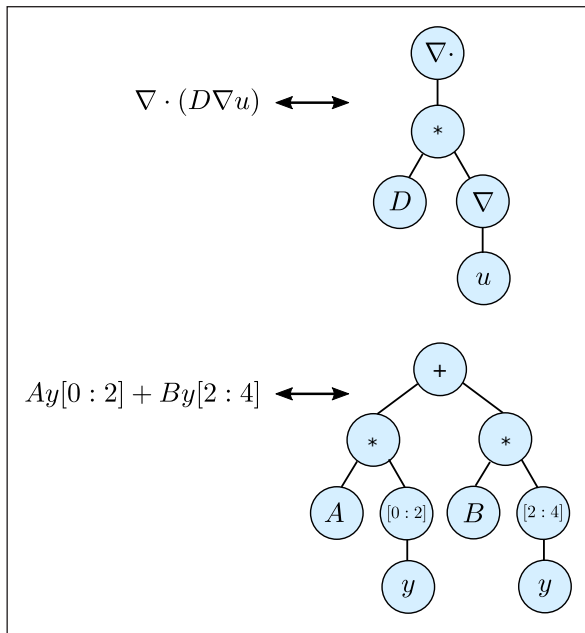


Figure 1 Models are encoded and passed down the pipeline using a symbolic expression tree data-structure. Leafs in the tree represent parameters, variables, matrices etc., while internal nodes represent either basic operators such multiplication or division, or continuous operators such as divergence or gradients.

multi-physics models PyBaMM includes “basic” versions of the SPM (`basic_spm.py`) and DFN (`basic_dfn.py`) which are defined in a single script, separate from the submodel structure. These are intended to act as a learning tool, and can be found in the lithium-ion models sub-directory of the PyBaMM GitHub repository.

The second core component of PyBaMM’s architecture is the pipeline process (see [Figure 2](#)). In the pipeline process different modular components operate on the model in turn. The pipeline is constructed in Python using PyBaMM classes, so that users have full control over the entire process, and can customise the pipeline or insert their own components at any stage. [Figure 2](#) depicts a typical pipeline with the following stages:

1. Define a battery model and geometry using PyBaMM’s syntax. This generates a collection of expression trees representing the model.
2. Parse the expression trees for the battery model and geometry, replacing any parameters with their provided numerical values. For convenience, parameter values may be provided in a csv file.
3. Mesh the geometry and discretise the model on this mesh with user-defined spatial methods. This process parses each expression tree converting variables into state vectors, and spatial operators (e.g. gradient and divergence) into matrices (accounting for the boundary conditions of the model).
4. Solve the model using a time-stepping algorithm. PyBaMM offers a consistent interface to a number of ordinary differential equation (ODE), differential

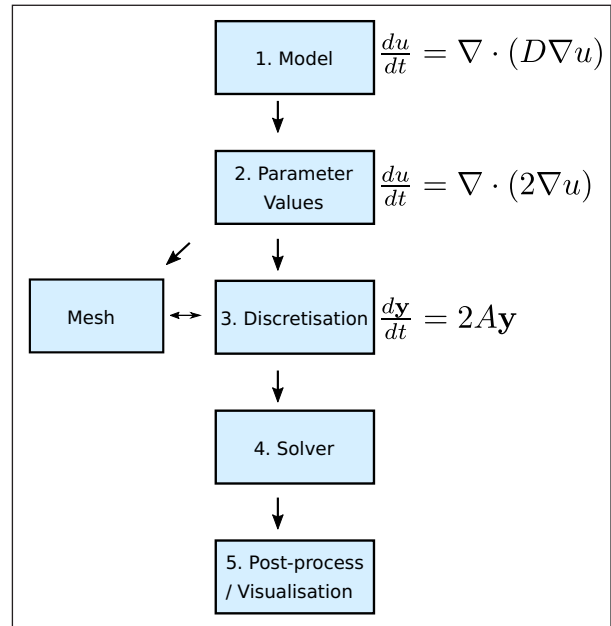


Figure 2 PyBaMM is designed around a pipeline approach. Models are initially defined using mathematical expressions encoded as expression trees. These models are then passed to a class which sets the parameters of the model, before being discretised into linear algebra expressions, and finally solved using a time-stepping class.

algebraic equation (DAE), and algebraic (root-finding) solvers, including via SciPy [16], SUNDIALS [20, 21, 22], CasADi [23], and JAX [24]. One of the main benefits of PyBaMM’s expression tree structure is that it provides the capability to automatically compute the Jacobian for any model, using symbolic differentiation, which significantly improves the performance of the numerical solvers.

5. Post-processes the solution. Built-in post processing utilities provided access to any user-defined output variables at any solution time or state. Additionally, PyBaMM includes a number of visualisation utilities which allow for easy plotting and comparison of any of the model variables (for example output, see [Figure 3](#)).

The various stages of the pipeline process are handled automatically by PyBaMM’s `Simulation` class, providing a user friendly way to solve battery models. The simplest example to use PyBaMM is to run a 1C constant-current discharge with a given model with all the default settings, as shown in [Listing 1](#). For greater customisation users can pass different parameters, adjust the mesh and discretisation, change the solver, and tailor the output of the plots, all via the `Simulation` class. For example, experimental protocols can be simulated using a simple text-based syntax, as shown in [Listing 2](#), or simulate non-constant current (dis)charge by passing time-current data. For more information please consult the latest documentation.

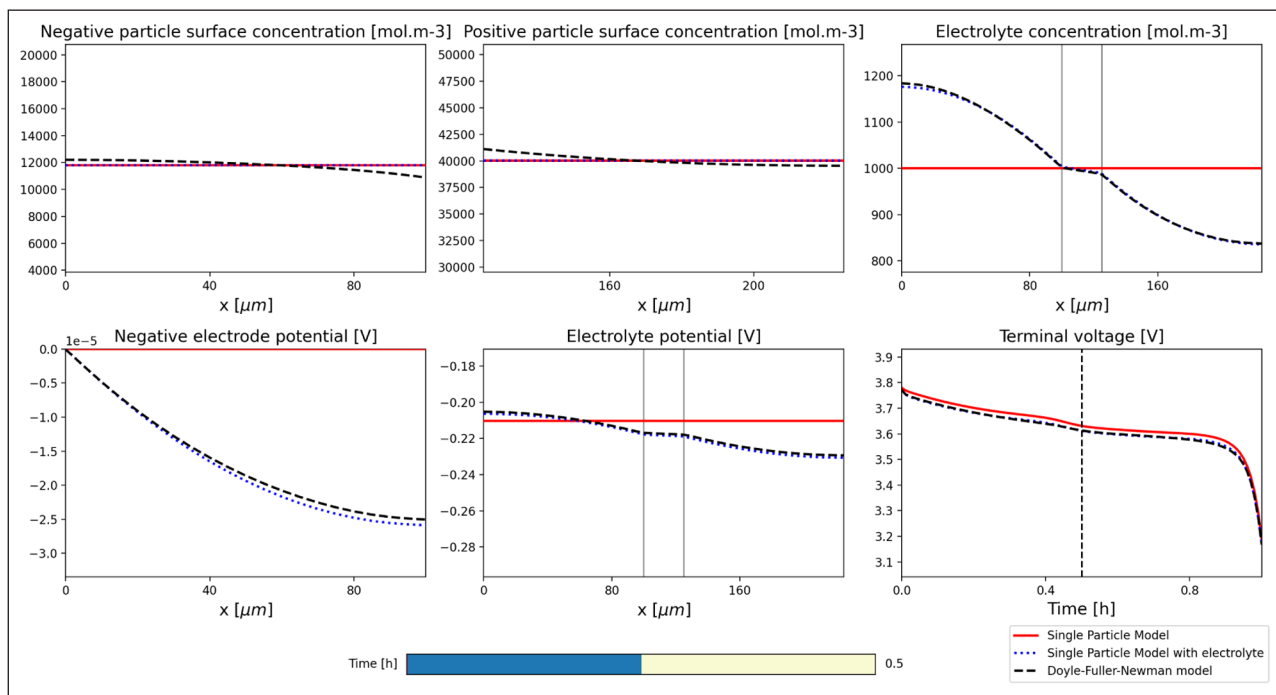


Figure 3 Interactive visualisation of solutions. The user can select the time at which to view the output using the time-slider bar at the bottom. This interactive plot is automatically generated by providing a list of model solutions and output variables.

Listing 1 Running a simulation in PyBaMM.

```

1 import pybamm
2 # Doyle-Fuller-Newman model
3 model = pybamm.lithium_ion.DFN()
4 sim = pybamm.Simulation(model)
5 sim.solve([0, 3600]) # solve for 1 hour
6 sim.plot()

```

Listing 2 Running an experiment in PyBaMM.

```

1 import pybamm
2 experiment = pybamm.Experiment(
3     [
4         "Discharge at C/10 for 10 hours
5         or until 3.3 V",
6         "Rest for 1 hour",
7         "Charge at 1 A until 4.1 V",
8         "Hold at 4.1 V until 50 mA",
9         "Rest for 1 hour",
10    ]
11    * 3,
12 )
13 model = pybamm.lithium_ion.DFN()
14 sim = pybamm.Simulation(
15     model,
16     experiment=experiment,
17     solver=pybamm.CasadiSolver(),
18 )
19 sim.solve()
20 sim.plot()

```

QUALITY CONTROL

Tests in PyBaMM are performed within the `unittest` framework. We follow a test-driven development process, and unit tests are implemented for every

class with unit test code coverage consistently above 98%. In addition, a smaller set of integration tests are implemented to ensure the end-to-end reliability of the code. The integration tests consist of tests that check every model in PyBaMM can be processed and solved for a set of default inputs, convergence tests between reduced-order and full-order models, convergence tests for each spatial method, and tests for each solver type.

PyBaMM is developed using git version control, with all unit and integration tests being run cross-platform via GitHub Actions every time a pull request is made. At the time of writing, the PyBaMM tests run on Ubuntu, macOS and Windows systems with Python 3.7-3.9.

The main PyBaMM repository contains a selection of [Jupyter Notebooks](#) that provide a useful set of examples on how to use PyBaMM for different tasks such as creating a new battery model, running the existing models, or changing the default parameters. These are tested along with the main PyBaMM code to ensure they are up to date. All of the examples, as well as a “Getting Started” guide can be accessed from the [PyBaMM website](#) and can be run interactively in a web browser via [Google Colab](#) with no installation necessary. Further examples can be found on the accompanying case studies repository, which, among other things, shows how PyBaMM can be used for parameter estimation and simulation of drive-cycle experiments.

Please consult the `CONTRIBUTING.md` file in the PyBaMM repository for more detailed and up-to-date information on our development workflow, testing and CI infrastructure, and coding style guidelines.

(2) AVAILABILITY OPERATING SYSTEM

PyBaMM can run on any Linux, MacOS or Windows system that has Python 3.6-3.8 installed, along with the dependencies listed below. The optional dependency, `scikits-odes`, currently only supports Linux and MacOS. For Windows users, we therefore recommend using Windows Subsystems for Linux (WSL); detailed instructions are available on GitHub. On Linux and MacOS, Google's JAX library can be used to provide additional autograd and solver capabilities.

PROGRAMMING LANGUAGE

Python 3.6-3.8

ADDITIONAL SYSTEM REQUIREMENTS

PyBaMM has no special requirements and can be run on a standard laptop or desktop machine.

DEPENDENCIES

Required:

- `numpy` ≥ 1.16
- `scipy` ≥ 1.3
- `pandas` ≥ 0.24
- `anytree` ≥ 2.4.3
- `autograd` ≥ 1.2
- `scikit-fem` ≥ 0.2.0
- `casadi` ≥ 3.5.0,
- `jupyter` (for example notebooks)
- `matplotlib` ≥ 2.0
- `jax` = 0.1.75, (not supported on Windows)
- `jaxlib` = 0.1.52, (not supported on Windows)

Optional:

- `scikits.odes` ≥ 2.4.0 (optional DAE solver, requires SUNDIALS 5.0.0)

LIST OF CONTRIBUTORS

The following people have contributed in some form to the development of PyBaMM at time of writing. An up-to-date list of contributors can be found in our `README`. Core developers are indicated in bold.

Valentin Sulzer, **Scott Marquis**, **Robert Timms**, **Martin Robinson**, **Ferran Brosa-Planella**, **Tom Tranter**, **Thibault Lestang**, Diego Alonso Álvarez, Jacqueline Edge, Colin Please, Jon Chapman, Fergus Cooper, Felipe Salinas, Peter Cho, Suhak Lee, Vivian Tran, Yannick Kuhn, Alexander Bessman, Daniel Albamonte, Anand Mohan Yadav, Weilong Ai.

SOFTWARE LOCATION

Name: GitHub (release v0.2.4)

Persistent identifier: <https://github.com/pybamm-team/PyBaMM/releases/tag/v0.2.4>

Licence: BSD 3-clause

Publisher: The PyBaMM team

Version published: v0.2.4

Date published: 07/09/20

Code repository

Name: GitHub

Persistent identifier: <https://github.com/pybamm-team/PyBaMM>

Licence: BSD-3-Clause

Date published: 04/11/2018

LANGUAGE

English

(3) REUSE POTENTIAL

We anticipate that the main use case will be the implementation, extension, and comparison of new models and parameter sets. For example, this will allow researchers to implement models that couple several degradation mechanisms together. Further, although PyBaMM has been written with battery models in mind, the expression tree and pipeline architecture could be potentially be used to solve different sets of continuum models numerically.

In addition to new models and parameter sets, the modular framework described in Section 1 allows researchers to add new numerical algorithms in the form of spatial discretisations or new ODE/DAE solvers. Any such extensions can then be immediately tested with the existing set of models and parameters. This allows researchers to quickly assess the accuracy and speed of their numerical algorithms for a range of models and relevant parameter values.

Information on how to extend the software in these ways is available both through tutorials in the API docs and example notebooks. All of the development is done through GitHub issues and pull requests, using the workflow explained in the `CONTRIBUTING.md` file. Users can request support by raising an issue on GitHub.

A CREATING A MODEL

In this section, we present an example of how to enter a simple diffusion model in PyBaMM. This model serves as a good representation of the types of models that arise in battery modelling because it contains most of the key components: spatial operators, parameters, Dirichlet and Neumann boundary conditions, and initial conditions.

We consider the concentration of some species c , on a spatial domain $x \in [0,1]$, and at some time $t \in [0,\infty)$. The concentration of the species is taken to evolve according to a nonlinear diffusion process with the concentration being fixed at $x = 0$ and a constant

inward flux of species imposed at $x = 1$. Mathematically, the model is stated as

$$\frac{\partial c}{\partial t} = \nabla \cdot (D(c) \nabla c) \quad \text{in } 0 < x < 1, \quad t > 0, \quad (1a)$$

$$c = 1 \quad \text{at } x = 0, \quad (1b)$$

$$D(c) \frac{\partial c}{\partial x} = 1 \quad \text{at } x = 1, \quad (1c)$$

$$c = x + 1 \quad \text{at } t = 0, \quad (1d)$$

where $D(c) = k(1+c)$ is the diffusion coefficient and k is a parameter, which we will refer to as the diffusion parameter.

In [Listing 3](#), we provide the PyBaMM code implementing (1). Note that operator overloading of $*$ and $+$ allows symbols to be intuitively combined to produce expression trees. A more detailed and up-to-date introduction to the syntax is provided in the online examples available on GitHub.

The model is now represented by a collection of expression trees and can therefore be solved by passing it through the pipeline just like any other model in PyBaMM. Additionally, extending the model to include

additional physics is simple and intuitive due to the simple symbolic representation of the underlying mathematical equations. For example, we can add a source term to the governing equation (1a) by only modifying one line of code (line 10 of [Listing 3](#)) and still obtain useful properties of the model such as the analytical Jacobian.

The common interface of all PyBaMM models makes it easy to perform the pipeline process as illustrated here upon multiple models or the same model with different options activated. Therefore, comparing the results of different models, mesh types, discretisations, and solvers then becomes straightforward within the PyBaMM framework.

ACKNOWLEDGEMENTS

The authors are grateful to all contributors, workshop attendees, and the following people for useful discussions, feedback and support on the early development of PyBaMM: Jamie Foster, David Howey, Ivan Korotkin, Charles Monroe, Greg Offer, and Giles Richardson.

FUNDING STATEMENT

VS acknowledges funding from the EPSRC Center for Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1) in collaboration with BBOX. SM acknowledges funding from the EPSRC Center for Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1) in collaboration with Siemens. RT and JC acknowledge support from the Faraday Institution (EP/S003053/1). MR acknowledges funding from the EPSRC Impact Acceleration Account - University of Oxford (D4D00010).


COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Valentin Sulzer  orcid.org/0000-0002-8687-327X
DPhil Student; Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford OX2 6GG, GB

Scott G. Marquis  orcid.org/0000-0002-6895-990X
DPhil Student; Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford OX2 6GG, GB

Robert Timms  orcid.org/0000-0002-8858-4818
PDRA; Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford OX2 6GG, GB

Listing 3 Defining a model in PyBaMM.

```

1 # 1. Initialise model
2 model = pybamm.BaseModel()
3
4 # 2. Define parameters and variables
5 c = pybamm.Variable("c", domain="unit
line")
6 k = pybamm.Parameter("Diffusion parameter")
7
8 # 3. State governing equations
9 D = k * (1 + c)
10 dcdt = pybamm.div(D * pybamm.grad(c))
11 model.rhs = {c: dcdt}
12
13 # 4. State boundary conditions
14 D_right = pybamm.BoundaryValue(D, "right")
15 model.boundary_conditions = {
16     c: {
17         "left": (1, "Dirichlet"),
18         "right": (1/D_right, "Neumann")
19     }
20 }
21
22 # 5. State initial conditions
23 x = pybamm.SpatialVariable("x",
domain="unit line")
24 model.initial_conditions = {c: x + 1}

```


Martin Robinson  orcid.org/0000-0002-1572-6782

RSE; Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

S. Jon Chapman  orcid.org/0000-0003-3347-6024

Professor; Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford OX2 6GG, GB

REFERENCES

- EPSRC Press Office.** Greg Clark announces Faraday Institution, October 2017. [epsrc.ukri.org/newsevents/news/faradayinstitution/](https://www.epsrc.ukri.org/newsevents/news/faradayinstitution/). [Online; accessed 11-September-2019].
- COMSOL Inc.** COMSOL multiphysics reference manual, version 5.4. www.comsol.com.
- Newman J.** FORTRAN programs for the simulation of electrochemical systems. www.cchem.berkeley.edu/jsngrp/fortran.html.
- Moura SJ.** Fast doyle-fuller-newman (DFN) electrochemical-thermal battery model simulator. github.com/scott-moura/fastDFN.
- Torchio M, Magni L, Gopaluni RB, Braatz RD, Raimondo DM.** LION-SIMBA: A matlab framework based on a finite volume model suitable for li-ion battery design, simulation, and control. *Journal of The Electrochemical Society*. 2016; 163(7): A1192–A1205. DOI: <https://doi.org/10.1149/2.0291607jes>
- Smith RB, Bazant MZ.** Multiphase porous electrode theory. *Journal of The Electrochemical Society*. 2017; 164(11): E3291–E3310. DOI: <https://doi.org/10.1149/2.0171711jes>
- Marquis SG, Sulzer V, Timms R, Please CP, Chapman SJ.** An asymptotic derivation of a single particle model with electrolyte. *Journal of The Electrochemical Society*. 2019; 166(15): A3693. DOI: <https://doi.org/10.1149/2.0341915jes>
- Sulzer V, Chapman SJ, Please CP, Howey DA, Monroe CW.** Faster lead-acid battery simulations from porous-electrode theory: Part I. Physical model. *Journal of The Electrochemical Society*. 2019; 166(12): A2363–A2371. DOI: <https://doi.org/10.1149/2.0301910jes>
- Sulzer V, Chapman SJ, Please CP, Howey DA, Monroe CW.** Faster leadacid battery simulations from porous-electrode theory: Part II. Asymptotic analysis. *Journal of The Electrochemical Society*. 2019; 166(12): A2372–A2382. DOI: <https://doi.org/10.1149/2.0441908jes>
- Chen CH, Brosa Planella F, O'Regan K, Gastol D, Widanage WD, Kendrick E.** Development of Experimental Techniques for Parameterization of Multi-scale Lithium-ion Battery Models. *Journal of The Electrochemical Society*. 2020; 167(8): 080534. DOI: <https://doi.org/10.1149/1945-7111/ab9050>
- Tranter TG, Timms R, Heenan TMM, Marquis SG, Sulzer V, Jnawali A, Kok MDR, Please CP, Chapman SJ, Shearing PR,** et al. Probing heterogeneity in li-ion batteries with coupled multiscale models of electrochemistry and thermal transport using tomographic domains. *Journal of The Electrochemical Society*. 2020; 167(11): 110538. DOI: <https://doi.org/10.1149/1945-7111/aba44b>
- Marquis SG, Timms R, Sulzer V, Please CP, Chapman SJ.** A suite of reduced-order models of a single-layer lithium-ion pouch cell. *Journal of The Electrochemical Society*. 2020; 167(14): 140513. DOI: <https://doi.org/10.1149/1945-7111/abbce4>
- Timms R, Marquis SG, Sulzer V, Please CP, Chapman SJ.** Asymptotic reduction of a lithium-ion pouch cell model. *arXiv preprint arXiv:2005.05127*, 2020. <https://arxiv.org/abs/2005.05127>.
- Salinas F, Kowal J.** Discharge rate capability in aged li-ion batteries. *Journal of the Electrochemical Society*. 2020. DOI: <https://doi.org/10.1149/1945-7111/abc207>
- Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ,** et al. Array programming with NumPy. *Nature*. 2020; 585(7825): 357–362. DOI: <https://doi.org/10.1038/s41586-020-2649-2>
- 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. *Nature methods*. 2020; 17(3): 261–272. DOI: <https://doi.org/10.1038/s41592-019-0686-2>
- The pandas development team.** pandas-dev/pandas: Pandas, February 2020. DOI: <https://doi.org/10.5281/zenodo.3509134>
- Hunter JD.** Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007; 9(3): 90–95. DOI: <https://doi.org/10.1109/MCSE.2007.55>
- Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, Kelley K, Hamrick J, Grout J, Corlay S, Ivanov P, Avila D, Abdalla S, Willing C, Jupyter development team.** Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides F, Schmidt B (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, Netherlands, 2016. IOS Press. DOI: <https://doi.org/10.3233/978-1-61499-649-1-87>
- Hindmarsh AC.** The PVODE and IDA algorithms. *Technical report, Lawrence Livermore National Lab., CA (US)*, 2000. DOI: <https://doi.org/10.2172/802599>
- Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS.** SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*. 2005; 31(3): 363–396. DOI: <https://doi.org/10.1145/1089014.1089020>
- Malengier B, Kišon P, Tocknell J, Abert C, Bruckner F, Bisotti M-A.** ODES: a high level interface to ODE and DAE solvers. *The Journal of Open Source Software*. Feb 2018; 3(22): 165. DOI: <https://doi.org/10.21105/joss.00165>
- Andersson JAE, Gillis J, Horn G, Rawlings JB, Diehl**

M. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*. 2019; 11(1): 1–36. DOI: <https://doi.org/10.1007/s12532-018-0139-4>

24. **Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, Wanderman-Milne S.** JAX: composable transformations of Python+NumPy programs, 2018. github.com/google/jax.

TO CITE THIS ARTICLE:

Sulzer V, Marquis SG, Timms R, Robinson M, Chapman SJ 2021 Python Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, 9: 14. DOI: <https://doi.org/10.5334/jors.309>

Submitted: 05 November 2019 Accepted: 13 April 2021 Published: 08 June 2021

COPYRIGHT:

© 2021 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press.

