Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 59-5 (2025) 169–174

Unlocking electrochemical battery modelling: an open-source suite comparison methodology ⋆

Andres M. Diaz Aguilar, Alessio A. Lodge, and Feye S.J. Hoekstra *

* *Powertrains Dept., TNO Mobility & Built Environment, Helmond, The Netherlands (e-mail: alessio.lodge@tno.nl)*

Abstract: This paper presents a methodology to compare open-source suites for electrochemical battery simulations in terms of simulation output and computation time. The proposed approach leverages the Battery Parameter eXchange standard to introduce a parameter porting framework that supports the implementation of equivalent cell parameters across suites, and provides a method for solver tolerance selection used for computational speed comparison. The methodology is showcased by comparing two of the most comprehensive open-source suites, PyBaMM and TOOFAB. While both provide lean model implementation, PyBaMM shows more versatility and faster mean simulation times, with TOOFAB closing the gap on simulations with coarser discretizations, particularly when order reductions are applied. Furthermore, considering portability to C and implementation simplicity, TOOFAB has a strong advantage.

Keywords: Lithium ion batteries; physics-based model; fast charging; PyBaMM; TOOFAB

1. INTRODUCTION

Lithium-ion batteries (LIBs) are fundamental energy storage technologies for the energy transition, particularly important for decarbonizing transport through battery electric vehicles (BEVs). However, despite their crucial role, BEVs face adoption challenges due to longer refuelling (charging) times in comparison to combustion vehicles, as well as performance loss and safety risks due to battery degradation. To address these challenges, it is crucial to develop fast charging strategies that optimize charging speed and mitigate degradation.

While fast-charging strategies have been developed using data-driven equivalent-circuit models (Drees et al., 2022) and deep learning approaches (Park et al., 2022), physics-based approaches, which leverage electrochemical Physics-Based Models (PBMs), offer the unique advantage of capturing the underlying electrochemical and thermal processes within the battery. Through control strategies based on non-invasive estimation of the system's electrochemical states derived from PBMs, health-aware fast charging strategies can be designed to modulate inputs that reduce ageing and maximise charging speed. This includes limiting internal conditions that thermodynamically induce degradation, such as limiting the anode potential to reduce lithium plating (Rangarajan et al., 2020), as well as using PBMs with additional physics to directly model ageing phenomena, allowing for direct control of degradation as done by Medina et al. (2023); Khalik et al. (2020).

Having determined that PBMs are well suited for fast charging control, PBMs remain a critical area of research, with key remaining challenges including 1) model parametrisation of real cells, 2) which model equations and parameter realisations should be used to model the electrochemical behaviour, and 3) computational performance of PBM solvers to achieve real-time applicability. To tackle these challenges, the academic community has developed a number of open-source battery modelling suites, an overview of which is shown in Table 1. Differences between the suites lie in the employed electrochemical models, including variants of the Doyle Fuller Newman (DFN) and Single Particle Model (with electrolyte) (SPM(e)); model approximations and order reduction techniques; programming language and software architecture; numerical solver implementation; and portability to embedded systems. Certain suites are also more complete than others by including thermal models, ageing models, and parametrisation capabilities. However, determining which suite is most suitable for fast charging control or comparing suites in terms of model accuracy and computational performance is far from trivial due to model equation differences, parameter inconsistencies, solver tolerances, etc.

This paper presents a methodology for comparing open-source PBM suites with a focus on employing PBMs in real-time control applications, making a first direct comparison PyBaMM and TOOFAB, two of the most complete suites currently in the literature (see Table 1). The proposed comparison methodology enables the use of a single parameter set through a unified porting framework based on the BPX (Battery Parameter eXchange, see Korotkin et al. (2023)) standard. It also provides a method for solver tolerance selection critical for comparison of computational speed. This comparison sheds light on fundamental differences between suites, providing a roadmap for selecting and optimizing PBM simulators for real-time applications, particularly for fast charging.

2. COMPARISON METHODOLOGY

Differences in model equations and parameters employed make it a non-trivial process to model the same cell across software suites. To tackle this, we leverage the BPX open information exchange standard, which defines the parameters used in PBMs to model a cell, the electrochemical

⋆ This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101103898 under title of NEXTBMS (https://nextbms.eu).

10.1016/j.ifacol.2025.07.100

Table 1. Open-source battery modelling software.

Suite	Language	Model	Ageing	Parameterization	Thermal model	Original source
PyBaMM	Python	DFN, SPM(e)	✓	✓[1]	✓	Sulzer et al. (2021)
TOOFAB	MATLAB	DFN	✓	✓	✓	Khalik et al. (2021a)
SPMeT[2]	MATLAB	SPMe		✓	✓	Wassiliadis et al. (2022)
COBRAPRO	MATLAB	DFN		✓		Ha et al. (2024)
LIONSIMBA	MATLAB	DFN			✓	Torchio et al. (2016)
DEARLIBS	MATLAB	DFN				Lee et al. (2021)
PETLION	Julia	DFN	✓		✓	Berliner et al. (2021)
JuBat	Julia	DFN, SPM(e)			✓	Ai et al. (2024)
Liibra	Julia	Reduced DFN				Planden et al. (2022)
BatterySimulator	C++	DFN, SPM				Jiang et al. (2022)
batP2dFoam	C++	DFN				Yin et al. (2023)

[1] While not natively part of PyBaMM, PyBOP (see Planden et al. (2024)) may be used to parameterize PyBaMM models
[2] Based on the model developed by Perez et al. (2016); Moura et al. (2017).

model equations employing these parameters, and a specific JSON schema to store and share cell parameters, which may be used for portability across simulators. In this paper, we make use of the 12.5 Ah NMC111-graphite pouch cell parametrised by About:Energy, available in the BPX format at About:Energy (2023). To accommodate for the differences in parameter definitions between suites, porting tools are required to map BPX parameters to suite-specific ones. While such a porting tool is provided natively for PyBaMM, the same must be developed for other software suites to ensure accurate parameter definitions tailored to each suite's modelling approach. Developing one for TOOFAB exposed a number of parameter discrepancies, highlighted in Section 3.

Solver tolerances can have a significant effect on computation times, as stricter tolerances increase the computational effort required for accuracy improvement. Depending on the solver, different tolerances may yield similar accuracies, meaning that tolerances for each suite must be determined individually based on the simulation accuracy they yield. One approach to do so has been proposed by Khalik et al. (2021a), in which the tolerance is set as the largest possible value such that the Normalized Root Mean Square Error (NRMSE) between the tested model and a reference model (with a fixed standard tolerance, typically 10^{-6}) remains below 10^{-4}, ensuring consistency in terms of achieved accuracy. The NRMSE is given by

$$\frac{1}{|\mathcal{Z}|}\sum_{p,q\in\mathcal{Z}}\frac{\sqrt{\frac{1}{K}\sum_{k=0}^{K}(p_k-q_k)^2}}{\max_k\left\{\frac{1}{2}(p_k+q_k)\right\}-\min_k\left\{\frac{1}{2}(p_k+q_k)\right\}}, \quad (1)$$

with k the time index, K the maximum time, $p = q \in \mathcal{Z} = \{V, \phi_e, c_e, c_{s,\text{surf}}, j_n\}$ with p the reference solution and q the reduced tolerance solution, $|\mathcal{Z}|$ the cardinality of $\mathcal{Z}$, V the terminal voltage, ϕ_e the electrolyte potential, c_e electrolyte concentration, $c_{s,\text{surf}}$ solid surface concentration, and j_n the particle interfacial molar flux. With the tolerance determined, for dynamic time-stepping solvers, the maximum allowable time-step setting for which the simulation remained stable was used, as limiting it drastically increases computation time. For static time-stepping solvers, such as TOOFAB's in-house solver, the default time-stepping of $dt = 1$s was used.

To compare the performance of different suites, we consider as input three different current profiles (illustrated in Figure 1), namely

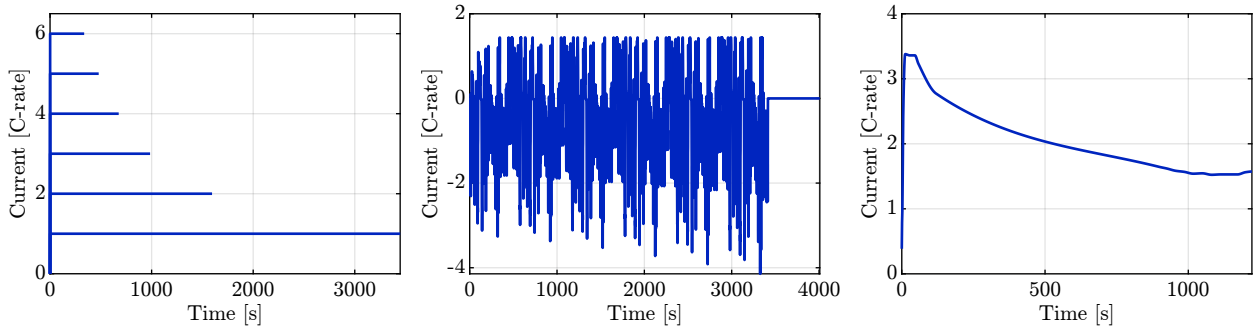
(1) Constant Current (CC) charge at 1C, 2C, 3C, 4C, 5C, and 6C.
(2) US06 drive cycle test starting at 0.8 SOC
(3) Fast charge profile starting at 0.1 SOC (inspired by Wassiliadis et al. (2022))

CC inputs at different C-rates are used as higher C-rates exacerbate model non-linearities, revealing differences between the suites. The drive cycle is applied due to its dynamic nature, which presents a challenge to solvers applying dynamic time stepping. Lastly, the fast charge cycle is primarily intended as a demonstration to showcase the application we are ultimately pursuing.

3. HIGH-LEVEL SUITE COMPARISON

PyBaMM (Python Battery Mathematical Modeling) is currently the most popular battery modelling suite as well as the one with the most features, seeing community-based development and continuous funding. Introduced by Sulzer et al. (2021), PyBaMM is a Python-based suite defined by its modularity, explicitly separating the battery model selection, discretization approach, numerical solver, parameter choice, and solver settings in the simulation environment. Paired with a broad set of predefined electrochemical models (e.g. DFN, SPMe, SPM); a set of 15 python-based numerical solvers as well as the C++ based IDAKLU solver; a variety of sub-models to capture additional physics and electrochemical reactions (e.g. thermal effects, side reactions or stress related ageing); a comprehensive list of predefined parameter set for various cell formats and chemistries derived from literature; and a combination of predefined post-processing functionalities; PyBaMM allows for users to easily create and modify their simulations to accommodate their requirements, with full documentation, examples and video tutorials. Furthermore, PyBaMM provides a series of additional libraries which build upon their electrochemical simulation capabilities, such as LiionPack for the modelling of battery packs, and PyBOP for cell model parametrization.

In contrast, TOOFAB, developed by Khalik et al. (2021a) and le Roux et al. (2023), proposes a MATLAB-based suite for DFN-type battery modelling. While it does not include other PBMs and has fewer predefined parameter sets, it provides a lean DFN solver implementation that does not rely on any toolboxes to perform its computations, as all governing equations are discretized and solved numerically with an in-house Newton-based approach. TOOFAB also offers the ability to toggle thermal and ageing models, as



(a) Constant Current (CC): 1C to 6C (b) Drive Cycle (DC) (c) Fast Charge (FC)

Fig. 1. Input current profiles used to compare the suites.

well as a number of pre-defined model simplifications to increase computational speed, including:

[S1] A linearization of the Butler-Volmer equation with respect to the overpotential η around the origin, resulting in $j_n = \frac{i_0}{RT}(\phi_s - \phi_e - U)$

[S2] A zeroth-order Taylor approximation of concentration-dependent parameters (κ, D_e, ν, and D_s), with evaluation points either chosen as [S2-I] the values of the concentration-dependent parameters at the previous time step, [S2-II] or at a constant expected average concentration and stoichiometry ($\tilde{c_e} = c_{e,0}$; $\tilde{\theta} = (\theta_{100\%} + \theta_{0\%}/2)$).

[S3] A two-parameter polynomial approximation of the solid-phase diffusion, assuming a parabolic concentration profile over the radial particle dimension.

Besides this, TOOFAB includes a parameter estimation routine with positive electrode OCP function fitting (assuming a standard graphite OCP), using parameter grouping in the fitting procedure to improve identifiability, reflecting earlier works of Khalik et al. (2021b).

While PyBaMM and TOOFAB differ greatly in their implementations, they make up the most complete suites in terms of features as illustrated in Table 1. Both include modelling of ageing processes, a lumped thermal model, and thermal/concentration-dependent parameters. Additionally, both include model reduction options to improve simulation times, such as the modification of spatial discretization along the x direction and particle r direction, or the substitution of the Fickian Diffusion Law (in the particle mass transfer equation) with a polynomial approximation. Only TOOFAB however includes the zeroth-order Taylor approximation of concentration-dependent parameters and the linearization of the Butler-Volmer equation.

Concerning usability, while both suites include the option to provide input current data as a time-series, PyBaMM allows the current to be given as a "recipe". This refers to a list of commands provided as a Python string (such as: "1C discharge until 2.5V"), from which the input current is derived. PyBaMM also allows for power, voltage, and resistance to be considered as simulation inputs. On the other hand, TOOFAB allows for the input current to be provided as a function handle, allowing users to specify the input freely for dynamic testing strategies.

Regarding parameter definitions, even if the same DFN modelling framework is adopted by PyBaMM, TOOFAB, and the BPX standard, discrepancies in their model equations are reflected by the usage of differing parameters. For example, the thermodynamic factor ν from the electrolyte charge conservation equation, commonly defined as

$$\nu(c_e, T) = \left(1 - t_+^0\right)\left(1 + \frac{d\ln f_\pm}{d\ln c_e}\right), \tag{2}$$

with t_+^0 the transference number and $f_\pm$ the mean molar salt activity coefficient (Valøen et al., 2005), is not present in the BPX DFN equations. While t_+^0 is defined in BPX, $f_\pm$ is not, as it is not used in their charge conservation formulation. As such, the direct porting of these parameters, which are used in both PyBaMM and TOOFAB model equations, is not possible. The native PyBaMM BPX porting tool follows the typical assumption of $\frac{d\ln f_\pm}{d\ln c_e} \approx 0$ to circumvent this, which has been implemented in the TOOFAB porting process to ensure parameter harmonization. Importantly, while TOOFAB defines $\frac{d\ln f_\pm}{d\ln c_e}$ and t_+^0 as individual parameters, PyBaMM defines the thermodynamic factor as $\nu_{\text{Py}}(c_e, T) = 1 + \frac{d\ln f_\pm}{d\ln c_e}$. This naming discrepancy means that to define $\frac{d\ln f_\pm}{d\ln c_e}$ as a parameter in PyBaMM, it must be defined as ν_{Py} and shifted by 1.

The BPX electrolyte charge conservation equation also does not include the Bruggeman constant b, required by TOOFAB and PyBaMM. However, by including the transport efficiency $\mathcal{B}$ (also known as inverse MacMullin number), which relates these parameters through $\mathcal{B}(x) = \epsilon^b$, where ϵ is the electrolyte porosity, b can be extracted. The native PyBaMM BPX porting tool employed in this paper does not take this into account, and instead sets $b = 1.5$ for any BPX parameter set, a feature that is recommended to be changed in future PyBaMM updates.

Another important difference between the model equations from BPX, TOOFAB and PyBaMM lies in the Bulter-Volmer kinetics, specifically as they relate to the exchange current density j_0. The BPX standard defines j_0 with ratios to reference maximum solid concentration $c_{s,\max}$ and electrolyte concentration $c_{e,0}$, while TOOFAB and PyBaMM omit these ratios. To account for this, the ported reaction rate $K_{\text{Py,TF}}$ adjusts for these ratios, as well as the Faraday constant F included in the BPX's expression of j_0, also not present in the suites, resulting in

$$K_{\text{Py,TF}} = \frac{K_{\text{BPX}} F}{c_{s,\max} c_{e,0}^{0.5}}. \tag{3}$$

Regarding C-code portability, the programming language of each suite plays a defining role, as portability often relies on language-specific properties. MATLAB-based

Table 2. Simulation results with V_t in [mV], c^-_{bulk} and $c^-_{s,\text{surf}}$ in $[\frac{\text{mol}}{\text{m}^3}]$, and t/t_s in [-]$\times 10^{-3}$.

	CC (mean over C-rates)				Drive cycle				Fast charge			
	RMSD			Time	RMSD			Time	RMSD			Time
	V_t	c^-_{bulk}	$c^-_{s,\text{surf}}$	t/t_s	V_t	c^-_{bulk}	$c^-_{s,\text{surf}}$	t/t_s	V_t	c^-_{bulk}	$c^-_{s,\text{surf}}$	t/t_s
Py_{20}	0.0	0.0	0.0	2.3	0.0	0.0	0.0	10.7	0.0	0.0	0.0	1.3
Py_{10}	3.7	5.4	28.4	1.5	0.6	1.0	15.2	5.7	0.3	3.2	16.8	0.8
Py_{3}	19.4	75.6	351.8	1.4	2.4	14.8	72.7	3.5	1.2	45.3	204.8	0.8
$\text{Py}_{20,\text{rcp}}$	36.9	16.3	54.2	3.4	-	-	-	-	-	-	-	-
$\text{Py}_{10,\text{rcp}}$	37.4	13.0	57.2	2.3	-	-	-	-	-	-	-	-
$\text{Py}_{3,\text{rcp}}$	81.8	72.0	359.4	2.0	-	-	-	-	-	-	-	-
$\text{Py}^{\text{IDA}}_{20}$	0.0	0.0	0.6	2.1	0.1	2.0	2.0	4.4	0.0	0.3	0.3	1.1
$\text{Py}^{\text{IDA}}_{10}$	3.7	5.4	28.4	1.7	0.6	1.2	15.3	2.7	0.2	3.8	17.3	0.8
$\text{Py}^{\text{IDA}}_{3}$	19.4	75.6	351.8	1.2	2.4	15.8	72.7	1.8	1.2	45.9	204.9	0.7
$\text{Py}^{\text{IDA}}_{20,\text{rcp}}$	18.8	9.9	12.4	2.7	-	-	-	-	-	-	-	-
$\text{Py}^{\text{IDA}}_{10,\text{rcp}}$	19.5	4.6	27.4	2.3	-	-	-	-	-	-	-	-
$\text{Py}^{\text{IDA}}_{3,\text{rcp}}$	28.3	65.8	352.2	2.1	-	-	-	-	-	-	-	-
TF_{20}	5.1	25.0	1018.4	15.3	1.1	22.4	27.7	21.5	0.8	19.1	436.5	13.3
TF_{10}	4.1	40.8	1028.7	2.4	1.2	21.2	23.7	3.4	0.7	27.8	439.0	2.3
TF_{3}	18.6	171.4	1181.4	1.2	3.1	68.9	76.9	1.5	1.1	125.0	492.3	0.9
$\text{TF}_{20,\text{s}}$	5.1	25.0	1018.3	8.0	1.1	22.4	27.7	8.5	0.8	19.1	436.5	8.0
$\text{TF}_{10,\text{s}}$	4.1	40.8	1028.5	1.8	1.2	21.2	23.6	1.8	0.7	27.8	439.0	1.6
$\text{TF}_{3,\text{s}}$	18.6	171.4	1181.2	0.9	3.1	68.8	76.9	1.0	1.1	124.9	492.3	0.9

suites have the unique ability to utilize the Coder toolbox to automatically generate C-code for a subset of MATLAB functions. This is particularly relevant for TOOFAB, which provides its full DFN model and solver within one standalone .m file, without relying on toolboxes or external solvers. However, other MATLAB-based suites that use non-compatible toolboxes or structures unsupported by MATLAB Coder will present issues for C-code generation.

Differently to MATLAB, Python-based suites, such as PyBaMM, generally do not offer good portability to C-code. PyBaMM's use of dynamic typing and object-oriented features does not directly translate into the static typing, procedural design, and manual memory management requirements of C, and would thus need manual integration and dependency management. This is without accounting for PyBaMM's usage of Python libraries, which would need to be ported for full functionality. Nevertheless, there is the possibility to export PyBaMM C-code through the SUNDIALS generate function, the results of which, however, are not directly deployable on an embedded system.

4. RESULTS

The complete set of model realisations compared in this study is presented in Table 3, with tolerances determined using the approach described in Section 2. Py_d and TF_d denote PyBaMM and TOOFAB respectively, and $d \in \{3, 10, 20\}$ the spatial discretisation of all cell domains, including discretization in x of both electrodes and separator, and r of the electrode particles.

All simulation outputs are directly compared to Py_{20}, taken as reference, which uses the default CASADI solver and the finest mesh option of the study. Furthermore, this setup uses time-series based input current allowing for direct comparison of results. When recipe-based simulations are compared, interpolation is used to match the sample times of the time-based reference. To compare simulation outputs, the Root Mean Square Difference (RMSD) is used to quantify the deviation of model outputs of interest from the reference model. The RMSD for c^-_{bulk} is given by

$$\sqrt{\frac{1}{K}\sum_{k=0}^{K}\left(c^k_{\text{bulk,ref}} - c^k_{\text{bulk,sim}}\right)^2}, \tag{4}$$

and for $c^-_{s,\text{surf}}$ by

$$\sqrt{\frac{1}{K}\sum_{k=0}^{K}\frac{1}{X}\sum_{x=0}^{X}\left(c^{k,x}_{s,\text{surf,ref}}|_{r_{\max}} - c^{k,x}_{s,\text{surf,sim}}|_{r_{\max}}\right)^2}, \tag{5}$$

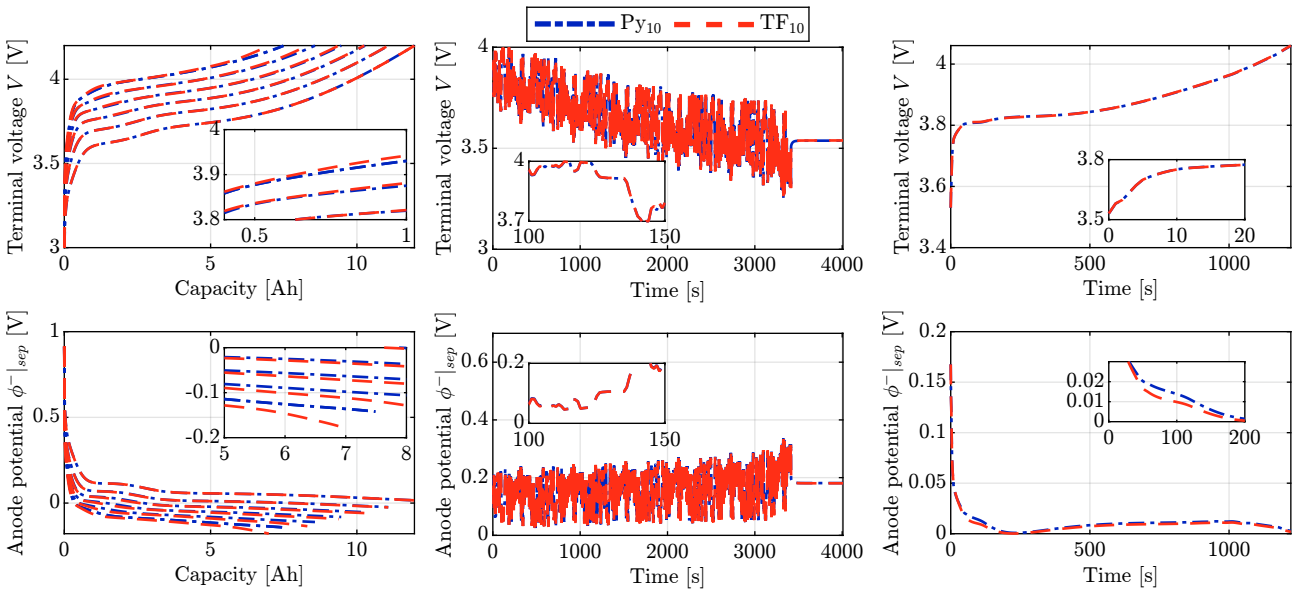
with x the position along the cell length and X the total cell thickness. Furthermore, simulation times (real-time taken to complete the simulation) have been normalized by the experiment duration. All RMSD and normalized simulation time results are presented in Table 2.

It must be noted that the RMSD is not a true simulation error, but instead highlights the difference between a particular simulation and its reference counterpart. Furthermore, the comparison of concentrations between time-series and recipe-based inputs is not entirely fair: since the current input is different, the capacity through time will vary and naturally so will the bulk and surface concentrations. It is displayed here purely for reflection, reinforcing

Table 3. Considered model forms

simulation	simp.	solver	input	abs. tolerance
Py_d	-	CASADI*	time-series	4×10^{-6}
$\text{Py}_{d,\text{rcp}}$	-	CASADI*	recipe based	4×10^{-6}
$\text{Py}^{\text{IDA}}_{d}$	-	IDAKLU	time-series	2×10^{-6}
$\text{Py}^{\text{IDA}}_{d,\text{rcp}}$	-	IDAKLU	recipe based	2×10^{-6}
TF_d	-	in-house	time-series	0.05
$\text{TF}_{d,\text{s}}$	[S2-I]	in-house	time-series	0.05

* The "safe" configuration was used for CC simulations (with the default $dt_{\max} = 600$s for 1C, and $dt_{\max} = 50$s for higher C-rates), and the "fast" configuration for the drive cycle and fast charge.



(a) Constant Current (CC): 1C to 6C (b) Drive Cycle (DC) (c) Fast Charge (FC)

Fig. 2. Comparison of the terminal voltage and anode potential of Py_{10} and TF_{10} simulated with inputs from Figure 1.

the herein proposed use of time-series based comparisons as the prime methodology for future studies.

While both suites show good agreement on terminal voltage at different C-rates, the TOOFAB anode potential displays an increased difference to PyBaMM for higher C-rates, as showcased in Figure 2. This difference is largely caused by a discrepancy in spatially resolved open-circuit potential, a component of anode potential, itself a function of solid surface concentration. As can be observed in Figure 3, the surface concentration of particles near the separator increases more rapidly over time in TOOFAB than in PyBaMM, whereas particles closer to the current collectors see a slower increase of the same. This indicates a faster lithiation of particles near the separator and slower lithiation for those near the current collectors. This does not impact the bulk concentration over time, as the difference in $c^-_{s,\text{surf}}$ is orders of magnitude greater than in c^-_{bulk}. This spatial imbalance in lithiation could have implications on simulated degradation, especially during fast charging, where higher local lithiation (and thus higher surface concentration) near the separator may increase lithium plating in TOOFAB more than PyBaMM.

The model simplification [S2-I] in TOOFAB minimally influences simulation accuracy, as demonstrated by consistently low RMSD values across all input current types. Meanwhile, it remarkably accelerates computation times, particularly with coarser mesh discretizations. TOOFAB offers additional simplifications to [S2-I] as discussed in 3, which may be implemented for further improvements in computation times. While these will lead to decreased accuracy as compared to the simplifications used in this paper, they also display stability issues, particularly observed for [S3], and hence were excluded in the study.

TOOFAB also exhibits very consistent computation times regardless of the input current used. This is different from PyBaMM, which is more sensitive to dynamic current profiles, resulting in a higher fraction of real-time taken to simulate drive cycle current profiles, particularly true for CASADI-based simulations. Nevertheless, TOOFAB is less well optimized for simulations with very fine mesh discretizations, showing a disproportionate increase in computation times for simulations in which $d = 20$.

Regarding PyBaMM solver discrepancies, while time-series based simulations show a faster mean computation time over C-rates in comparison to recipe-based simulations, this does not hold true generally for all C-rates. For low C-rates, recipe-based simulations have a marginally faster computation time, which is quickly lost as C-rates increase. Furthermore, a larger RMSD is observed for recipe-based CC simulations when compared to time-series based due to the coarser time discretization exhibited in these simulations. This is reflected in Figure 4, where very large time steps can be observed at the beginning of the CASADI-based $Py_{10,\text{rec}}$ simulation. It is important to keep in mind that RMSEs were calculated based on the number of discretizations of the reference simulation, and as such points in between the time discretization points of $Py_{10,\text{rec}}$ had to be interpolated, at the root of the large difference.

However, while IDAKLU recipe-based simulations also see fewer time steps than their time-series based counterparts, the simulation portions in which more time steps are taken are much more wisely achieved than for CASADI simulations (fewer time steps when linear increase, more when non-linear). This is particularly visible in Figure 4, where $Py^{\text{IDA}}_{10,\text{rec}}$ has a more refined time discretization around the origin than time-series based simulations, while still having less total time steps (139 and 362 respectively). This shows that IDAKLU is very well suited for dynamic time stepping with recipe-based simulation to achieve smoother curves, and maintains overall very good fidelity with the reference simulations. Nevertheless, this smart time stepping does not translate to large improvements in computation times. If anything, while the recipe-based IDAKLU experiments have fewer total time steps than time-series based, computation times are longer overall,

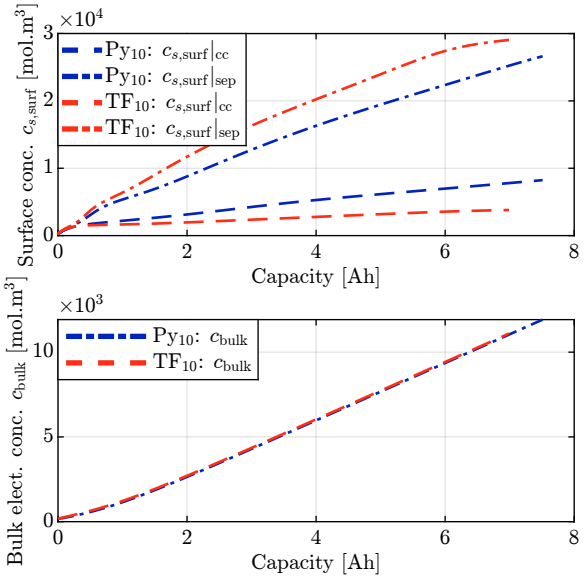


Fig. 3. Comparison of negative electrode surface concentration at different spatial points (top) and bulk concentration (bottom) for Py_{10} and TF_{10} during 6C charge.

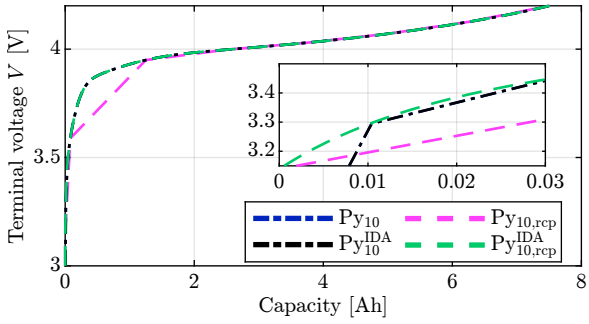


Fig. 4. Comparison of CASADI and IDAKLU time-series and recipe-based input for 6C CC charge.

which is likely the result of the solver having to optimize for time step size over the full simulation.

5. CONCLUSION

This paper has presented a comprehensive methodology for comparing open-source PBM software suites. A detailed comparison of PyBaMM and TOOFAB has exemplified this framework and revealed key differences between them. TOOFAB demonstrated consistent and fast simulation times, particularly with order reductions applied and coarser spatial discretizations, making it particularly well-suited for embedded applications with code that offers easy portability to C. PyBaMM, with its modular architecture and diverse solver options, offered better computation times overall, with consistent simulation times as the mesh was refined compared to TOOFAB, but more sensitivity to dynamic current profiles. Furthermore, it presented greater flexibility, with the possibility to model accurate and fast recipe-based simulations with the IDAKLU solver.

Future work should expand the comparison to a wider array of suites and include additional modelling effects, such as ageing and thermal phenomena for a more complete picture of the modelling capabilities of each suite.

REFERENCES

(2023). About:energy 12.5 Ah NMC111—graphite pouch cell parameterisation for BPX.

Ai, W. et al. (2024). Jubat: A julia-based framework for battery modelling using fem. *SoftwareX*.

Berliner, M.D. et al. (2021). Methods—PETLION: Open-source software for millisecond-scale porous electrode theory-based Li-ion battery simulations. *JES*.

Drees, R. et al. (2022). Durable fast charging of Li-ion batteries based on simulations with an electrode equivalent circuit model. *Batteries*.

Ha, S. et al. (2024). Cobrapro: An open-source software for the doyle-fuller-newman model with co-simulation parameter optimization framework. *JES*.

Jiang, Y. et al. (2022). A user-friendly Li battery simulator based on open-source cfd. *Digital Chemical Engineering*.

Khalik, Z. et al. (2020). Ageing-aware charging of Li-ion batteries using an electrochemistry-based model with capacity-loss side reactions. In *Proc. IEEE ACC*.

Khalik, Z. et al. (2021a). Model simplifications and their impact on computational complexity for an electrochemistry-based battery modeling toolbox. *J. Power Sources*.

Khalik, Z. et al. (2021b). Parameter estimation of the doyle–fuller–newman model for Li-ion batteries by parameter normalization, grouping, and sensitivity analysis. *J. Power Sources*.

Korotkin, I. et al. (2023). Battery parameter exchange.

le Roux, F.A. et al. (2023). Improved parameter estimation of the doyle-fuller-newman model by incorporating temperature dependence. *IFAC-PapersOnLine*.

Lee, S.B. et al. (2021). A robust and sleek electrochemical battery model implementation: A MATLAB framework. *JES*.

Medina, R. et al. (2023). Health-conscious charging of Li-ion battery cells: Using PBMs to minimize calendar and cyclic ageing effects. In *Proc. IEEE VPPC*.

Moura, S.J. et al. (2017). Battery state estimation for a SPM with electrolyte dynamics. *IEEE TCST*.

Park, S. et al. (2022). A deep reinforcement learning framework for fast charging of Li-ion batteries. *IEEE T TRANSP ELECTR*.

Perez, H.E. et al. (2016). Optimal charging of batteries via a SPM with electrolyte and thermal dynamics. In *Proc. IEEE ACC*.

Planden, B. et al. (2024). Python battery optimisation and parameterisation (pybop).

Planden, B. et al. (2022). A computationally informed realisation algorithm for Li-ion batteries implemented with liibra.jl. *J. Energy Storage*.

Rangarajan, S.P. et al. (2020). Anode potential controlled charging prevents Li plating. *J. Mater. Chem. A*.

Sulzer, V. et al. (2021). Python battery mathematical modelling (pybamm). *JORS*.

Torchio, M. et al. et al. (2016). Lionsimba: a matlab framework based on a finite volume model suitable for Li-ion battery design, simulation, and control. *JES*.

Valøen, L.O. et al. (2005). Transport properties of LiPF6-based Li-ion battery electrolytes. *JES*.

Wassiliadis, N. et al. (2022). A systematic approach for the parameter identification of electrochemical battery models enabling health-aware fast charging control of battery electric vehicles. *J. Energy Storage*.

Yin, X. et al. (2023). batP2dFoam: An efficient segregated solver for the P2D model of Li-ion batteries. *JES*.