The Journal of Open Source Software

PyBOP: A Python package for battery model optimisation and parameterisation

Brady Planden [1], **Nicola E. Courtier** [1,2], **Martin Robinson** [3], **Agriya Khetarpal** [4], **Ferran Brosa Planella** [2,5], and **David A. Howey** [1,2¶]

1 Department of Engineering Science, University of Oxford, Oxford, UK **2** The Faraday Institution, Harwell Campus, Didcot, UK **3** Research Software Engineering Group, University of Oxford, Oxford, UK **4** Quansight PBC **5** Mathematics Institute, University of Warwick, Coventry, UK ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The Python Battery Optimisation and Parameterisation (`PyBOP`) package provides methods for estimating and optimising battery model parameters, offering both deterministic and stochastic approaches with example workflows to assist users. `PyBOP` enables parameter identification from data for various battery models, including the electrochemical and equivalent circuit models provided by the popular open-source `PyBaMM` package (Sulzer et al., 2021). Using the same approaches, `PyBOP` can also be used for design optimisation under user-defined operating conditions across a variety of model structures and design goals. `PyBOP` facilitates optimisation with a range of methods, providing diagnostics for examining optimiser performance and convergence of both the cost function and corresponding parameters. Identified parameters can be used for prediction, online estimation and control, and design optimisation, accelerating battery research and development.

Statement of need

`PyBOP` is a Python package that provides a user-friendly, object-oriented interface for optimising battery model parameters. `PyBOP` leverages the open-source `PyBaMM` package (Sulzer et al., 2021) to formulate and solve battery models. Together, these tools serve a broad audience including students, engineers, and researchers in academia and industry, enabling the use of advanced models where previously this was not possible without specialised knowledge of battery modelling, parameter inference, and software development. `PyBOP` emphasises clear and informative diagnostics and workflows to support users with varying levels of domain expertise, and provides access to a wide range of optimisation and sampling algorithms. These capabilities are enabled through interfaces to `PINTS` (Clerx et al., 2019), `SciPy` (Virtanen et al., 2020), and `PyBOP`'s own implementations of algorithms such as Adaptive Moment Estimation with Weight Decay (AdamW) (Loshchilov & Hutter, 2017), Gradient Descent (Cauchy & others, 1847), and Cuckoo Search (Yang & Suash Deb, 2009).

`PyBOP` complements other lithium-ion battery modelling packages built around `PyBaMM`, such as `liionpack` for battery pack simulation (Tranter et al., 2022) and `pybamm-eis` for fast numerical computation of the electrochemical impedance of any battery model.

Architecture

`PyBOP` formulates the inference process into three core classes: `Problem`, `Optimiser`, and `Sampler`, as shown in Figure 1. Each of these objects represents a base class with child classes that construct specialised functionality for different workflows. Management of these different workflows is achieved through a builder pattern for the `Problem` class, aiming to provide a robust, flexible interface. The `Problem` object offers the required functionality to compute the forward model, generate residuals if needed, and compute the corresponding cost for parameter value candidates. Multiple `Problem` builders are provided to construct optimisation `Problems` for both time-series and electrochemical impedance spectroscopy PyBaMM domains, alongside a pure Python problem for general optimisation. The builder structure allows for extensibility and flexibility when new optimisation problems are required without requiring refactoring of PyBOP's core classes. An example of this is the `MultiFittingProblem` and corresponding builder, which generalises the pure Python problem and builder for optimisation tasks where multiple objectives are minimised. One such common use case for the `MultiFittingProblem` is parameter identification workflows where the initial model state varies for each corresponding set of observations; however, many other use cases are available through this generalised interface. The syntax for building a `PyBaMM`-based parameter inference workflow is shown below.

```
builder = (
    pybop.builders.Pybamm()
    .set_dataset(dataset)
    .set_simulation(model, parameter_values=parameter_values)
    .add_parameter(pybop.Parameter("Negative electrode thickness [m]"))
    .add_cost(pybop.costs.pybamm.SumSquaredError("Voltage [V]"))
)
problem = builder.build()
optim = pybop.CMAES(problem)
result = optim.run()
```

The `PyBaMMProblem` class interfaces with the `PyBaMM` forward solution via a composed `Pipeline` object that manages the `PyBaMM` model, including initial state calculation, discretising and meshing the model, and exception handling with mocks for improved convergence. This `Pipeline` provides a singular interface for `PyBOP` to manage the `PyBaMM` simulation, allowing the `Problem` and optimisation classes to be agnostic to the various contexts required to acquire the forward solution. In addition, the `Pipeline` also manages multiprocessing for the `PyBaMMProblem`, as this is completed within the numerical C++ solver. The `Optimiser` and `Sampler` classes orchestrate the parameter inference process through either optimisation algorithms or Monte Carlo sampling. For the `PyBaMM`-based `Problem` classes, `PyBOP` constructs the cost functions as a PyBaMM expression, which is applied to the user-provided model after a defensive copy is performed. This implementation allows the cost to be computed alongside the forward solution, with gradient information available through `PyBaMM`'s automatic differentiation capabilities. Custom cost definitions are supported through a generalised `UserCost` subclass.

Furthermore, for PyBaMM-based builders, PyBOP supports user-provided PyBaMM models for optimisation and parameter inference workflows. This allows users to use both canonical models offered by PyBaMM and custom formulations with PyBOP's optimisation methods. Under these conditions, PyBOP aims to apply the minimal number of modifications to the provided model in an effort to improve optimisation convergence, as well as corresponding goodness-of-fit criteria. One such example is spatial re-discretisation, which is required for the standard PyBaMM mesh construction for optimisation of geometric parameters. In this situation, `PyBOP` rebuilds the `PyBaMM` model only when necessary, aiming to limit the effect on workflow performance. In addition to the convergence information, identifiability metrics are provided with the correspondingly estimated parameter values through Hessian approximation, as well as Sobol sampling from the `salib` package.

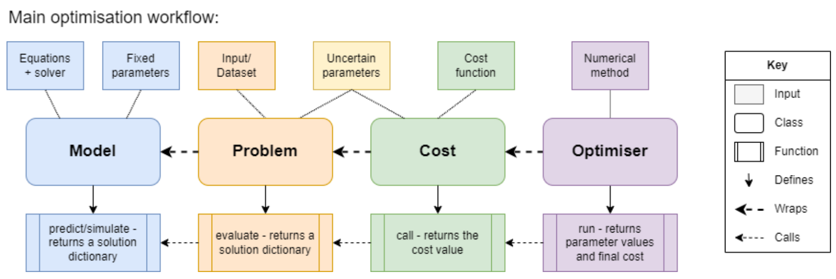
Main optimisation workflow:
Equations + solver
Fixed parameters
Input/ Dataset
Uncertain parameters
Cost function
Numerical method
Model
Problem
Cost
Optimiser
predict/simulate - returns a solution dictionary
evaluate - returns a solution dictionary
call - returns the cost value
run - returns parameter values and final cost
Key
Input
Class
Function
Defines
Wraps
Calls

Figure 1: The core PyBOP architecture with base class interfaces. Each class provides a direct mapping to a step in the optimisation workflow.

Alongside construction of the simulation process, the `Pipeline` object provides methods for obtaining sensitivities from the prediction, enabling gradient-based optimisation. A forward prediction, along with its corresponding sensitivities, is provided to the problem class for processing and exception control. A standardised data structure is then provided to the cost classes, which compute a distance, design, or likelihood-based metric for optimisation. For point-based optimisation, the optimisers minimise the cost function or the negative log-likelihood if a likelihood class is provided. Bayesian inference is provided by sampler classes, which accept the `LogPosterior` class and sample from it using `PINTS`-based Monte Carlo algorithms. In the typical workflow, the classes in Figure 1 are constructed in sequence, from left to right in the figure.

In addition to the core architecture, `PyBOP` provides several specialised inference and optimisation features. One example is parameter inference from electrochemical impedance spectroscopy (EIS) simulations, where PyBOP discretises and linearises the EIS forward model into a sparse mass matrix form with an accompanying auto-differentiated Jacobian. This is then translated into the frequency domain, providing a direct solution to compute the input-output impedance. In this situation, the forward models are constructed within the spatial re-discretisation workflow, allowing for geometric parameter inference from EIS simulations and data. The currently implemented cost classes are listed in Table 1.

Table 1: List of default cost (or likelihood) classes.

Error Measures / Likelihoods	Design Metrics
Sum-squared error	Volumetric energy density
Root-mean-squared error	Gravimetric energy density
Minkowski	
Sum-of-power	
Gaussian log likelihood	
Maximum a Posteriori	

Similarly, the current optimisation algorithms are presented in Table 2. It should be noted that SciPy minimize includes several gradient-based and gradient-free methods. From here on, the point-based parameterisation and design-optimisation tasks will simply be referred to as optimisation tasks. This simplification can be justified by comparing Equation 5 and Equation 7; deterministic parameterisation is simply an optimisation task to minimise a distance-based cost between model output and measured values.

Table 2: Currently supported optimisers classified by candidate solution type, including gradient information.

Gradient-based	Evolutionary	(Meta)heuristic
Weight decayed adaptive moment estimation (AdamW)	Covariance matrix adaptation (CMA-ES)	Particle swarm (PSO)
Improved resilient backpropagation (iRProp-)	Exponential natural (xNES)	Nelder-Mead
Gradient descent	Separable natural (sNES)	Cuckoo search
SciPy minimize	SciPy differential evolution	Simulated Annealing

In addition to deterministic optimisers (Table 2), `PyBOP` also provides Monte Carlo sampling routines to estimate distributions of parameters within a Bayesian framework. These methods construct a posterior parameter distribution that can be used to assess uncertainty and practical identifiability. The individual sampler classes are currently composed within `PyBOP` from the `PINTS` library, with a base sampler class implemented for interoperability and direct integration with `PyBOP`'s model, problem, and likelihood classes. The currently supported samplers are listed in Table 3.

Table 3: Sampling methods supported by `PyBOP`, classified according to the candidate proposal method.

Gradient-based	Adaptive	Slicing	Evolutionary	Other
Monomial gamma	Delayed rejection adaptive	Rank shrinking	Differential evolution	Metropolis random walk
No-U-turn	Haario Bardenet	Doubling		Emcee hammer
Hamiltonian	Haario	Stepout		Metropolis adjusted Langevin
Relativistic	Rao Blackwell			

Background

Battery models

In general, battery models (after spatial discretisation) can be written in the form of a differential-algebraic system of equations,

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f(t, \mathbf{x}, \boldsymbol{\theta}), \tag{1}$$

$$0 = g(t, \mathbf{x}, \boldsymbol{\theta}), \tag{2}$$

$$\mathbf{y}(t) = h(t, \mathbf{x}, \boldsymbol{\theta}), \tag{3}$$

with initial conditions

$$\mathbf{x}(0) = \mathbf{x}_0(\boldsymbol{\theta}). \tag{4}$$

Here, t is time, $\mathbf{x}(t)$ are the (spatially discretised) states, $\mathbf{y}(t)$ are the outputs (e.g., the terminal voltage) and $\boldsymbol{\theta}$ are the unknown parameters.

Common battery models include various types of equivalent circuit models (e.g., the Thévenin model), the Doyle–Fuller–Newman (DFN) model (Doyle et al., 1993; Fuller et al., 1994) based on porous electrode theory, and its reduced-order variants including the single particle model (SPM) (Brosa Planella et al., 2022) and the multi-species multi-reaction (MSMR) model (Verbrugge et al., 2017). Simplified models that retain acceptable predictive accuracy at lower computational cost are widely used, for example, in battery management systems, while physics-based models are required to understand the impact of physical parameters on performance. This separation of complexity traditionally results in multiple parameterisations for a single battery type, depending on the model structure.

Examples

Parameterisation

The parameterisation of battery models is challenging due to the large number of parameters that need to be identified compared to the number of measurable outputs (Andersson et al., 2022; Miguel et al., 2021; Wang et al., 2022). A complete parameterisation often requires stepwise identification of smaller sets of parameters from a variety of excitations and different datasets (Chen et al., 2020; Chu et al., 2019; Kirk et al., 2023; Lu et al., 2021). Furthermore, parameter identifiability can be poor for a given set of excitations and datasets, requiring improved experimental design in addition to uncertainty-capable identification methods (Aitio et al., 2020).

A generic data-fitting optimisation problem may be formulated as:

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}_{(\hat{\mathbf{y}}_i)}(\boldsymbol{\theta}) \quad \text{subject to equations (1)-(4)} \tag{5}$$

where $\mathcal{L} : \boldsymbol{\theta} \mapsto [0, \infty)$ is a cost function that quantifies the agreement between the model output $\mathbf{y}(t)$ and a sequence of observations $(\hat{\mathbf{y}}_i)$ measured at times t_i. Within the PyBOP framework, the FittingProblem class packages the model output along with the measured observations, both of which are then passed to the cost classes for computation of the specific cost function. For gradient-based optimisers, the Jacobian of the cost function with respect to unknown parameters, $\partial \mathcal{L} / \partial \theta$, is computed for step-size and directional information.

Next, we demonstrate the fitting of synthetic data where the model parameters are known. Throughout this section, as an example, we use PyBaMM's implementation of the single particle model with an added contact resistance submodel. We assume that the model is already fully parameterised apart from two parameters, namely, the lithium diffusivity of the negative electrode active material particles (denoted "negative particle diffusivity") and the contact resistance, with corresponding true values of [3.3e-14 m^2/s, 10 mΩ]. To start, we generate synthetic time-domain data corresponding to a one-hour discharge from 100% to 0% state of charge, denoted as 1C rate, followed by 30 minutes of relaxation. This dataset is then corrupted with zero-mean Gaussian noise of amplitude 2 mV, with the resulting signal shown by the blue dots in Figure 2 (left). The initial states are assumed known, although this assumption is not generally necessary. The PyBOP repository contains several other example notebooks that follow a similar inference process. The underlying cost landscape to be explored by the optimiser is shown in Figure 2 (right), with the initial position denoted alongside the known true system parameters for this synthetic inference task. In general, the true parameters are not known.

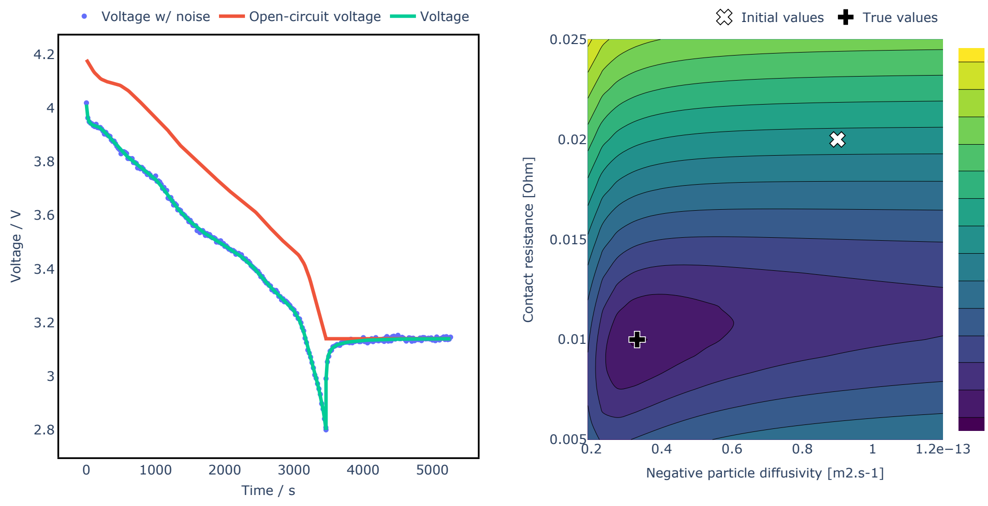

Figure 2: The synthetic fitting dataset (left) and cost landscape (right) for an example time-series battery model parameterisation using a root-mean-squared error cost function.

We can also use PyBOP to generate and fit electrochemical impedance data using methods within pybamm-eis that enable fast impedance computation of battery models (Dhoot et al., 2024). Using the same model and parameters as in the time-domain case, Figure 3 shows the numerical impedance prediction available in PyBOP alongside the cost landscape for the corresponding inference task. At the time of publication, gradient-based optimisation and sampling methods are not available when using an impedance workflow.

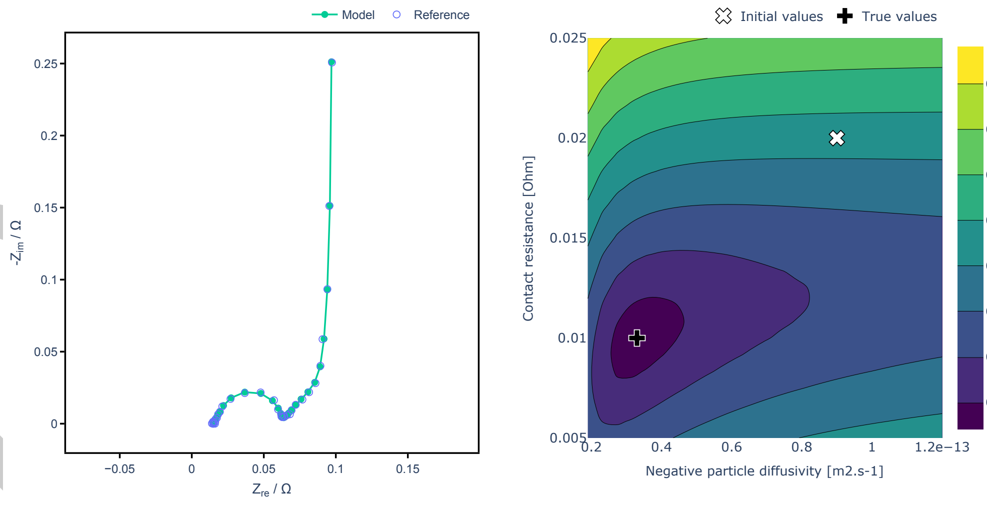

Figure 3: The data and model fit (left) and cost landscape (right) for a frequency-domain impedance parameterisation with a root-mean-squared error cost function, at 5% SOC.

To avoid confusion, in the remainder of this section, we continue with identification in the time domain (Figure 2). In general, however, time- and frequency-domain models and data may be combined for improved parameterisation. As gradient information is available for our time-domain example, the choice of distance-based cost function and optimiser is not constrained. Due to the difference in magnitude between the two parameters, we apply the logarithmic parameter transformation offered by PyBOP. This transforms the search space of the optimiser to allow for a common step size between the parameters, improving convergence in this particular case. As a demonstration of the parameterisation capabilities of PyBOP, Figure 4 (left) shows the rate of convergence for each of the distance-minimising cost functions, while Figure 4 (right) shows analogous results for maximising a likelihood. The optimisation is performed with SciPy minimize using the gradient-based L-BFGS-B method.

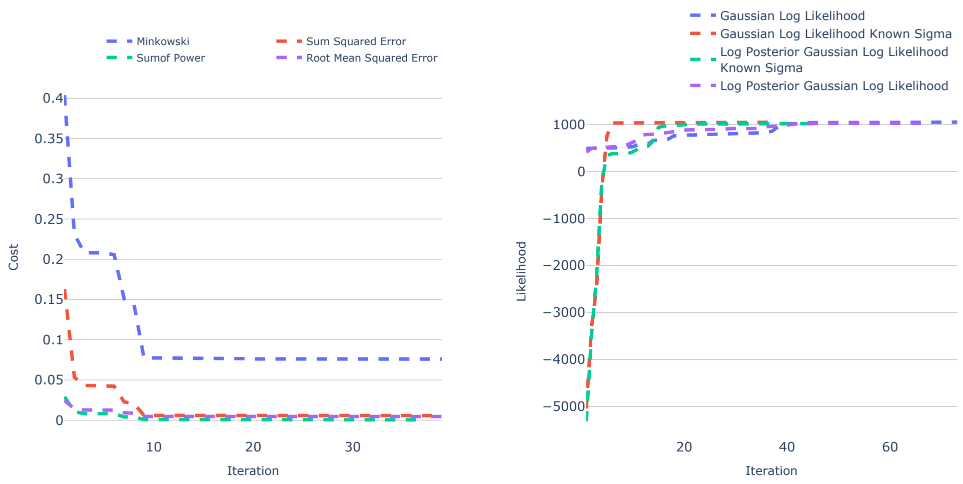

Figure 4: Optimiser convergence using various cost (left) and likelihood (right) functions and the L-BFGS-B algorithm.

Using the same model and parameters, we compare example convergence rates of various algorithms across several categories: gradient-based methods in Figure 5 (left), evolutionary strategies in Figure 5 (middle) and (meta)heuristics in Figure 5 (right) using a mean-squared-error cost function. We also show the cost function and optimiser iterations in Figure 6, with the three rows showing the gradient-based optimisers (top), evolution strategies (middle), and (meta)heuristics (bottom). Note that the performance of the optimiser depends on the cost landscape, the initial guess or prior, and the hyperparameters for each specific problem.

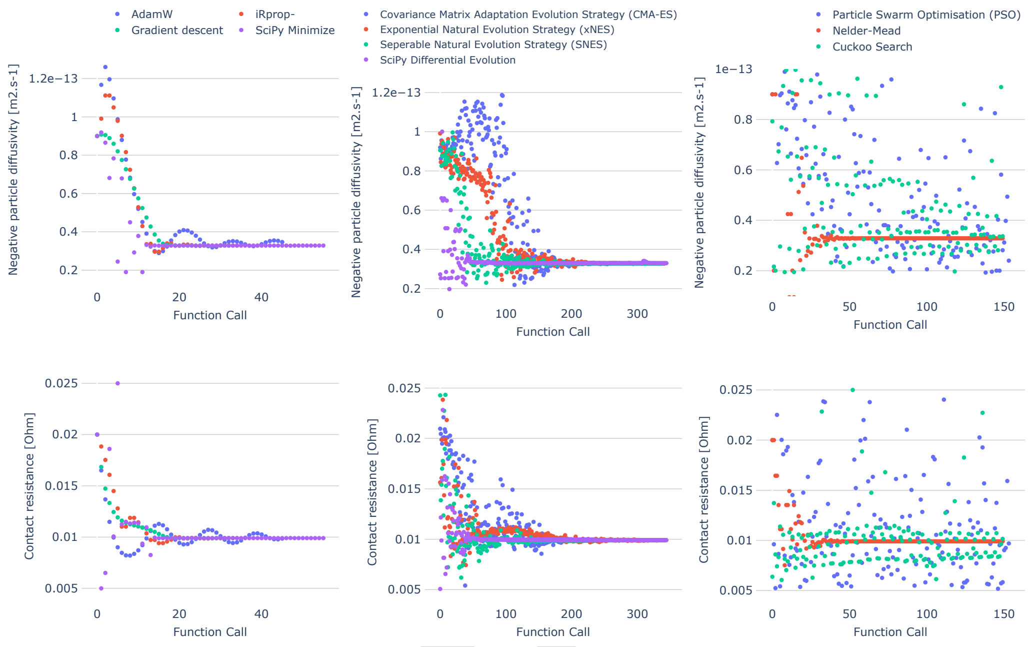

Figure 5: Convergence in parameter values for several optimisation algorithms provided by PyBOP.

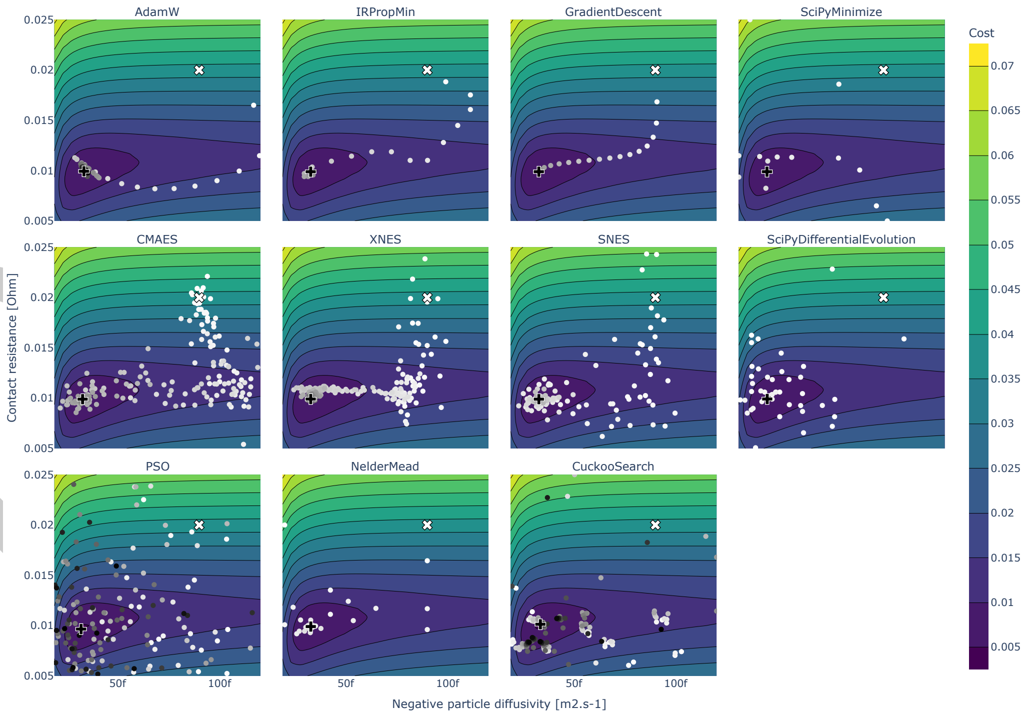

Figure 6: Cost landscape contour plot with corresponding optimisation traces, for several optimisers.

This example parameterisation task can also be approached from a Bayesian perspective, using PyBOP's sampler methods. First, we introduce Bayes' rule,

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}, \tag{6}$$

where $P(\theta|D)$ is the posterior parameter distribution, $P(D|\theta)$ is the likelihood function, $P(\theta)$ is the prior parameter distribution, and $P(D)$ is the model evidence, or marginal likelihood, which acts as a normalising constant. In the case of maximum likelihood estimation or maximum a posteriori estimation, one wishes to maximise $P(D|\theta)$ or $P(\theta|D)$, respectively, and this may be formulated as an optimisation problem as per Equation 5.

To estimate the full posterior parameter distribution, however, one must use sampling or other inference methods to reconstruct the function $P(\theta|D)$. The posterior distribution provides information about the uncertainty of the identified parameters, e.g., by calculating the variance or other moments. Monte Carlo methods are used here to sample from the posterior. The selection of Monte Carlo methods available in `PyBOP` includes gradient-based methods such as No-U-Turn (Hoffman & Gelman, 2011) and Hamiltonian (Brooks et al., 2011), as well as heuristic methods such as differential evolution (Braak, 2006), and also conventional methods based on random sampling with rejection criteria (Metropolis et al., 1953). `PyBOP` offers a sampler class that provides the interface to samplers, the latter being provided by the Probabilistic Inference on Noisy Time-series (`PINTS`) package. Figure 7 shows the sampled posteriors for the synthetic model described previously, using an adaptive covariance-based sampler called Haario Bardenet (Haario et al., 2001).

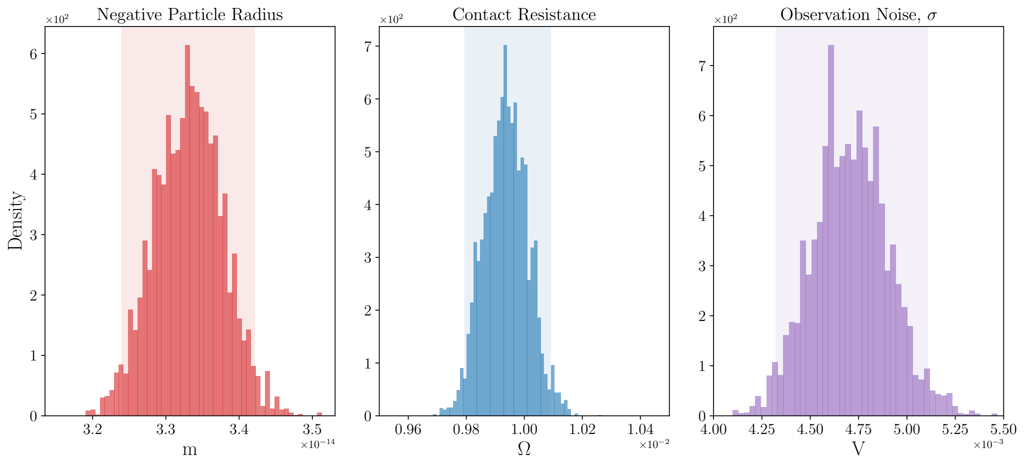

Figure 7: Posterior distributions of model parameters alongside identified noise on the observations. Shaded areas denote the 95th percentile credible interval for each parameter.

Design optimisation

Design optimisation is supported in `PyBOP` to guide device design development by identifying parameter sensitivities that can unlock improvements in performance. This problem can be viewed in a similar way to the parameterisation workflows described previously, but with the aim of maximising a design-objective cost function rather than minimising a distance-based cost function. `PyBOP` performs maximisation by minimising the negative of the cost function. In design problems, the cost metric is no longer a distance between two time series, but a metric evaluated on a model prediction. For example, to maximise the gravimetric energy (or power) density, the cost is the integral of the discharge energy (or power) normalised by the cell mass. Such metrics are typically quantified for operating conditions such as a 1C discharge at a given temperature.

In general, design optimisation can be written as a constrained optimisation problem,

$$\min_{\theta\in\Omega} \mathcal{L}(\theta) \quad \text{subject to equations (1)-(4)}, \tag{7}$$

where $\mathcal{L} : \theta \mapsto [0, \infty)$ is a cost function that quantifies the desirability of the design and Ω is the set of allowable parameter values.

As an example, we consider the challenge of maximising the gravimetric energy density, subject to constraints on two of the geometric electrode parameters (Couto et al., 2023). In this case, we use the PyBaMM implementation of the single particle model with electrolyte (SPMe) to investigate the impact of the positive electrode thickness and the active material volume fraction on the energy density. Since the total volume fraction must sum to unity, the positive electrode porosity for each optimisation iteration is defined in relation to the active material volume fraction. It is also possible to update the 1C rate corresponding to the theoretical capacity for each iteration of the design.

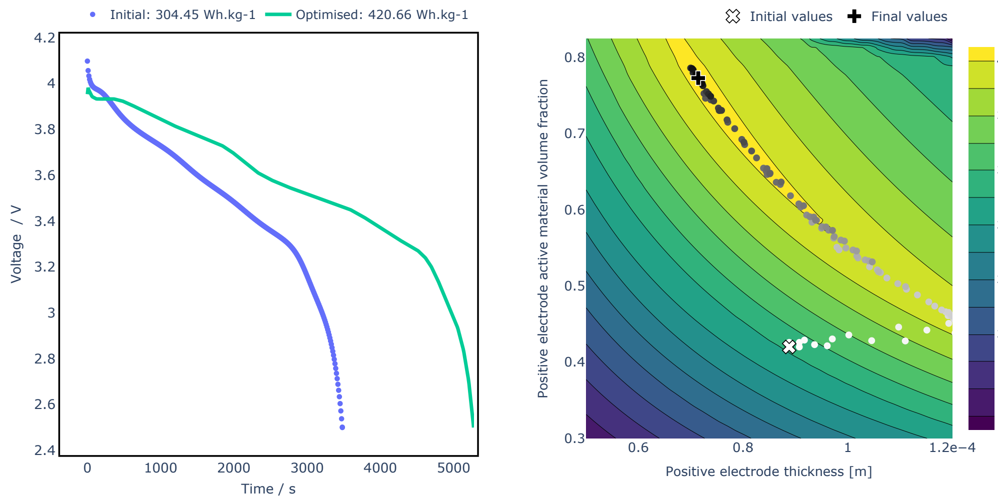

Figure 8: Initial and optimised voltage profiles alongside the gravimetric energy density cost landscape.

Figure 8 (left) shows the predicted improvement in the discharge profile between the initial and optimised parameter values for a fixed-rate 1C discharge selected from the initial design and (right) the Nelder-Mead search over the parameter space.

Acknowledgements

We gratefully acknowledge all contributors to PyBOP. This work was supported by the Faraday Institution Multiscale Modelling project (ref. FIRG059), UKRI's Horizon Europe Guarantee (ref. 10038031), and EU IntelLiGent project (ref. 101069765).

References

Aitio, A., Marquis, S. G., Ascencio, P., & Howey, D. (2020). Bayesian parameter estimation applied to the li-ion battery single particle model with electrolyte dynamicsa this work was carried out with funding received from the faraday institution (faraday.ac.uk; EP/S003053/1, ref. FIRG003). Scott marquis was supported by the EPSRC centre for doctoral training in industrially focused mathematical modelling (EP/L015803/1) in collaboration with siemens corporate technology. *IFAC-PapersOnLine*, *53*(2), 12497–12504. https://doi.org/https://doi.org/10.1016/j.ifacol.2020.12.1770

Andersson, M., Streb, M., Ko, J. Y., Löfqvist Klass, V., Klett, M., Ekström, H., Johansson, M., & Lindbergh, G. (2022). Parametrization of physics-based battery models from input-output data: A review of methodology and current research. *Journal of Power Sources*, *521*(November 2021), 230859. https://doi.org/10.1016/j.jpowsour.2021.230859

Braak, C. J. F. T. (2006). A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: Easy Bayesian computing for real parameter spaces. *Statistics and Computing*, *16*(3), 239–249. https://doi.org/10.1007/s11222-006-8769-1

Brooks, S., Gelman, A., Jones, G., & Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. Chapman; Hall/CRC. https://doi.org/10.1201/b10905

Brosa Planella, F., Ai, W., Boyce, A. M., Ghosh, A., Korotkin, I., Sahu, S., Sulzer, V., Timms, R., Tranter, T. G., Zyskin, M., Cooper, S. J., Edge, J. S., Foster, J. M., Marinescu, M., Wu, B., & Richardson, G. (2022). A Continuum of Physics-Based Lithium-Ion Battery Models Reviewed. *Progress in Energy*, *4*(4), 042003. https://doi.org/10.1088/2516-1083/ac7d31

Cauchy, A., & others. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, *25*(1847), 536–538.

Chen, C.-H., Brosa Planella, F., O'Regan, K., Gastol, D., Widanage, W. D., & Kendrick, E. (2020). Development of experimental techniques for parameterization of multi-scale lithium-ion battery models. *Journal of The Electrochemical Society*, *167*(8), 080534. https://doi.org/10.1149/1945-7111/ab9050

Chu, Z., Plett, G. L., Trimboli, M. S., & Ouyang, M. (2019). A control-oriented electrochemical model for lithium-ion battery, Part I: Lumped-parameter reduced-order model with constant phase element. *Journal of Energy Storage*, *25*(August), 100828. https://doi.org/10.1016/j.est.2019.100828

Clerx, M., Robinson, M., Lambert, B., Lei, C. L., Ghosh, S., Mirams, G. R., & Gavaghan, D. J. (2019). Probabilistic inference on noisy time series (PINTS). *Journal of Open Research Software*, *7*(1), 23. https://doi.org/10.5334/jors.252

Couto, L. D., Charkhgard, M., Karaman, B., Job, N., & Kinnaert, M. (2023). Lithium-ion battery design optimization based on a dimensionless reduced-order electrochemical model. *Energy*, *263*(PE), 125966. https://doi.org/10.1016/j.energy.2022.125966

Dhoot, R., Timms, R., & Please, C. (2024). *PyBaMM EIS: Efficient linear algebra methods to determine li-ion battery behaviour* (Version 0.1.4). https://www.github.com/pybamm-team/pybamm-eis

Doyle, M., Fuller, T. F., & Newman, J. (1993). Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell. *Journal of The Electrochemical Society*, *140*(6), 1526–1533. https://doi.org/10.1149/1.2221597

Fuller, T. F., Doyle, M., & Newman, J. (1994). Simulation and optimization of the dual lithium ion insertion cell. *Journal of The Electrochemical Society*, *141*(1), 1. https://doi.org/10.1149/1.2054684

Haario, H., Saksman, E., & Tamminen, J. (2001). An Adaptive Metropolis Algorithm. *Bernoulli*, *7*(2), 223. https://doi.org/10.2307/3318737

Hoffman, M. D., & Gelman, A. (2011). *The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo*. https://arxiv.org/abs/1111.4246

Kirk, T. L., Lewis-Douglas, A., Howey, D., Please, C. P., & Jon Chapman, S. (2023). Nonlinear electrochemical impedance spectroscopy for lithium-ion battery model parameterization. *Journal of The Electrochemical Society*, *170*(1), 010514. https://doi.org/10.1149/1945-7111/acada7

Loshchilov, I., & Hutter, F. (2017). *Decoupled Weight Decay Regularization*. arXiv. https:

//doi.org/10.48550/ARXIV.1711.05101

Lu, D., Scott Trimboli, M., Fan, G., Zhang, R., & Plett, G. L. (2021). Implementation of a physics-based model for half-cell open-circuit potential and full-cell open-circuit voltage estimates: Part II. Processing full-cell data. *Journal of The Electrochemical Society*, *168*(7), 070533. https://doi.org/10.1149/1945-7111/ac11a5

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114

Miguel, E., Plett, G. L., Trimboli, M. S., Oca, L., Iraola, U., & Bekaert, E. (2021). Review of computational parameter estimation methods for electrochemical models. *Journal of Energy Storage*, *44*(PB), 103388. https://doi.org/10.1016/j.est.2021.103388

Sulzer, V., Marquis, S. G., Timms, R., Robinson, M., & Chapman, S. J. (2021). Python Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, *9*(1), 14. https://doi.org/10.5334/jors.309

Tranter, T. G., Timms, R., Sulzer, V., Planella, F. B., Wiggins, G. M., Karra, S. V., Agarwal, P., Chopra, S., Allu, S., Shearing, P. R., & Brett, D. J. I. (2022). Liionpack: A python package for simulating packs of batteries with PyBaMM. *Journal of Open Source Software*, *7*(70), 4051. https://doi.org/10.21105/joss.04051

Verbrugge, M., Baker, D., Koch, B., Xiao, X., & Gu, W. (2017). Thermodynamic model for substitutional materials: Application to lithiated graphite, spinel manganese oxide, iron phosphate, and layered nickel-manganese-cobalt oxide. *Journal of The Electrochemical Society*, *164*(11), E3243. https://doi.org/10.1149/2.0341708jes

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wang, A. A., O'Kane, S. E. J., Brosa Planella, F., Houx, J. L., O'Regan, K., Zyskin, M., Edge, J., Monroe, C. W., Cooper, S. J., Howey, D. A., Kendrick, E., & Foster, J. M. (2022). Review of parameterisation and a novel database (LiionDB) for continuum Li-ion battery models. *Progress in Energy*, *4*(3), 032004. https://doi.org/10.1088/2516-1083/ac692c

Yang, X.-S., & Suash Deb. (2009). Cuckoo Search via levy flights. *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, 210–214. https://doi.org/10.1109/NABIC.2009.5393690