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Published: unpublished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 using both deterministic and stochastic approaches with example workflows. `PyBOP` enables parameter identification from data for various battery models, including electrochemical and equivalent circuit models from the open-source `PyBaMM` package (Sulzer et al., 2021). The same approaches enable design optimisation under user-defined operating conditions across various model structures and design goals. `PyBOP` facilitates optimisation with multiple methods, providing diagnostics for examining optimiser performance and convergence of cost functions and parameters. Identified parameters can be used for prediction, online estimation, control, and design optimisation, accelerating battery research and development.

Statement of need

`PyBOP` provides a user-friendly, object-oriented interface for optimising battery model parameters. It leverages the open-source `PyBaMM` package (Sulzer et al., 2021) to formulate and solve battery models. Together, these tools serve students, engineers, and researchers in academia and industry, enabling advanced model use without specialised knowledge of battery modelling, parameter inference, and software development. `PyBOP` emphasises clear diagnostics and workflows to support users with varying domain expertise, providing access to numerous optimisation and sampling algorithms. These capabilities are enabled through interfaces to `PINTS` (Clerx et al., 2019), `SciPy` (Virtanen et al., 2020), and `PyBOP`'s implementations of algorithms including 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`, including `liionpack` for battery pack simulation (Tranter et al., 2022) and `pybamm-eis` for fast electrochemical impedance computation.

Architecture

`PyBOP` formulates the inference process into four core architectural components: `Builder`, `Pipeline`, `Problem`, and `Optimiser`/`Sampler`, as shown in Figure 1. `Builder` classes construct optimisation problems using a fluent interface, `Pipeline` classes manage simulation execution, `Problem` classes coordinate cost evaluation, and `Optimiser`/`Sampler` classes perform parameter inference. Each component represents a base class with child classes providing specialised functionality for different workflows.

The enhanced builder pattern provides a robust interface for constructing optimisation problems. The BaseBuilder class defines a common interface with methods including set_dataset(), add_parameter(), and add_cost(), enabling method chaining. Specialised builders (Pybamm, PybammEIS, Python, MultiFitting) extend this base functionality for specific use cases. This structure ensures extensibility for new optimisation problems without refactoring PyBOP's core classes. Multiple costs can be added with automatic weighting, and the builder validates requirements before constructing the final problem instance. The syntax for building a PyBaMM-based parameter inference workflow is shown below.

```python
# Builder pattern with extendable interface
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]"))
)

# Build and run inference
problem = builder.build()
optim = pybop.CMAES(problem)
result = optim.run()
```

A key architectural enhancement is explicit Pipeline classes that encapsulate simulation logic separate from problem coordination. The PybammPipeline manages PyBaMM model execution, including model building, discretisation, parameter updates, and sensitivity calculations. This separation allows Problem classes to focus on cost evaluation and optimisation coordination while Pipeline classes handle simulation execution complexities. The pipeline architecture provides a consistent interface between PyBOP and underlying simulation engines, facilitating future extensions to other modelling frameworks.

The Problem classes follow a clean hierarchy with Problem as the base class providing run() and run_with_sensitivities() methods. PybammProblem coordinates between PybammPipeline instances and cost evaluation, supporting multiple weighted costs and automatic hyperparameter handling for Bayesian inference. Problem classes are agnostic to simulation details, handled by their associated pipeline instances. This architecture enables consistent interfaces across different simulation backends while maintaining flexibility for specialised optimisation workflows. The Optimiser and Sampler classes orchestrate parameter inference through optimisation algorithms or Monte Carlo sampling, interfacing with problem classes through standardised methods.

For PyBaMM-based builders, PyBOP supports user-provided PyBaMM models for optimisation and parameter inference workflows. This allows users to employ both canonical PyBaMM models and custom formulations with PyBOP's optimisation methods. PyBOP applies minimal modifications to provided models to improve optimisation convergence and goodness-of-fit criteria. For example, spatial re-discretisation is required for standard PyBaMM mesh construction when optimising geometric parameters. PyBOP rebuilds the PyBaMM model only when necessary to limit performance impact. Beyond convergence information, identifiability metrics are provided with estimated parameter values through Hessian approximation and Sobol sampling from the salib package.

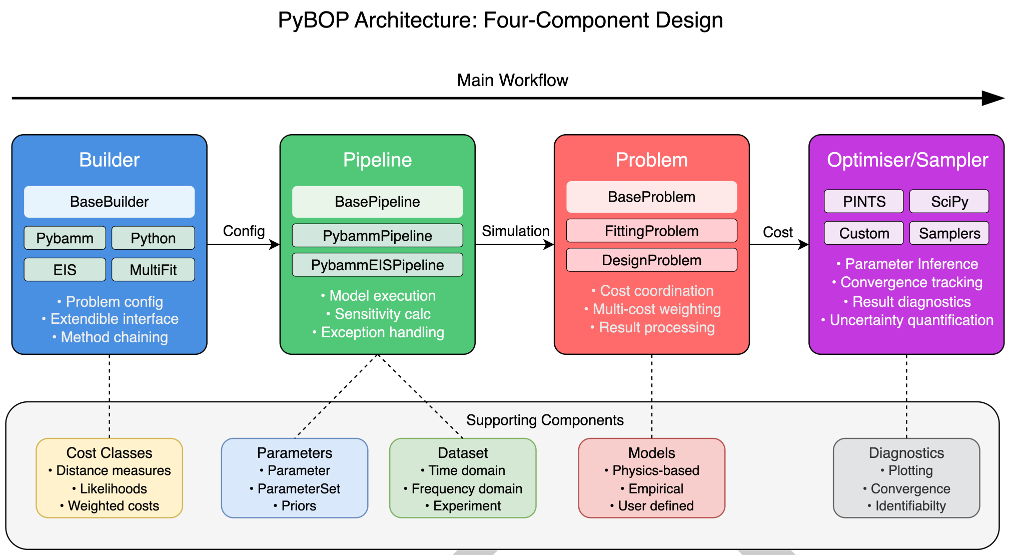

Figure 1: The core `PyBOP` architecture with four main components: Builder, Pipeline, Problem, and Optimiser/Sampler. Each component provides a direct mapping to a step in the optimisation workflow, with clear separation of concerns between construction, simulation, coordination, and inference.

The `Pipeline` object provides methods for obtaining sensitivities from predictions, enabling gradient-based optimisation. Forward predictions with corresponding sensitivities are provided to the problem class for processing and exception control. A standardised data structure is then provided to cost classes, which compute distance, design, or likelihood-based metrics for optimisation. The restructured cost system supports multiple costs with automatic weighting and metadata introspection. Cost classes can define hyperparameters automatically added to the optimisation problem, enabling seamless integration of likelihood-based methods with hyperparameter inference. Cost evaluation is cleanly separated from simulation execution, with costs computed from pipeline outputs rather than embedded in the simulation process.

For point-based optimisation, optimisers minimise the cost function or negative log-likelihood if a likelihood class is provided. Bayesian inference is provided by sampler classes, which accept the `LogPosterior` class and sample using `PINTS`-based Monte Carlo algorithms. In typical workflows, the classes in Figure 1 are constructed sequentially from left to right.

Beyond the core architecture, `PyBOP` provides specialised inference and optimisation features. Parameter inference from electrochemical impedance spectroscopy (EIS) simulations is handled through the `PybammEISPipeline`, which discretises and linearises the EIS forward model into sparse mass matrix form with an auto-differentiated Jacobian. The `PybammEIS` builder constructs problems for impedance-based parameter identification, with the pipeline managing frequency-domain transformations and impedance calculations. This architecture enables geometric parameter inference from EIS simulations while maintaining the same consistent interface as time-domain problems. Currently implemented cost classes are listed in Table 1.

Table 1: List of default cost 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	

Current optimisation algorithms are presented in Table 2. Note that SciPy minimize includes several gradient-based and gradient-free methods. Hereafter, point-based parameterisation and design-optimisation tasks are referred to as optimisation tasks. This simplification is justified by comparing Equation 5 and Equation 7; deterministic parameterisation is an optimisation task to minimise 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)
Gradient descent	Exponential natural (xNES)	Nelder-Mead
SciPy minimize	Separable natural (sNES)	Cuckoo search
Improved resilient backpropagation (iRProp-/+)	SciPy differential evolution	Simulated Annealing

Beyond deterministic optimisers (Table 2), `PyBOP` provides Monte Carlo sampling routines to estimate parameter distributions within a Bayesian framework. These methods construct posterior parameter distributions for assessing uncertainty and practical identifiability. Individual sampler classes are 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. 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,

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

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

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

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 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 reduced-order variants including the single particle model (SPM) (Brosa Planella et al., 2022) and multi-species multi-reaction (MSMR) model (Verbrugge et al., 2017). Simplified models retaining acceptable predictive accuracy at lower computational cost are widely used in battery management systems, while physics-based models are required to understand physical parameter impacts on performance. This complexity separation traditionally results in multiple parameterisations for a single battery type, depending on model structure.

Examples

Parameterisation

Battery model parameterisation is challenging due to the large number of parameters requiring identification compared to measurable outputs (Andersson et al., 2022; Miguel et al., 2021; Wang et al., 2022). Complete parameterisation often requires stepwise identification of smaller parameter sets from various excitations and datasets (Chen et al., 2020; Chu et al., 2019; Kirk et al., 2023; Lu et al., 2021). Parameter identifiability can be poor for given excitations and datasets, requiring improved experimental design and 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 quantifying agreement between model output $\mathbf{y}(t)$ and observations $(\hat{\mathbf{y}}_i)$ measured at times t_i. Within the PyBOP framework, the `FittingProblem` class packages model output with measured observations, passing both to cost classes for cost function computation. For gradient-based optimisers, the Jacobian $\partial\mathcal{L}/\partial\theta$ is computed for step-size and directional information.

We demonstrate fitting synthetic data with known model parameters. We use PyBaMM's single particle model with added contact resistance submodel. The model is fully parameterised except for two parameters: lithium diffusivity of negative electrode active material particles ("negative particle diffusivity") and contact resistance, with true values [3.3e-14 m^2/s, 10 mΩ]. We generate synthetic time-domain data for a one-hour discharge from 100% to 0% state of charge (1C rate) followed by 30 minutes relaxation. This dataset is corrupted with zero-mean Gaussian noise of 2 mV amplitude, shown as blue dots in Figure 2 (left). Initial states are assumed known, though this is not generally necessary. The PyBOP repository contains example notebooks following similar inference processes. The underlying cost landscape explored by the optimiser is shown in Figure 2 (right), with initial position and known true system parameters for this synthetic inference task. Generally, true parameters are unknown.

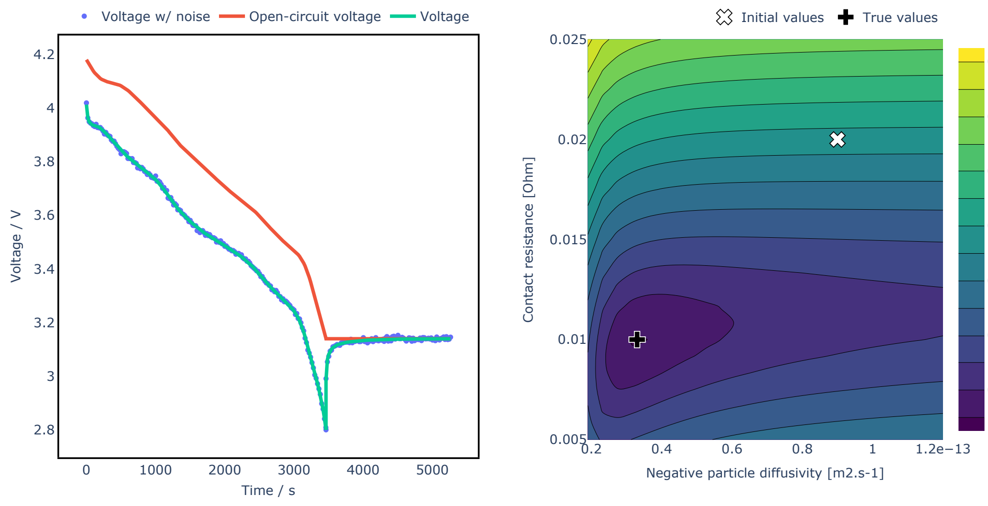

Figure 2: The synthetic fitting dataset (left) and cost landscape (right) for an example time-series battery model parameterisation using a root-mean-squared error cost function.

PyBOP can generate and fit electrochemical impedance data using `pybamm-eis` methods enabling fast impedance computation of battery models (Dhoot et al., 2024). Using the same model and parameters as the time-domain case, Figure 3 shows numerical impedance prediction available in PyBOP alongside the cost landscape for the corresponding inference task. At publication, gradient-based optimisation and sampling methods are unavailable for impedance workflows.

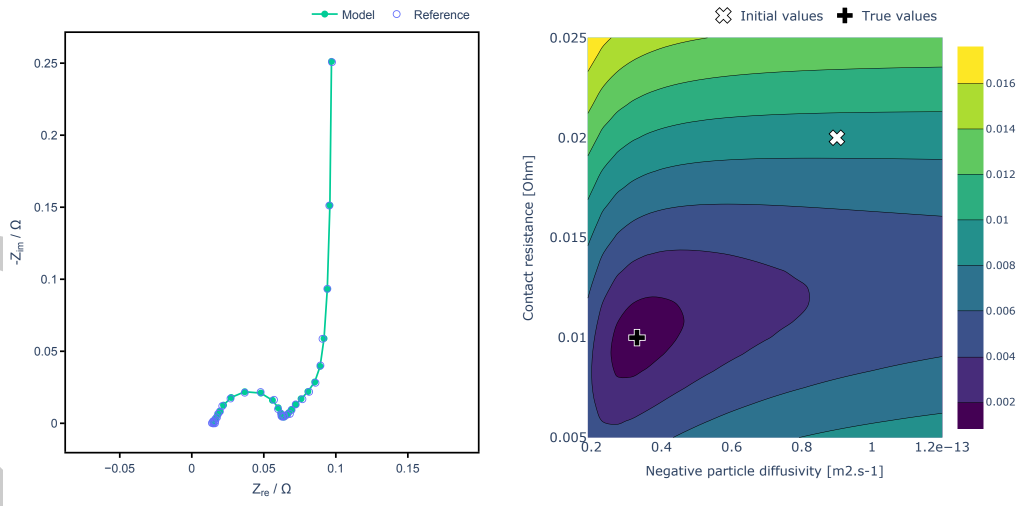

Figure 3: The data and model fit (left) and cost landscape (right) for a frequency-domain impedance parameterisation with a root-mean-squared error cost function, at 5% SOC.

We continue with time-domain identification (Figure 2). Generally, time- and frequency-domain models and data may be combined for improved parameterisation. As gradient information is available for our time-domain example, distance-based cost function and optimiser choice is unconstrained. Due to magnitude differences between parameters, we apply logarithmic parameter transformation offered by PyBOP. This transforms the optimiser search space to allow common step sizes between parameters, improving convergence. Demonstrating PyBOP's parameterisation capabilities, Figure 4 (left) shows convergence rates for distance-minimising cost functions, while Figure 4 (right) shows analogous results for likelihood maximisation.

Optimisation uses SciPy minimize with the gradient-based L-BFGS-B method.

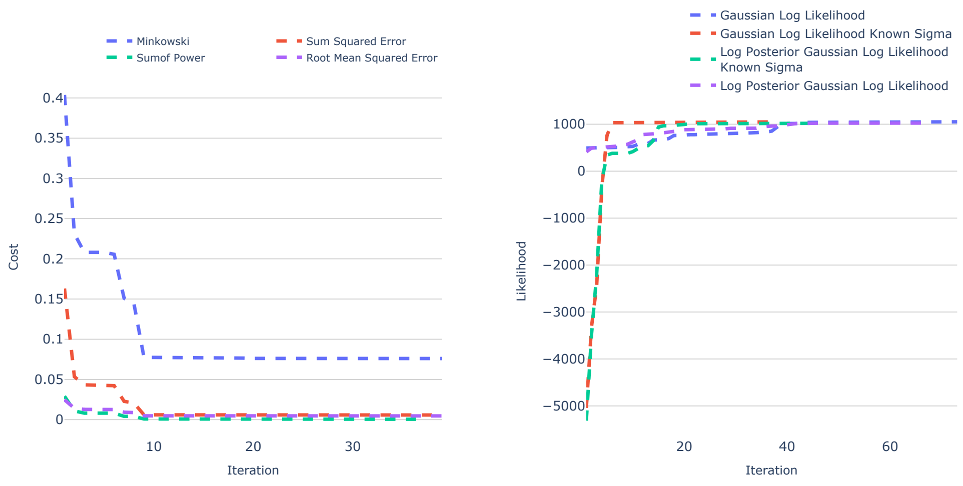

Figure 4: Optimiser convergence using various cost (left) and likelihood (right) functions and the L-BFGS-B algorithm.

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

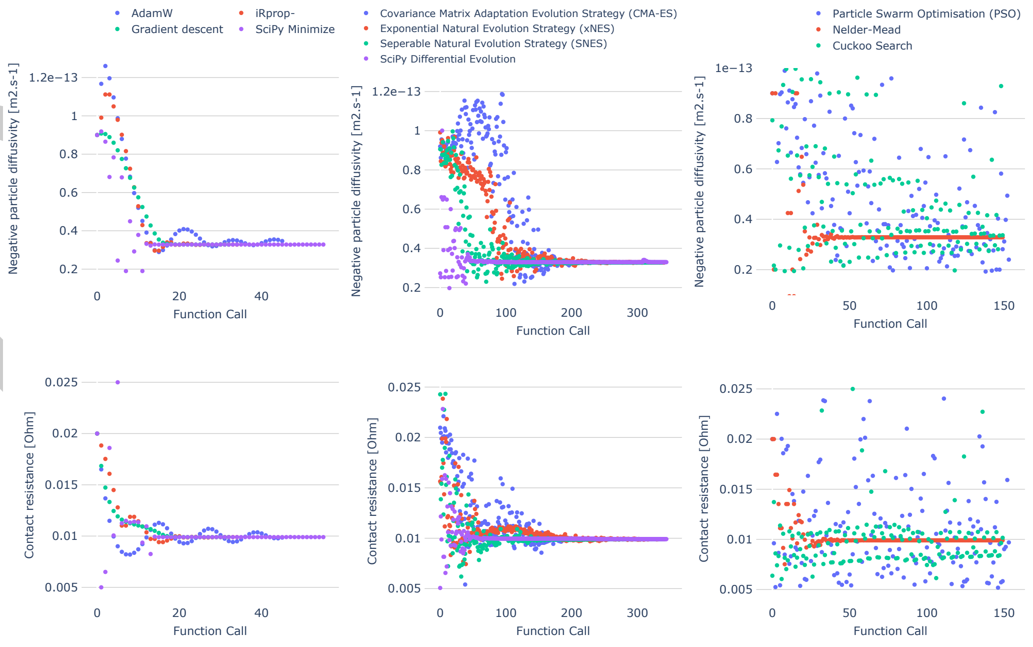

Figure 5: Convergence in parameter values for several optimisation algorithms provided by PyBOP.

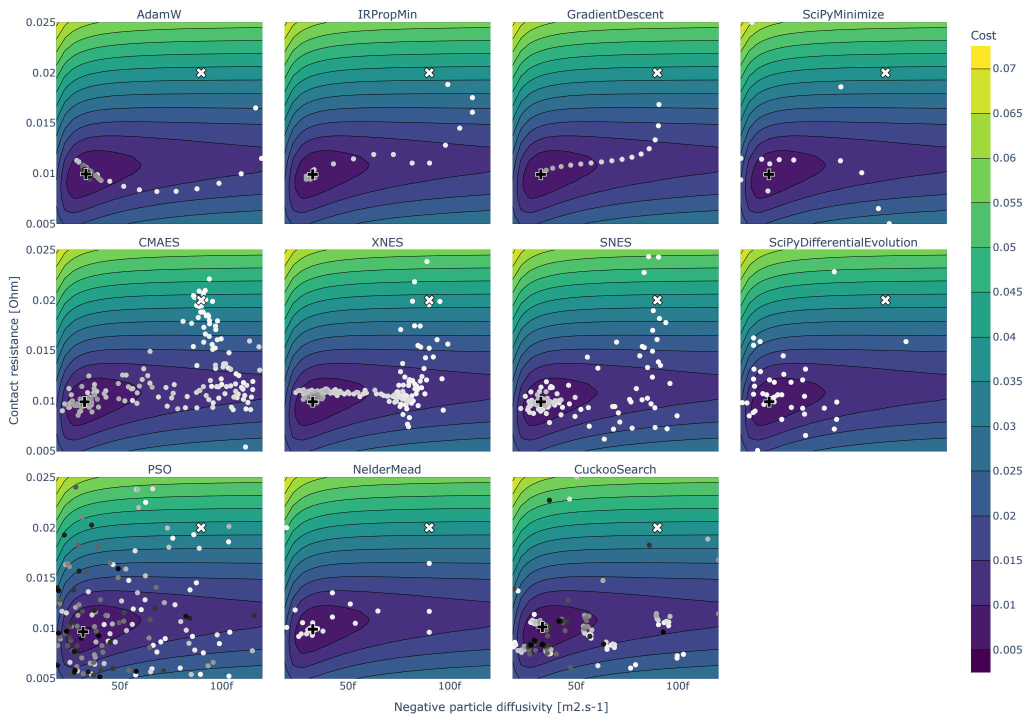

Figure 6: Cost landscape contour plot with corresponding optimisation traces, for several optimisers.

This parameterisation task can 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 acting as a normalising constant. For maximum likelihood estimation or maximum a posteriori estimation, one maximises $P(D|\theta)$ or $P(\theta|D)$, respectively, formulated as an optimisation problem per Equation 5.

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

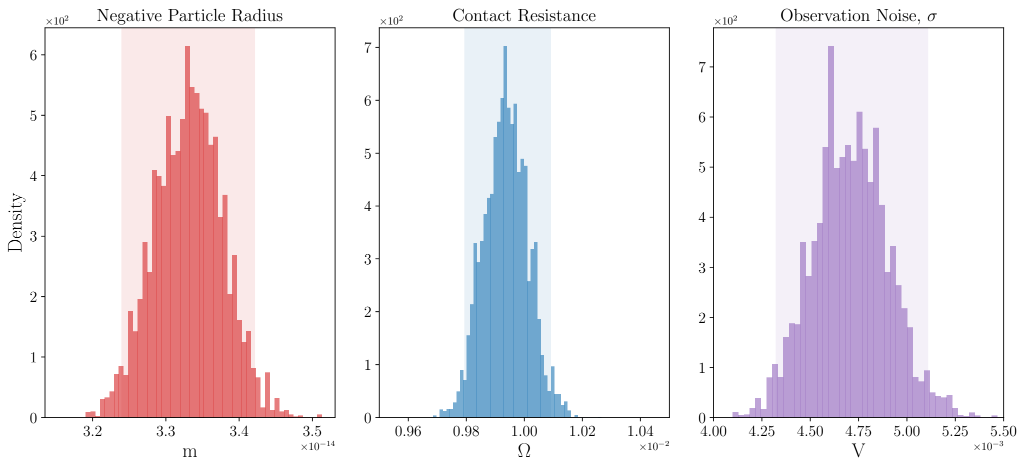

Figure 7: Posterior distributions of model parameters alongside identified noise on the observations. Shaded areas denote the 95th percentile credible interval for each parameter.

Design optimisation

PyBOP supports design optimisation to guide device design development by identifying parameter sensitivities that unlock performance improvements. This problem is similar to parameterisation workflows but aims to maximise a design-objective cost function rather than minimise a distance-based cost function. PyBOP performs maximisation by minimising the negative cost function. In design problems, the cost metric is no longer distance between time series, but a metric evaluated on model predictions. For example, to maximise gravimetric energy (or power) density, the cost is the integral of discharge energy (or power) normalised by cell mass. Such metrics are typically quantified for operating conditions like 1C discharge at 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 quantifying design desirability and Ω is the set of allowable parameter values.

We consider maximising gravimetric energy density subject to constraints on two geometric electrode parameters (Couto et al., 2023). We use the PyBaMM single particle model with electrolyte (SPMe) to investigate positive electrode thickness and active material volume fraction impacts on energy density. Since total volume fraction must sum to unity, positive electrode porosity for each optimisation iteration is defined relative to active material volume fraction. The 1C rate corresponding to theoretical capacity can be updated for each design iteration.

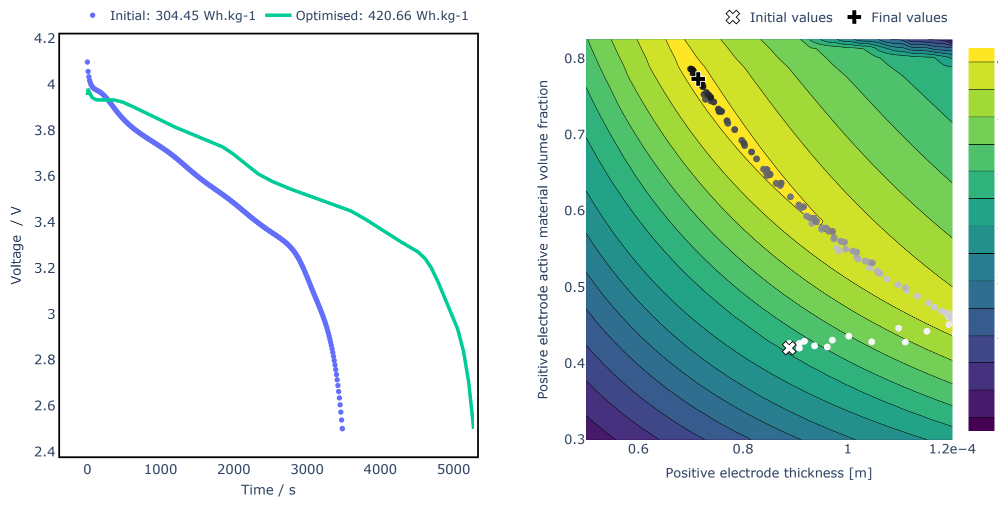

Figure 8: Initial and optimised voltage profiles alongside the gravimetric energy density cost landscape.

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

Acknowledgements

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