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 Labs **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Published: unpublished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 both the electrochemical and equivalent circuit models provided by the popular open-source `PyBaMM` package (Sulzer et al., 2021). Using the same workflow, `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 using a range of methods, with diagnostics for examining optimiser performance and convergence of the cost and corresponding parameters. Identified parameters can be used for prediction, on-line control, and design optimisation to accelerate research and development and improve battery utilisation.

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 packages 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 methods are provided 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), gradient descent, and Cuckoo search.

`PyBOP` supports the battery parameter exchange (BPX) standard (Korotkin et al., 2023) for sharing battery parameter sets. These parameter sets are typically costly to obtain due to the specialised equipment and time required for characterisation experiments, the need for battery domain knowledge, and the computational cost of parameter estimation. `PyBOP` reduces the requirements for the latter two categories by providing fast parameter estimation, standardised workflows, and parameter set interoperability (via BPX).

This package 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 for any battery model, since the identified parameters from PyBOP are easily exportable to these other packages.

Architecture

PyBOP has a layered structure designed to package up the necessary functionality to compute forward predictions, process the results, and run the optimisation and sampling algorithms. The forward model is solved using the popular battery modelling package, PyBaMM, with construction, parameterisation, and discretisation managed by PyBOP's model interface to PyBaMM. This provides a robust object construction process with a consistent interface between models and optimisers. The statistical methods and optimisation algorithms are constructed to interface cleanly with the forward model predictions. Furthermore, identifiability metrics are provided with the estimated parameters (through Hessian approximation of the cost functions in frequentist workflows, and posterior distributions in Bayesian workflows).

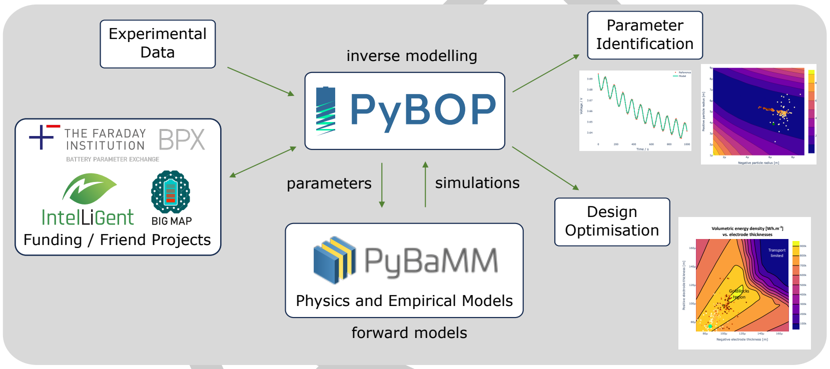

Figure 1: PyBOP's interface to supporting funding agencies, alongside a visualisation of the general workflow for parameterisation and optimisation.

PyBOP formulates the inference process into four key classes, namely, the model, the problem, the cost/likelihood, and the optimiser/sampler, as shown in Figure 2. Each of these objects represents a base class with child classes constructing specialised functionality for different workflows. The model class constructs a PyBaMM forward model for a given set of model equations, initial conditions, spatial discretisation, and numerical solver. By composing PyBaMM directly into PyBOP, specialised models can be constructed alongside the standard models, which can be modified and purposefully constructed for different inference tasks. One such example is spatial re-discretisation, which is required when one or more geometric parameters are being optimised. In this situation, PyBOP rebuilds the PyBaMM model a minimum number of times while maintaining the problem, cost, and optimiser objects, providing improved performance benefits to users. Alongside construction of the forward model, PyBOP's model class provides methods for obtaining sensitivities from the prediction, enabling gradient-based optimisation algorithms. 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 then computes a distance, design, or likelihood-based metric for optimisation. For deterministic 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 at the time of submission. In the typical workflow, the classes in Figure 2 are constructed in sequence.

In addition to the core architecture, PyBOP provides several specialised inference and optimisation processes. One such instance is parameter inference from electrochemical impedance spectroscopy (EIS) simulations. PyBOP discretises the EIS forward model into a sparse mass matrix form with accompanying auto-differentiated Jacobian. These objects are then translated into the frequency domain with a linear solution used to compute the battery model impedance. In this situation, the forward models are constructed within the spatial re-discretisation workflow, allowing for geometric parameter inference from EIS simulations. Furthermore, `PyBOP` builds on the `JAX` (Bradbury et al., 2018) numerical solvers provided by `PyBaMM` by providing `JAX`-based cost functions for automatic forward model differentiation with respect to the parameters. This functionality provides a performance improvement, and allows users to use many other JAX-based inference packages in order to optimise their cost function, such as `Numpyro` (Phan et al., 2019), `BlackJAX` (Cabezas et al., 2024), and `Optax` (DeepMind et al., 2020).

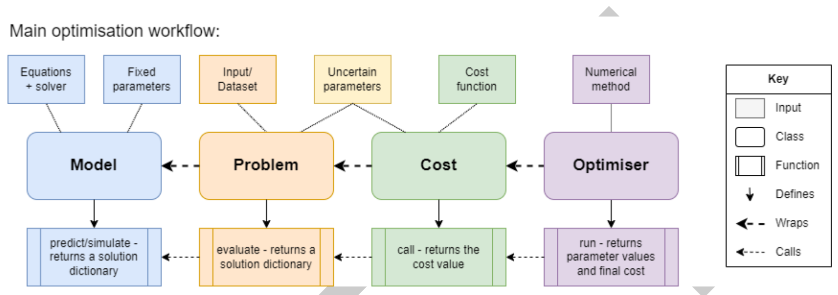
Main optimisation workflow:
Equations + solver
Fixed parameters
Input/ Dataset
Uncertain parameters
Cost function
Numerical method
Model
Problem
Cost
Optimiser
predict/simulate - returns a solution dictionary
evaluate - returns a solution dictionary
call - returns the cost value
run - returns parameter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

Figure 2: The core `PyBOP` architecture, showcasing the base class interfaces. Each class provides a direct mapping to a classical step in the optimisation workflow.

The currently implemented subclasses for the model, problem, and cost classes are listed in Table 1. The model and optimiser classes can be selected in combination with any problem-cost pair.

Table 1: List of available model, problem and cost (or likelihood) classes.

Battery Models	Problem Types	Cost / Likelihood Functions
Single particle model (SPM)	Fitting problem	Sum squared error
SPM with electrolyte (SPMe)	Design problem	Root mean squared error
Doyle-Fuller-Newman (DFN)	Observer	Minkowski
Many particle model (MPM)		Sum of power
Multi-species multi-reaction (MSMR)		Gaussian log likelihood
Weppner Huggins		Maximum a posteriori
Equivalent circuit model (ECM)		Volumetric energy density
		Gravimetric energy density
		Unscented Kalman filter

Similarly, the current algorithms available for optimisation tasks are presented in Table 2. It should be noted that `SciPy` minimize comprises a range of gradient and non-gradient methods. From now on, the point-based parameterisation and design optimisation tasks will simply be referred to as optimisation tasks. This simplification can be justified by examining Equation 5 and Equation 7 and confirming that deterministic parameterisation can be viewed as an optimisation task to minimise a distance-based cost function.

Table 2: The currently supported optimisation algorithms 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-based	Evolutionary	(Meta)heuristic
Improved resilient backpropagation (iRProp-)	Exponential natural (xNES)	Nelder-Mead
Gradient descent	Separable natural (sNES)	Cuckoo search
SciPy minimize	SciPy differential evolution	

As discussed above, in addition to point-based optimisation estimates Table 1, `PyBOP` also provides Monte Carlo sampling routines in order to recover distributions of likely parameter values. The Monte Carlo methods construct a posterior distribution on the inference parameters, which 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 the `PyBOP` 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 proposed 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 capabilities at a lower cost are widely used, for example in battery management systems, while physics-based models are required to understand the impact of physical parameters on battery 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 measurable outputs (Andersson et al., 2022; Miguel et al., 2021; Wang et al., 2022). A complete parameterisation often requires a stepwise identification of smaller sets of parameters from a variety of excitations and different data sets (Chen et al., 2020; Chu et al., 2019; Kirk et al., 2023; Lu et al., 2021).

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 the computation of the specific cost function. For gradient-based optimisers, the Jacobian of the cost function with respect to the unknown parameters, $(\frac{\partial \mathcal{L}}{\partial \theta})$ is computed for step size and directional information.

Next, we demonstrate the fitting of synthetic data where the system parameters are known. In this example problem, we use PyBaMM's implementation of the single particle model (SPM) with an added contact resistance submodel. We assume that the battery model is already parameterised except for two dynamic parameters, namely the lithium diffusivity of the negative electrode active material particle (denoted "negative particle diffusivity") and the contact resistance. We generate synthetic data from a one-hour discharge from 100% state of charge, to 0% (denoted as 1C rate), followed by 30 minutes of relaxation. This data is then corrupted with zero mean Gaussian noise of amplitude 2mV, shown by the dots in Figure 3 (left). The initial states are assumed known, although such an assumption is not generally necessary. The PyBOP repository contains multiple example notebooks that follow a similar inference process for readers that would like further information. The underlying cost landscape to be explored by the optimiser is shown in Figure 3 (right) with the initial position denoted alongside the known system parameters for this synthetic inference task. In general, the true parameters are not known a priori.

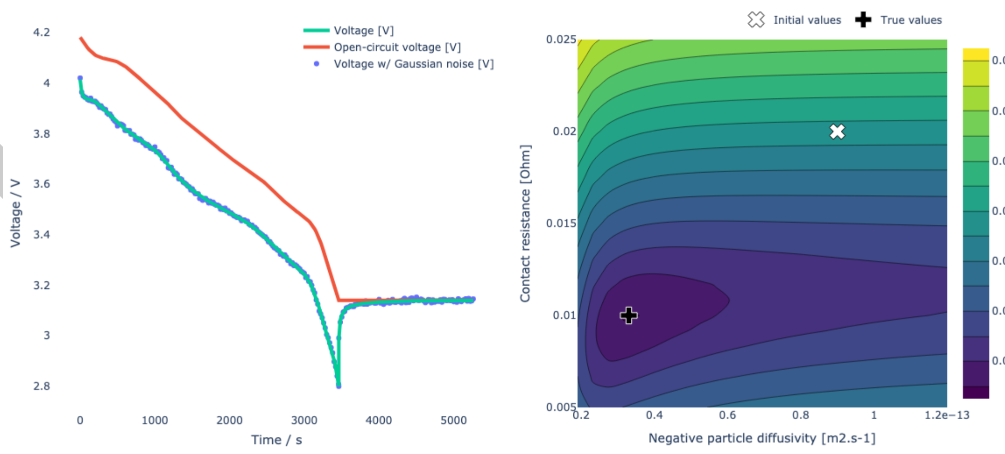

Figure 3: The cost landscape for the time-series parameterisation problem with a root mean squared error cost function.

As mentioned above, PyBOP also provides inference and optimisation capabilities through numerical computation of the electrochemical impedance. This is based on the methods presented in the pybamm-eis package and allows fast impedance computation by inversion of a sparse mass matrix after scaling and subtraction of the auto-differentiated Jacobian matrix. More information about this procedure is available in (Dhoot et al., n.d.). The Figure 4 below shows the numerical impedance prediction available in PyBOP alongside the cost landscape constructed for the inference task. At the time of publication, gradient-based optimisation and sampling methods are not available when using an impedance workflow.

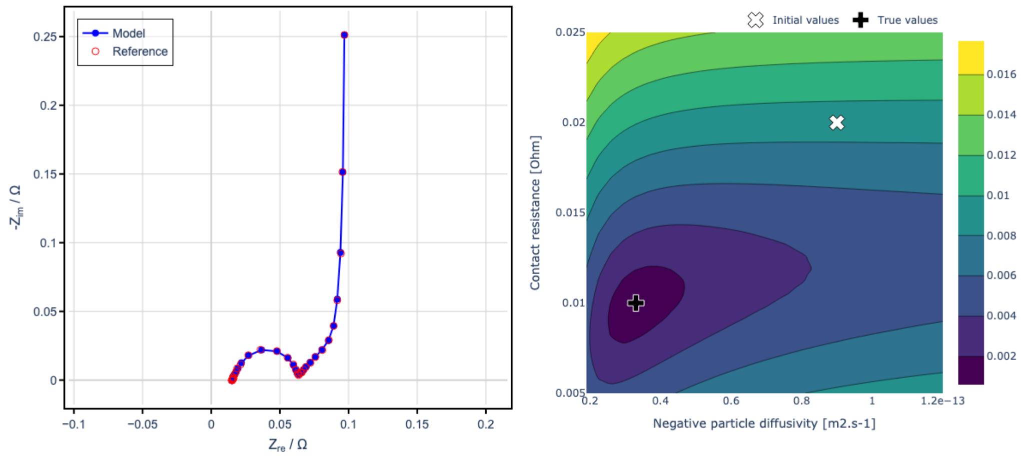

Figure 4: The cost landscape for the frequency domain impedance parameterisation problem with a root mean squared error cost function at 5% SOC.

To avoid over complicating this example, we will continue with identification in the time-domain; however, in general these two simulation methods can be combined for improved system excitation. As gradient information is available for this problem, 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, which is generally is not required, but improves convergence in this problem. As a demonstration of the parameterisation capabilities of PyBOP, Figure 5 (left) shows the rate of convergence for each of the distance-minimising cost functions, while Figure 5 (right) shows analogous results for maximising a likelihood. The optimisation is performed with SciPy Minimize using the gradient-based L-BFGS-B method.

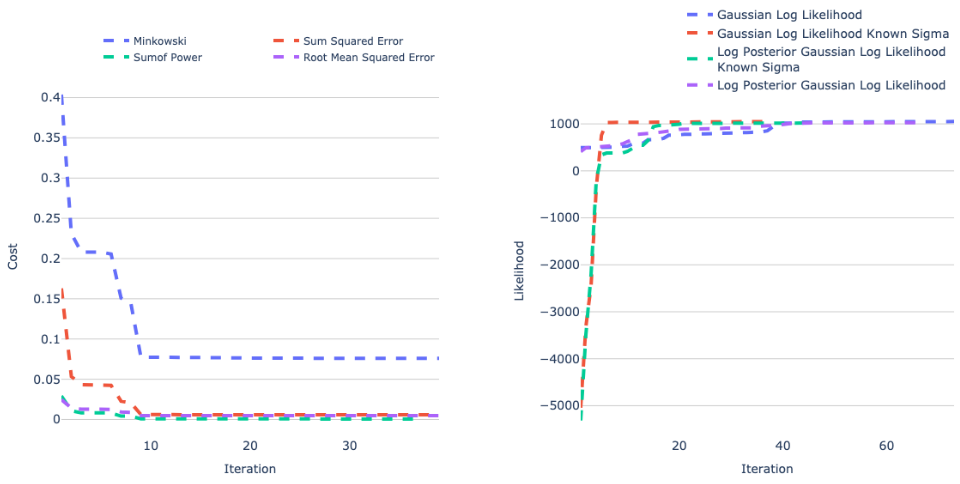

Figure 5: Convergence in the likelihood functions obtained using various likelihood functions and the L-BFGS-B algorithm.

Next, the performance of the various optimisation algorithms is presented by category: gradient-based methods in Figure 7 (left), evolutionary strategies in Figure 7 (middle) and (meta)heuristics in Figure 7 (right) for a mean squared error cost function. Note that the performance of the optimiser depends on the cost environment, prior information and corresponding hyperparameters for each specific problem.

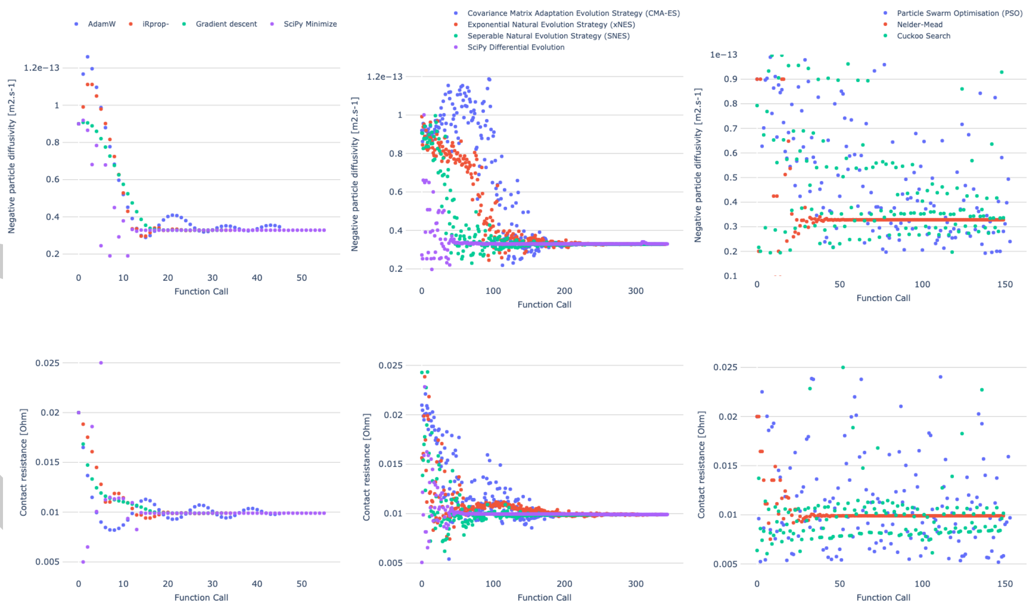

Figure 6: Convergence in the parameter values obtained for the various optimisation algorithms provided by PyBOP.

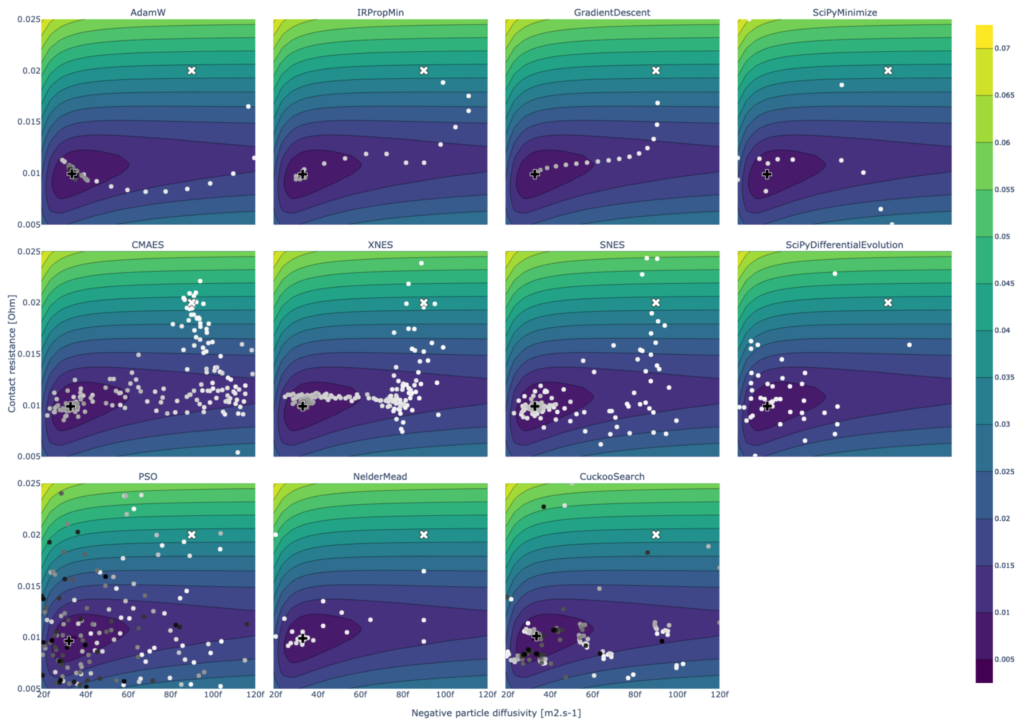

Figure 7: Cost landscape contour plot with corresponding optimisation traces. The three rows show the gradient-based optimisers (top), evolution strategies (middle), and (meta)heuristics (bottom). The order from left to right corresponds to the entries in Table 2.

This parameterisation task can also be approached from a Bayesian perspective, which we will present below using PyBOP's sampler methods. The optimisation equation presented in Equation 5 does not represent the Bayesian parameter identification task, and as such we introduce Bayes' theorem as,

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

where, $P(\theta|D)$ is the posterior, representing the probability density function of the parameter. $P(D|\theta)$ is the likelihood function, which assesses the parameter values alongside a noise model. $P(\theta)$ encapsulates the prior knowledge about the parameters, and finally $P(D)$ is the model evidence, which acts as a normalising constant. Our goal in parameter inference is to identify the parameter values with the highest probability, which can be represented either by a point-based metric or a posterior. The posterior distribution provides additional information about the uncertainty of the identified parameters. Monte Carlo methods are used 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 finally conventional methods based on random sampling with rejection criteria (Metropolis et al., 1953). PyBOP offers a sampler class that provides the interface to these samplers, which are provided by the Probabilistic Inference of Noise Time-Series (PINTS) package. Figure 8 below shows the sampled posterior for the synthetic workflow described above, using an adaptive covariance-based sampler, Haario Bardenet (Haario et al., 2001).

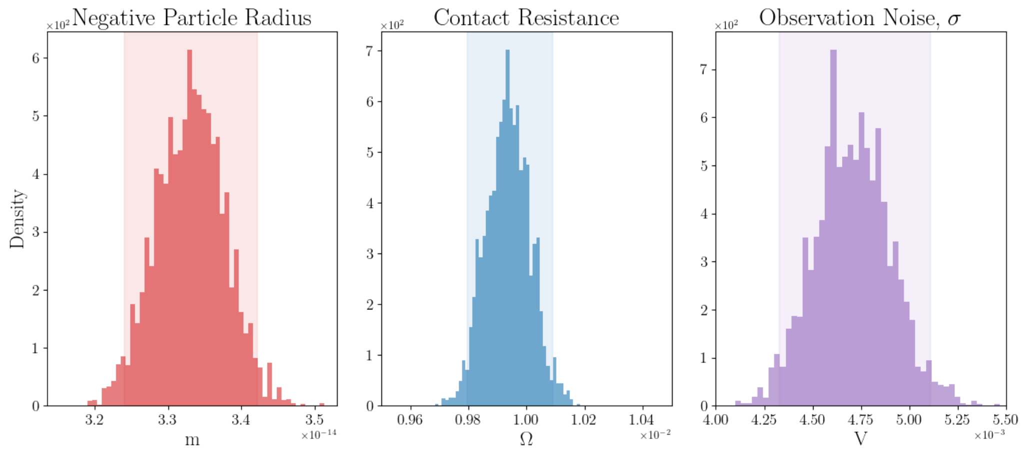

Figure 8: Posterior distributions for model parameters alongside identified noise on the observations. Shaded area denotes confidence bounds for each parameter.

Design optimisation

Design optimisation is supported within `PyBOP` to guide future battery design development by identifying parameter sensitivities that can unlock improvements in battery performance. This problem can be viewed in a similar way to the parameterisation workflows described above, but with the aim of maximising a cost function rather than minimising it. `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 in the form of a constrained optimisation problem as:

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

where $\mathcal{L} : \boldsymbol{\theta} \mapsto [0, \infty)$ 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 problem of maximising the gravimetric energy density subject to constraints on two of the geometric electrode parameters (Couto et al., 2023). For this example, we use the`PyBaMM` implementation of the single particle model with electrolyte (SPMe) to investigate the effect of the positive electrode thickness and the active material volume fraction on the target cost. Since the active material volume fraction is related to the electrode porosity, the porosity is defined with a driven constraint from the volume fraction. In this problem, we estimate the 1C rate from the theoretical capacity for each iteration of the design. For this example, we employ the Particle Swarm Optimisation (PSO) algorithm.

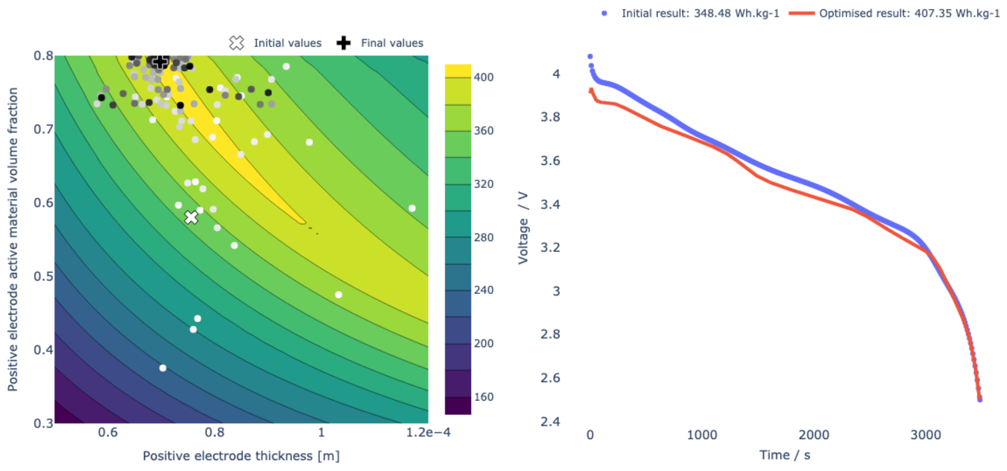

Figure 9: The gravimetric energy density landscape alongside the corresponding initial and optimised voltage profiles for a 1C discharge.

Figure 9 (left) shows the optimiser's search over the parameter space and (right) the predicted improvement in the discharge profile between the initial and optimised parameter values, simulated at their respective 1C rates.

Acknowledgements

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

References

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

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

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

Cabezas, A., Corenflos, A., Lao, J., & Louf, R. (2024). *BlackJAX: Composable Bayesian* *inference in JAX*. https://arxiv.org/abs/2402.10797

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

DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., … Viola, F. (2020). *The DeepMind JAX Ecosystem*. http://github.com/google-deepmind

Dhoot, R., Timms, R., & Please, C. (n.d.). *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

Korotkin, I., Timms, R., Foster, J. F., Dickinson, E., & Robinson, M. (2023). Battery parameter eXchange. In *GitHub repository*. The Faraday Institution. https://github.com/FaradayInstitution/BPX

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

Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*.

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