

1 PyBOP: A Python package for battery model 2 optimisation and parameterisation

3 **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¶}

5 **1** Department of Engineering Science, University of Oxford, Oxford, UK **2** The Faraday Institution,
6 Harwell Campus, Didcot, UK **3** Research Software Engineering Group, University of Oxford, Oxford, UK
7 **4** Quansight PBC **5** Mathematics Institute, University of Warwick, Coventry, UK ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

8 Summary

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

Statement of need

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

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

Architecture

37 PyBOP formulates the inference process into three core classes: Problem, Optimiser, and
38 Sampler, as shown in [Figure 1](#). Each of these objects represents a base class with child classes
39 constructing specialised functionality for different workflows. Management of these different

40 workflows is achieved through a builder pattern for the Problem class aiming to provide a
 41 robust, flexible interface. The Problem object offers the required functionality to compute the
 42 forward model, generate residuals if needed, and compute the corresponding cost for parameter
 43 value candidates. Multiple Problem builders are provided to construct optimisation Problem's
 44 for both time-series and electrochemical impedance spectroscopy PyBaMM domains alongside
 45 a pure python problem for general optimisation. The builder structure allows for extendability
 46 and flexibility when new optimisation problems are required without requiring refactoring of
 47 PyBOP's core classes. An example of this is the MultiFittingProblem and corresponding
 48 builder which generalises the pure python problem and builder for optimisation tasks where
 49 multiple objects are minimised. One such common use-case for the MultiFittingProblem is
 50 parameter identification workflows where the initial model state varies for each corresponding
 51 set of observations; however, many other use-cases are available through this generalised
 52 interface. The syntax for building a PyBaMM based parameter inference workflow is shown
 53 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()
```

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

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

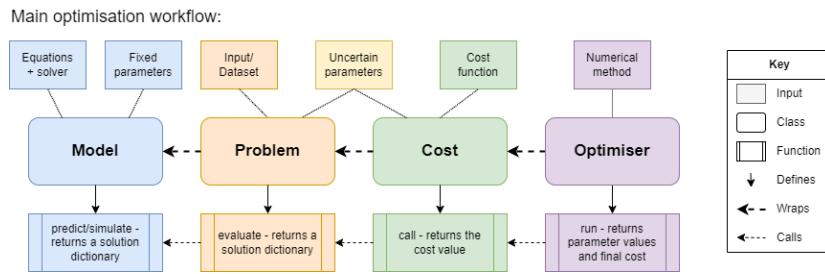


Figure 1: The core PyBOP architecture with base class interfaces. Each class provides a direct mapping to a step in the optimisation workflow.

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 computes 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 at the time of submission. 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 accompanying auto-differentiated Jacobian. This is then translated into the frequency domain, giving 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 and non-gradient 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 just 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

103 In addition to deterministic optimisers ([Table 2](#)), PyBOP also provides Monte Carlo sampling
104 routines to estimate distributions of parameters within a Bayesian framework. These methods
105 construct a posterior parameter distribution that can be used to assess uncertainty and practical
106 identifiability. The individual sampler classes are currently composed within PyBOP from the
107 PINTS library, with a base sampler class implemented for interoperability and direct integration
108 with PyBOP’s model, problem, and likelihood classes. The currently supported samplers are
109 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	Delayed rejection	Rank shrinking	Differential evolution	Metropolis random walk
gamma	adaptive	Doubling		Emcee hammer
No-U-turn	Haario	Stepout		Metropolis adjusted Langevin
Hamiltonian	Haario			
Relativistic	Rao Blackwell			

Background

Battery models

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

$$\frac{dx}{dt} = f(t, x, \theta), \quad (1)$$

$$0 = g(t, x, \theta), \quad (2)$$

$$y(t) = h(t, x, \theta), \quad (3)$$

116 with initial conditions

$$x(0) = x_0(\theta). \quad (4)$$

117 Here, t is time, $x(t)$ are the (spatially discretised) states, $y(t)$ are the outputs (e.g. the terminal
118 voltage) and θ are the unknown parameters.

119 Common battery models include various types of equivalent circuit models (e.g. the Thévenin
120 model), the Doyle–Fuller–Newman (DFN) model ([Doyle et al., 1993; Fuller et al., 1994](#))
121 based on porous electrode theory, and its reduced-order variants including the single particle
122 model (SPM) ([Brosa Planella et al., 2022](#)) and the multi-species multi-reaction (MSMR)

123 model (Verbrugge et al., 2017). Simplified models that retain acceptable predictive accuracy
 124 at lower computational cost are widely used, for example in battery management systems,
 125 while physics-based models are required to understand the impact of physical parameters on
 126 performance. This separation of complexity traditionally results in multiple parameterisations
 127 for a single battery type, depending on the model structure.

128 Examples

129 Parameterisation

130 The parameterisation of battery models is challenging due to the large number of parameters
 131 that need to be identified compared to the number of measurable outputs (Andersson et al.,
 132 2022; Miguel et al., 2021; Wang et al., 2022). A complete parameterisation often requires
 133 stepwise identification of smaller sets of parameters from a variety of excitations and different
 134 data sets (Chen et al., 2020; Chu et al., 2019; Kirk et al., 2023; Lu et al., 2021). Furthermore,
 135 parameter identifiability can be poor for a given set of excitations and data sets, requiring
 136 improved experimental design in addition to uncertainty capable identification methods (Aitio
 137 et al., 2020).

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

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

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

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

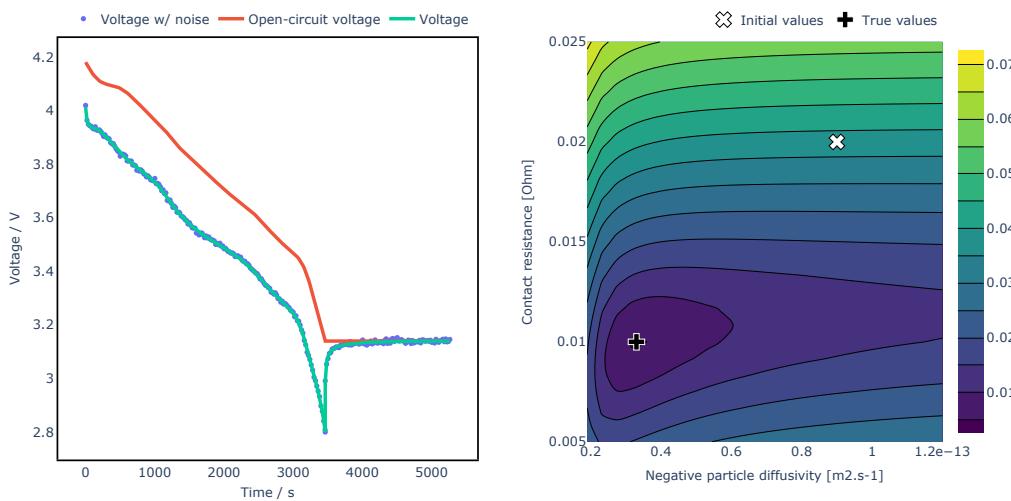


Figure 2: The synthetic fitting dataset (left) and cost landscape (right) for an example time-series battery model parameterisation using a root-mean-squared error cost function.

160 We can also use PyBOP to generate and fit electrochemical impedance data using methods
161 within pybamm-eis that enable fast impedance computation of battery models (Dhoot et al.,
162 2024). Using the same model and parameters as in the time-domain case, Figure 3 shows
163 the numerical impedance prediction available in PyBOP alongside the cost landscape for the
164 corresponding inference task. At the time of publication, gradient-based optimisation and
165 sampling methods are not available when using an impedance workflow.

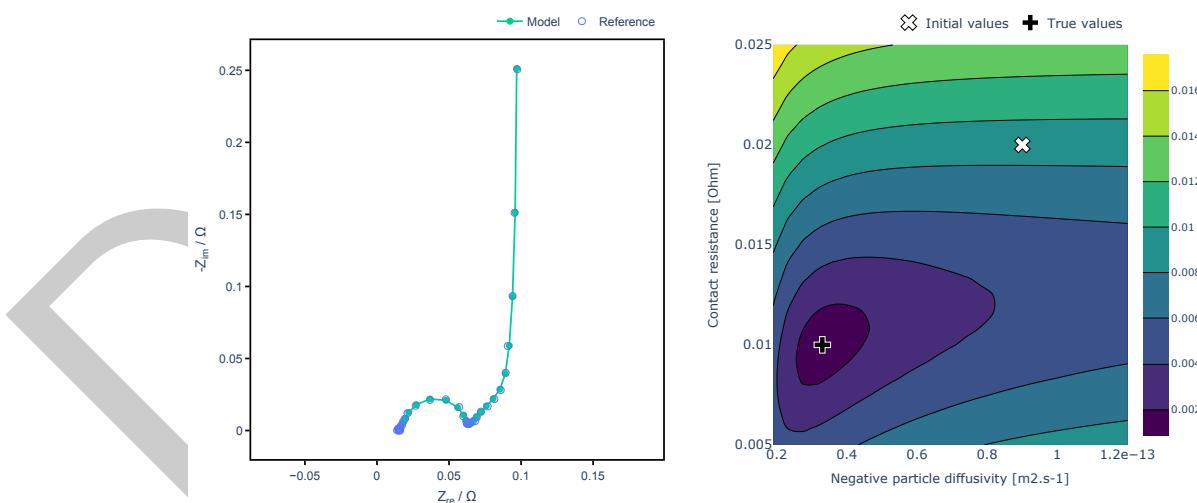


Figure 3: The data and model fit (left) and cost landscape (right) for a frequency-domain impedance parameterisation with a root-mean-squared error cost function, at 5% SOC.

166 To avoid confusion, in the remainder of this section, we continue with identification in the
167 time domain (Figure 2). In general, however, time- and frequency-domain models and data
168 may be combined for improved parameterisation. As gradient information is available for
169 our time-domain example, the choice of distance-based cost function and optimiser is not
170 constrained. Due to the difference in magnitude between the two parameters, we apply the
171 logarithmic parameter transformation offered by PyBOP. This transforms the search space of the
172 optimiser to allow for a common step size between the parameters, improving convergence in

173 this particular case. As a demonstration of the parameterisation capabilities of PyBOP, [Figure 4](#)
 174 (left) shows the rate of convergence for each of the distance-minimising cost functions, while
 175 [Figure 4](#) (right) shows analogous results for maximising a likelihood. The optimisation is
 176 performed with SciPy Minimize using the gradient-based L-BFGS-B method.

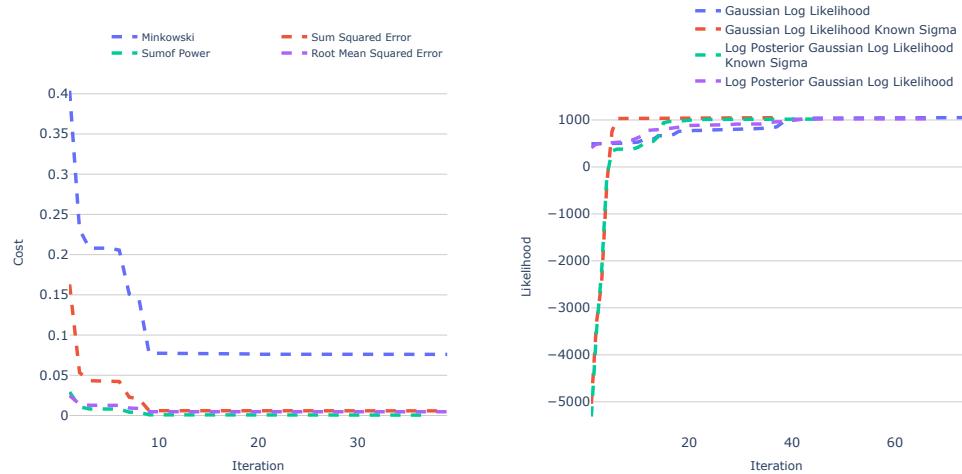


Figure 4: Optimiser convergence using various cost (left) and likelihood (right) functions and the L-BFGS-B algorithm.

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

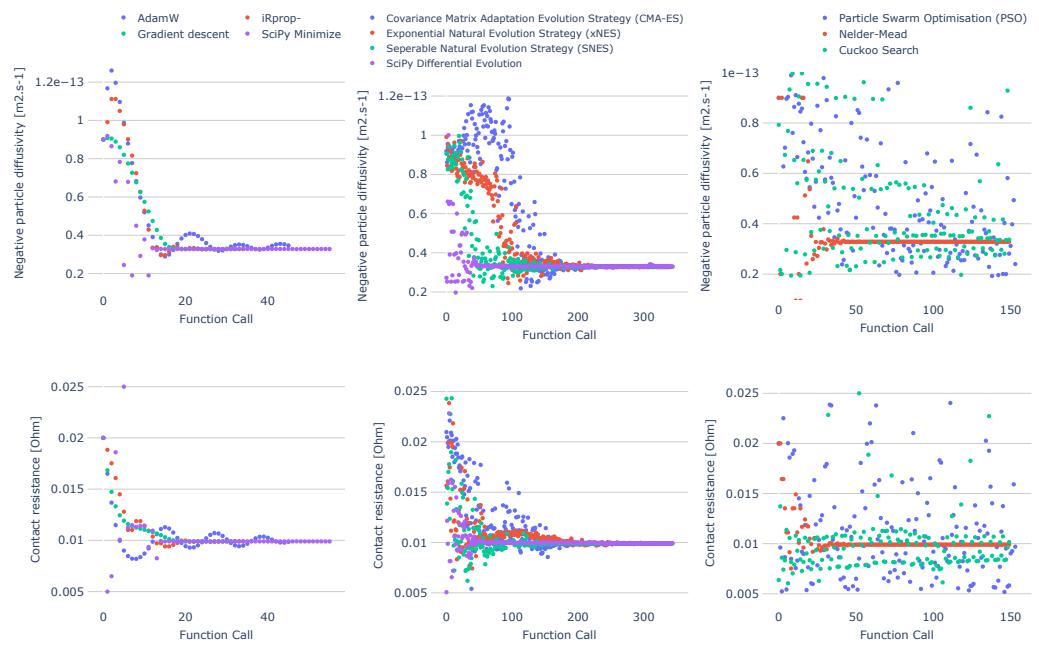


Figure 5: Convergence in parameter values for several optimisation algorithms provided by PyBOP.

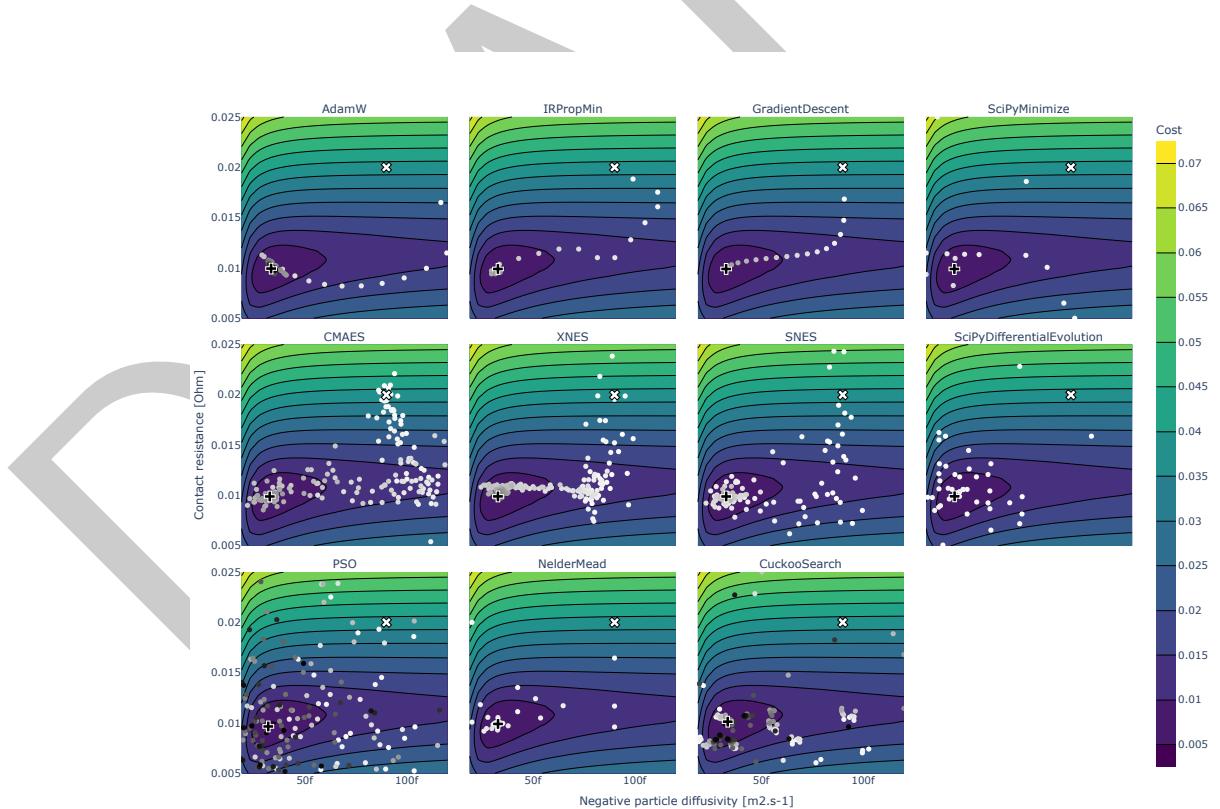


Figure 6: Cost landscape contour plot with corresponding optimisation traces, for several optimisers.

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

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

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](#)).

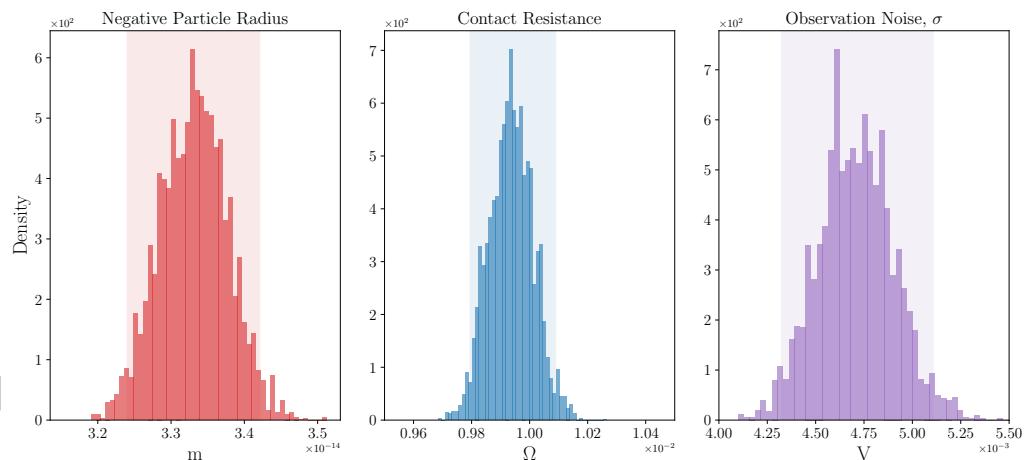


Figure 7: Posterior distributions of model parameters alongside identified noise on the observations. Shaded areas denote the 95th percentile credible interval for each parameter.

203 Design optimisation

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

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

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

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

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

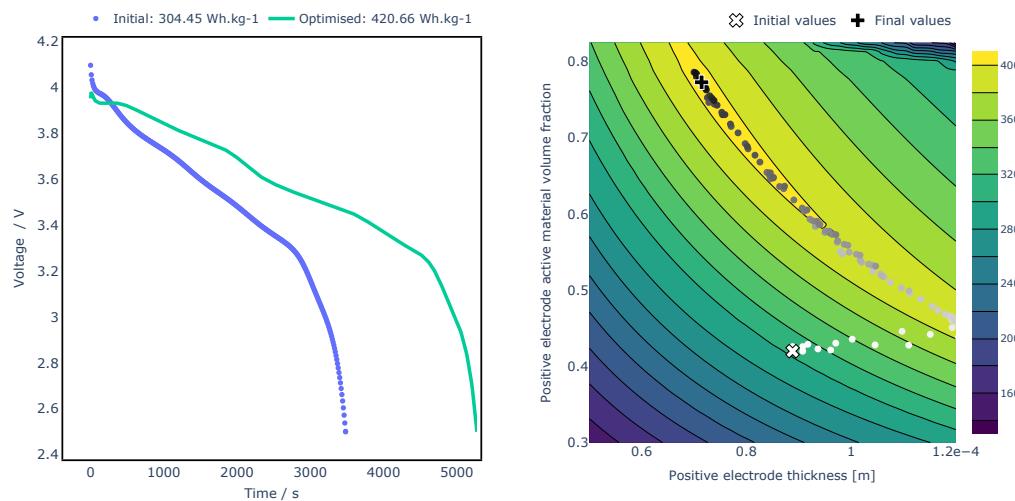


Figure 8: Initial and optimised voltage profiles alongside the gravimetric energy density cost landscape.

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

228 Acknowledgements

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

232 References

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

- 240 Andersson, M., Streb, M., Ko, J. Y., Löfqvist Klass, V., Klett, M., Ekström, H., Johansson,
 241 M., & Lindbergh, G. (2022). Parametrization of physics-based battery models from input-
 242 output data: A review of methodology and current research. *Journal of Power Sources*,
 243 521(November 2021), 230859. <https://doi.org/10.1016/j.jpowsour.2021.230859>
- 244 Braak, C. J. F. T. (2006). A Markov Chain Monte Carlo version of the genetic algorithm
 245 Differential Evolution: Easy Bayesian computing for real parameter spaces. *Statistics and*
 246 *Computing*, 16(3), 239–249. <https://doi.org/10.1007/s11222-006-8769-1>
- 247 Brooks, S., Gelman, A., Jones, G., & Meng, X.-L. (2011). *Handbook of markov chain monte*
 248 *carlo*. Chapman; Hall/CRC. <https://doi.org/10.1201/b10905>
- 249 Brosa Planella, F., Ai, W., Boyce, A. M., Ghosh, A., Korotkin, I., Sahu, S., Sulzer, V., Timms,
 250 R., Tranter, T. G., Zyskin, M., Cooper, S. J., Edge, J. S., Foster, J. M., Marinescu, M., Wu,
 251 B., & Richardson, G. (2022). A Continuum of Physics-Based Lithium-Ion Battery Models
 252 Reviewed. *Progress in Energy*, 4(4), 042003. <https://doi.org/10.1088/2516-1083/ac7d31>
- 253 Cauchy, A., & others. (1847). Méthode générale pour la résolution des systemes d'équations
 254 simultanées. *Comp. Rend. Sci. Paris*, 25(1847), 536–538.
- 255 Chen, C.-H., Brosa Planella, F., O'Regan, K., Gastol, D., Widanage, W. D., & Kendrick,
 256 E. (2020). Development of experimental techniques for parameterization of multi-scale
 257 lithium-ion battery models. *Journal of The Electrochemical Society*, 167(8), 080534.
 258 <https://doi.org/10.1149/1945-7111/ab9050>
- 259 Chu, Z., Plett, G. L., Trimboli, M. S., & Ouyang, M. (2019). A control-oriented electrochemical
 260 model for lithium-ion battery, Part I: Lumped-parameter reduced-order model with constant
 261 phase element. *Journal of Energy Storage*, 25(August), 100828. <https://doi.org/10.1016/j.est.2019.100828>
- 263 Clerx, M., Robinson, M., Lambert, B., Lei, C. L., Ghosh, S., Mirams, G. R., & Gavaghan, D.
 264 J. (2019). Probabilistic inference on noisy time series (PINTS). *Journal of Open Research*
 265 *Software*, 7(1), 23. <https://doi.org/10.5334/jors.252>
- 266 Couto, L. D., Charkhgard, M., Karaman, B., Job, N., & Kinnaert, M. (2023). Lithium-ion
 267 battery design optimization based on a dimensionless reduced-order electrochemical model.
 268 *Energy*, 263(PE), 125966. <https://doi.org/10.1016/j.energy.2022.125966>
- 269 Dhoot, R., Timms, R., & Please, C. (2024). *PyBaMM EIS: Efficient linear algebra meth-*
 270 *ods to determine li-ion battery behaviour* (Version 0.1.4). <https://www.github.com/pybamm-team/pybamm-eis>
- 272 Doyle, M., Fuller, T. F., & Newman, J. (1993). Modeling of Galvanostatic Charge and
 273 Discharge of the Lithium/Polymer/Insertion Cell. *Journal of The Electrochemical Society*,
 274 140(6), 1526–1533. <https://doi.org/10.1149/1.2221597>
- 275 Fuller, T. F., Doyle, M., & Newman, J. (1994). Simulation and optimization of the dual
 276 lithium ion insertion cell. *Journal of The Electrochemical Society*, 141(1), 1. <https://doi.org/10.1149/1.2054684>
- 278 Haario, H., Saksman, E., & Tamminen, J. (2001). An Adaptive Metropolis Algorithm. *Bernoulli*,
 279 7(2), 223. <https://doi.org/10.2307/3318737>
- 280 Hoffman, M. D., & Gelman, A. (2011). *The no-u-turn sampler: Adaptively setting path*
 281 *lengths in hamiltonian monte carlo*. <https://arxiv.org/abs/1111.4246>
- 282 Kirk, T. L., Lewis-Douglas, A., Howey, D., Please, C. P., & Jon Chapman, S. (2023).
 283 Nonlinear electrochemical impedance spectroscopy for lithium-ion battery model parame-
 284 terization. *Journal of The Electrochemical Society*, 170(1), 010514. <https://doi.org/10.1149/1945-7111/acada7>
- 286 Loshchilov, I., & Hutter, F. (2017). *Decoupled Weight Decay Regularization*. arXiv. <https://arxiv.org/abs/1612.04786>

- 287 [//doi.org/10.48550/ARXIV.1711.05101](https://doi.org/10.48550/ARXIV.1711.05101)
- 288 Lu, D., Scott Trimboli, M., Fan, G., Zhang, R., & Plett, G. L. (2021). Implementation of a
289 physics-based model for half-cell open-circuit potential and full-cell open-circuit voltage
290 estimates: Part II. Processing full-cell data. *Journal of The Electrochemical Society*, 168(7),
291 070533. <https://doi.org/10.1149/1945-7111/ac11a5>
- 292 Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953).
293 Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical
294 Physics*, 21(6), 1087–1092. <https://doi.org/10.1063/1.1699114>
- 295 Miguel, E., Plett, G. L., Trimboli, M. S., Oca, L., Iraola, U., & Bekaert, E. (2021). Review
296 of computational parameter estimation methods for electrochemical models. *Journal of
297 Energy Storage*, 44(PB), 103388. <https://doi.org/10.1016/j.est.2021.103388>
- 298 Sulzer, V., Marquis, S. G., Timms, R., Robinson, M., & Chapman, S. J. (2021). Python
299 Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, 9(1),
300 14. <https://doi.org/10.5334/jors.309>
- 301 Tranter, T. G., Timms, R., Sulzer, V., Planella, F. B., Wiggins, G. M., Karra, S. V., Agarwal,
302 P., Chopra, S., Allu, S., Shearing, P. R., & Brett, D. J. I. (2022). Liionpack: A python
303 package for simulating packs of batteries with PyBaMM. *Journal of Open Source Software*,
304 7(70), 4051. <https://doi.org/10.21105/joss.04051>
- 305 Verbrugge, M., Baker, D., Koch, B., Xiao, X., & Gu, W. (2017). Thermodynamic model for
306 substitutional materials: Application to lithiated graphite, spinel manganese oxide, iron
307 phosphate, and layered nickel-manganese-cobalt oxide. *Journal of The Electrochemical
308 Society*, 164(11), E3243. <https://doi.org/10.1149/2.0341708jes>
- 309 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D.,
310 Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson,
311 J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy
312 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in
313 Python. *Nature Methods*, 17, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- 314 Wang, A. A., O'Kane, S. E. J., Brosa Planella, F., Houx, J. L., O'Regan, K., Zyskin, M., Edge,
315 J., Monroe, C. W., Cooper, S. J., Howey, D. A., Kendrick, E., & Foster, J. M. (2022).
316 Review of parameterisation and a novel database (LiionDB) for continuum Li-ion battery
317 models. *Progress in Energy*, 4(3), 032004. <https://doi.org/10.1088/2516-1083/ac692c>
- 318 Yang, X.-S., & Suash Deb. (2009). Cuckoo Search via levy flights. *2009 World Congress on
319 Nature & Biologically Inspired Computing (NaBIC)*, 210–214. <https://doi.org/10.1109/NABIC.2009.5393690>