

# 4th Online Developer Meeting



The PyBOP Team 

19th September 2024

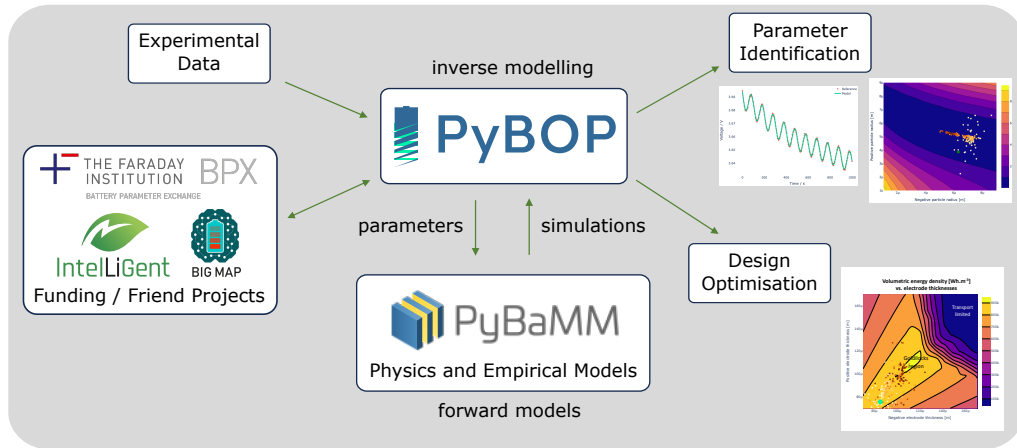
# Agenda

---



- ▶ Presentation on recent updates to PyBOP (10 min)
- ▶ Quick update from everyone (10 min)
- ▶ Discussion of open issues (20 min)
- ▶ Planning and assignment of tasks (15 min)

Introductory slides are available at the end of this presentation.



# Releases

---



v24.9.1 was released on September 18th.

<https://github.com/pybop-team/PyBOP/blob/develop/CHANGELOG.md>

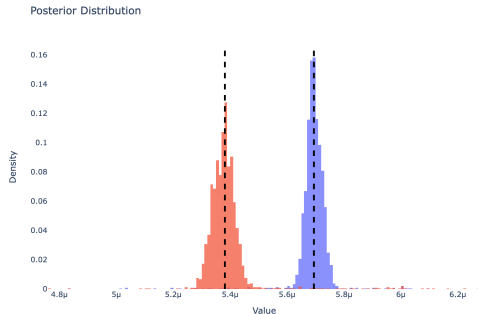
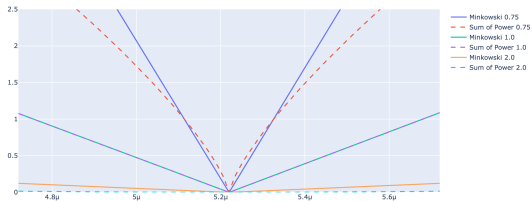
The Changelog is divided into new **features**, **bug fixes** and **breaking changes**.

## New features that landed in v24.9



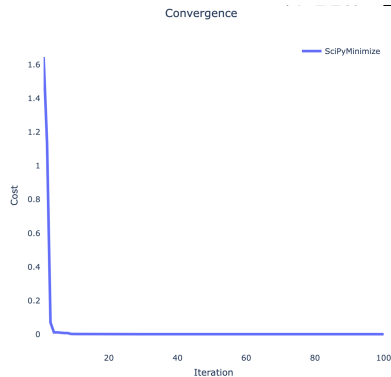
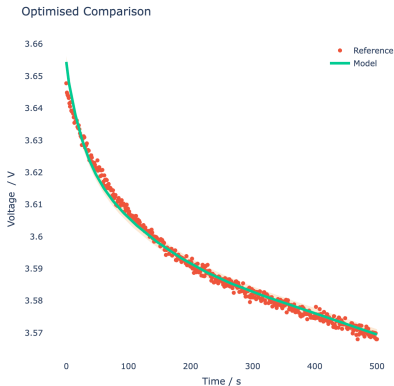
#462 - Adds Minkowski and SumofPower cost functions.

#6 - Adds Monte Carlo Sampling classes



## New features that landed in v24.9

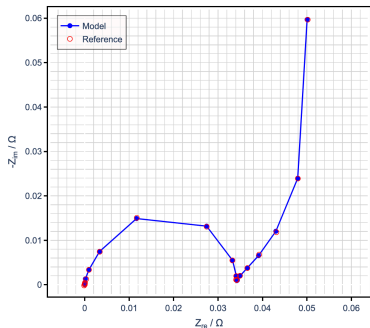
### #353 - Allows user defined nonlinear constraints



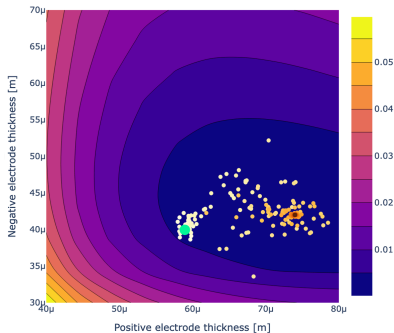
## New features that landed in v24.9

### #405 - Adds EIS prediction methods

Optimised Comparison

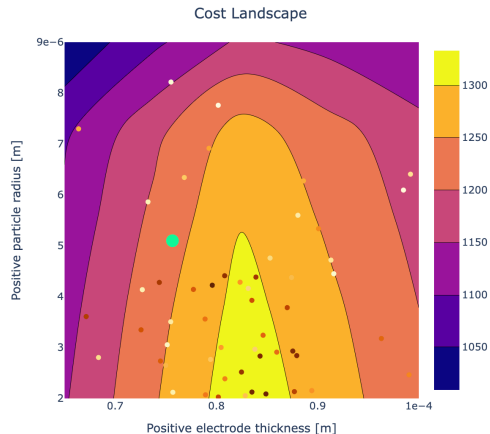


Cost Landscape



## Combine costs with weighting:

```
# Generate multiple cost functions and combine them
cost1 = pybop.GravimetricEnergyDensity(problem)
cost2 = pybop.VolumetricEnergyDensity(problem)
cost = pybop.WeightedCost(cost1, cost2, weights=[1, 1e-3])
```





## New features that landed in v24.9

#357 - Adds Transformations classes

#222 - Adds an example for performing electrode balancing

#441 - Adds an example for estimating constants within a `pybamm.FunctionalParameter`

#450 - Adds support for IDAKLU with output variables

#444 - Merge BaseModel `build()` and `rebuild()` functionality

# Discussion of open issues

---



Kick-off questions:

- ▶ What issue are you working on/would you like to work on?
- ▶ Do you have any blockers?

Open issues: <https://github.com/pybop-team/PyBOP/issues>  
Project board: <https://github.com/orgs/pybop-team/projects/8>

## Triage new issues

---

Other open items?

---

## End of meeting

---

Below:



1. Introductory slides
2. Feature highlights

# Introduction

---



With thanks to:

- ▶ the PyBaMM developers  PyBaMM
- ▶ the PINTS developers 



## 1. Parameter estimation from cell data

- How will my battery respond to different excitations?
- How is degradation affecting my battery performance?

## 2. Model comparison

- What can I learn from a more detailed model?
- Is it worth the extra computational effort?



## 3. Design optimisation within practical constraints

- How energy dense can I make my battery (at some rate)?

Build a model

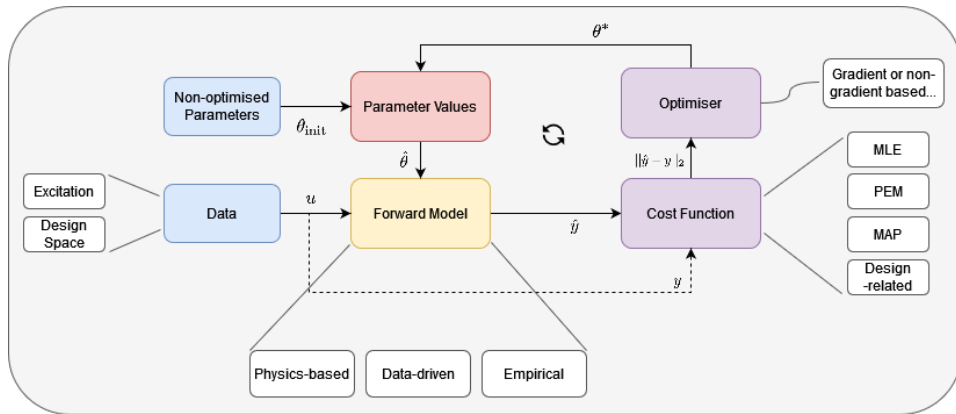


Perform an optimisation



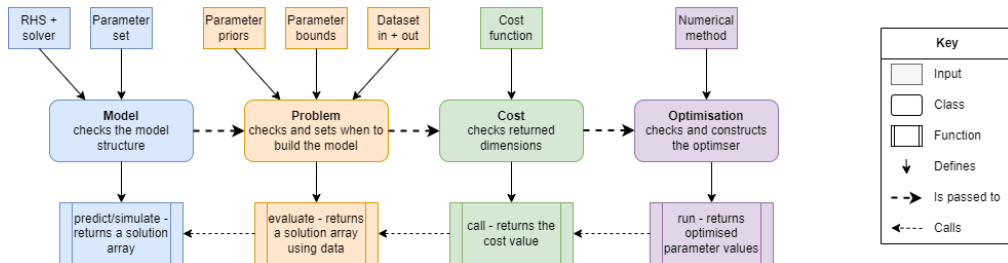
Learn from the result

Icons made by Uniconlabs, RaftelDesign and Freepik from [www.flaticon.com](http://www.flaticon.com)





Main optimisation workflow:



## Demo/examples

---



- ▶ Parameterisation
- ▶ Design optimisation

## 1. Define a model (PyBaMM's SPM) and fitting parameters

```
import pybop
import numpy as np

# Define model
parameter_set = pybop.ParameterSet.pybamm("Chen2020")
model = pybop.lithium_ion.SPM(parameter_set=parameter_set)

# Fitting parameters
parameters = pybop.Parameters(
    pybop.Parameter(
        "Negative particle radius [m]",
        prior=pybop.Gaussian(6e-06, 0.1e-6),
        bounds=[1e-6, 9e-6],
    ),
    pybop.Parameter(
        "Positive particle radius [m]",
        prior=pybop.Gaussian(4.5e-06, 0.1e-6),
        bounds=[1e-6, 9e-6],
    ),
)
```

## 2. Generate synthetic data and define problem, cost and optimisation

```
# Generate data
sigma = 0.001
t_eval = np.arange(0, 900, 3)
values = model.predict(t_eval=t_eval)
corrupt_values = values["Voltage [V]"].data + np.random.normal(0, sigma, len(t_eval))

# Form dataset
dataset = pybop.Dataset(
    {
        "Time [s]": t_eval,
        "Current function [A]": values["Current [A]"].data,
        "Voltage [V]": corrupt_values,
    }
)

# Generate problem, cost function, and optimisation class
problem = pybop.FittingProblem(model, parameters, dataset)
cost = pybop.SumSquaredError(problem)
optim = pybop.CMAES(cost, max_iterations=100)
```

## 3. Run the optimisation and plot the results

```
# Run the optimisation
x, final_cost = optim.run()
print("True parameters:", parameters.true_value())
print("Estimated parameters:", x)

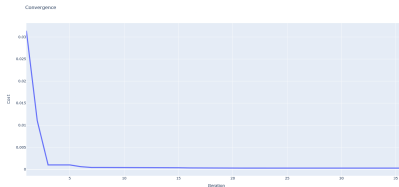
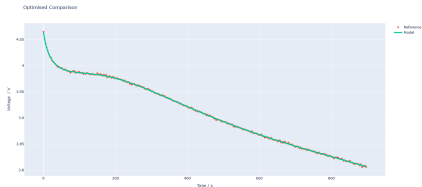
# Plot the timeseries output
pybop.quick_plot(problem, problem_inputs=x, title="Optimised Comparison")

# Plot convergence and parameter traces
pybop.plot_convergence(optim)

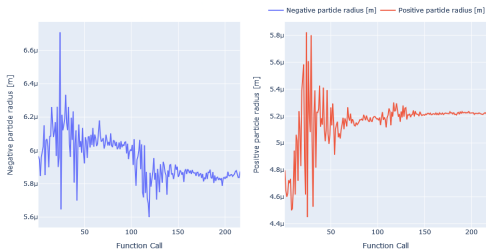
# Plot the parameter traces
pybop.plot_parameters(optim)

# Plot the cost landscape with optimisation path
pybop.plot2d(optim, steps=15)
```

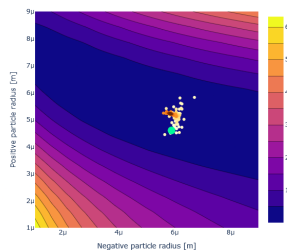
# Parameterisation results



Parameter Convergence



Cost Landscape



## 1. Define a model (PyBaMM's SPM) and design parameters

```
import pybop
import numpy as np

# Define parameter set and model
parameter_set = pybop.ParameterSet.pybamm("Chen2020", formation_concentrations=True)
model = pybop.lithium_ion.SPM(parameter_set=parameter_set)

# Fitting parameters
parameters = pybop.Parameters(
    pybop.Parameter(
        "Positive electrode thickness [m]",
        prior=pybop.Gaussian(7.56e-05, 0.05e-05),
        bounds=[65e-06, 10e-05],
    ),
    pybop.Parameter(
        "Positive particle radius [m]",
        prior=pybop.Gaussian(5.22e-06, 0.05e-06),
        bounds=[2e-06, 9e-06],
    ),
)
```

### 2. Set the target experiment and define problem, cost and optimisation

```
# Define test protocol
experiment = pybop.Experiment(
    ["Discharge at 1C until 2.5 V (5 seconds period)",
    ]
    init_soc = 1 # start from full charge
    signal = ["Voltage [V]", "Current [A]"]

# Generate problem
problem = pybop.DesignProblem(
    model, parameters, experiment, signal=signal, init_soc=init_soc
)

# Generate cost function and optimisation class:
cost = pybop.GravimetricEnergyDensity(problem, update_capacity=True)
optim = pybop.PSO(
    cost, verbose=True, allow_infeasible_solutions=False, max_iterations=15
)
```



### 3. Run the optimisation and plot the results

```
# Run optimisation
x, final_cost = optim.run()
print("Estimated parameters:", x)
print(f"Initial gravimetric energy density: {cost(optim.x0):.2f} Wh.kg-1")
print(f"Optimised gravimetric energy density: {final_cost:.2f} Wh.kg-1")

# Plot the timeseries output
if cost.update_capacity:
    problem._model.approximate_capacity(x)
pybop.quick_plot(problem, problem_inputs=x, title="Optimised Comparison")

# Plot the cost landscape with optimisation path
pybop.plot2d(optim, steps=10)
```

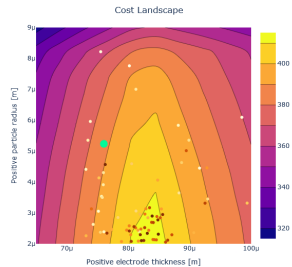
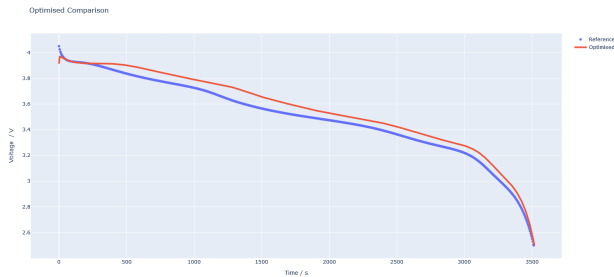
## Output

Estimated parameters:  $[6.51159429 \times 10^{-5} \ 2.32340723 \times 10^{-6}]$

Initial gravimetric energy density: 386.31 Wh.kg<sup>-1</sup>

Optimised gravimetric energy density: 410.78 Wh.kg<sup>-1</sup>

## Results

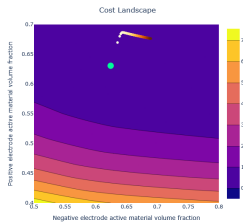


# Choice of optimisers

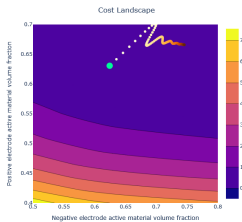
---



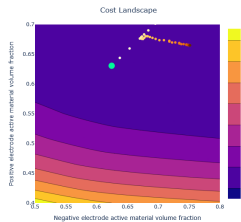
## Gradient descent



## Adaptive moment (AdamW)

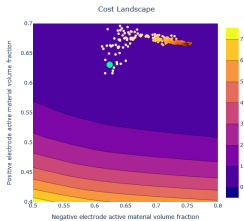


## Resilient backpropagation (IRPropMin)

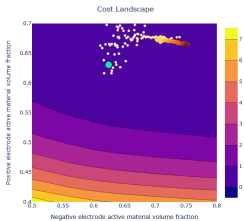


# Evolution strategies

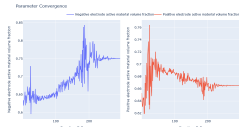
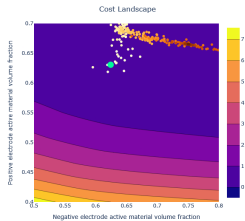
## Stochastic natural evolution strategy (SNES)



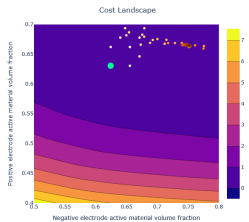
## Exponential natural evolution strategy (SNES)



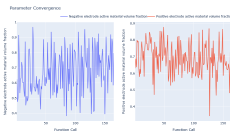
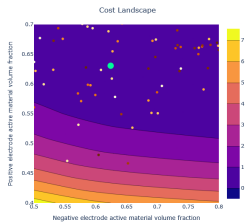
## Covariance matrix adaptation (CMA-ES)



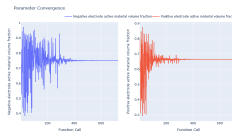
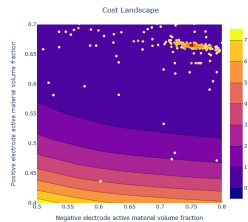
## Nelder Mead



## Particle swarm optimisation



## SciPy differential evolution



## Code of Conduct

PyBOP aims to foster a broad consortium of developers and users, both across and outside the Faraday Institution community.

Our values are:



Open-source (code and ideas should be shared)



Inclusivity and fairness (those who want to contribute may and input is appropriately recognised)



Inter-operability (aiming for modularity to enable maximum impact and inclusivity)



User-friendliness (putting user requirements first, thinking about user-assistance & workflows)

Icons made by Good Ware, Dewi Sari, Nhor Phai and Vectors Tank from [www.flaticon.com](http://www.flaticon.com)



<https://github.com/pybop-team/PyBOP>



SCAN ME

[pybop-docs.readthedocs.io](https://pybop-docs.readthedocs.io)

## Feature highlights

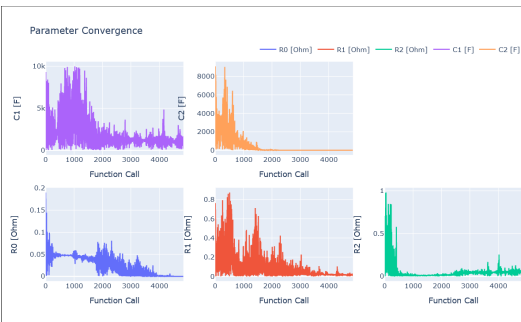
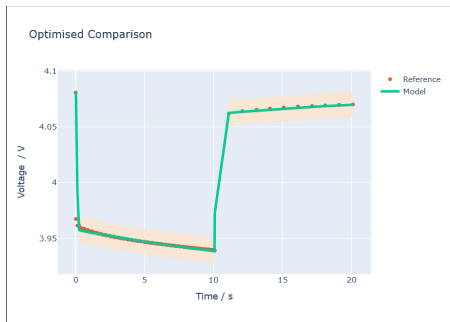
---





## New in v24.6: Experimental data fitting

#241 - Adds experimental circuit model fitting notebook with LG M50 data from:  
[https://github.com/WDWidanage/Simscape-Battery-Library/tree/main/Examples/parameterEstimation\\_TECMD/Data](https://github.com/WDWidanage/Simscape-Battery-Library/tree/main/Examples/parameterEstimation_TECMD/Data)



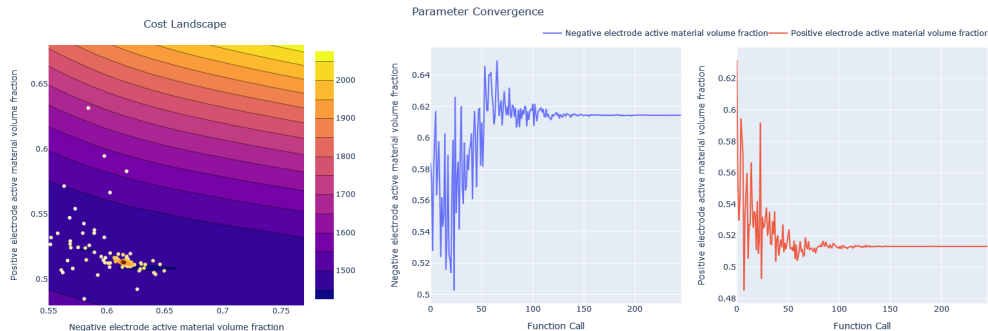
## New in v24.6: Additional PyBaMM models

#250 - Adds DFN, MPM, MSMR models and moves multiple construction variables to BaseEChem. Adds exception catch on simulate & simulateS1.

Battery Models	Optimization Algorithms	Cost Functions
Single Particle Model (SPM)	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	Sum of Squared Errors (SSE)
Single Particle Model with Electrolyte (SPMe)	Particle Swarm Optimization (PSO)	Root Mean Squared Error (RMSE)
Doyle-Fuller-Newman (DFN)	Adaptive Moment Estimation (Adam)	Maximum Likelihood Estimation (MLE)
Many Particle Model (MPM)	Improved Resilient Backpropagation (iRProp-)	Maximum a Posteriori (MAP)
Multi-Species Multi-Reactants (MSMR)	Exponential Natural Evolution Strategy (xNES)	Unscented Kalman Filter (UKF)
Equivalent Circuit Models (ECM)	Separable Natural Evolution Strategy (sNES)	Gravimetric Energy Density
	Gradient Descent	Volumetric Energy Density
	Nelder-Mead	
	SciPy Minimize & Differential Evolution	

## New in v24.6: Maximum a Posteriori

#275 - Adds Maximum a Posteriori (MAP) cost function with corresponding tests and example script.



## New in v24.6: Testing and infrastructure



Credit and thanks to Agriya and Brady!

### Features

- #304 - Decreases the testing suite completion time.
- #301 - Updates default echem solver to "fast with events" mode.
- #251 - Increment PyBaMM > v23.5, remove redundant tests within integration tests, increment citation version, fix ex
- #285 - Drop support for Python 3.8.
- #268 - Fixes the GitHub Release artifact uploads, allowing verification of codesigned binaries and source distributi
- #79 - Adds BPX as a dependency and imports BPX support from PyBaMM.
- #267 - Add classifiers to pyproject.toml, update project.urls.
- #195 - Adds the Nelder-Mead optimiser from PINTS as another option.

### Bug Fixes

- #317 - Installs seed packages into nox sessions, ensuring that scheduled tests can pass.
- #308 - Enables testing on both macOS Intel and macOS ARM (Silicon) runners and fixes the scheduled tests.
- #299 - Bugfix multiprocessing support for Linux, MacOS, Windows (WSL) and improves coverage.
- #270 - Updates PR template.
- #91 - Adds a check on the number of parameters for CMAES and makes XNES the default optimiser.