# PyBOP: #1

BRADY PLANDEN

*Engineering Science*
*University of Oxford*

Long Presentation, November 2023

Battery Intelligence Lab

Predictive continuum models require parameter sets, which are challenging to create. These sets are created by:

- Experimentalists, who might not have expertise in the model structure;

Predictive continuum models require parameter sets, which are challenging to create. These sets are created by:

- Experimentalists, who might not have expertise in the model structure;
- Modellers, who fit the parameter set from varying quality of data

# Predictive modelling

and using other peoples data

Predictive continuum models require parameter sets, which are challenging to create. These sets are created by:

- Experimentalists, who might not have expertise in the model structure;
- Modellers, who fit the parameter set from varying quality of data
- A mixture of the two, (i.e. the ideal situation)

# Ad-Hoc fitting results in varying model performance

Three different popular parameter sets, each with variations in construction:

- Chen2020 $\rightarrow$ teardown + imaging, half & full cell cycling, refit parameters

# Ad-Hoc fitting results in varying model performance

Three different popular parameter sets, each with variations in construction:

- Chen2020 $\rightarrow$ teardown + imaging, half & full cell cycling, refit parameters
- Ecker2015 $\rightarrow$ teardown + spectrometry, half & full cell cycling, refit parameters

# Ad-Hoc fitting results in varying model performance

Three different popular parameter sets, each with variations in construction:

- Chen2020 $\rightarrow$ teardown + imaging, half & full cell cycling, refit parameters
- Ecker2015 $\rightarrow$ teardown + spectrometry, half & full cell cycling, refit parameters
- Mohat2020 $\rightarrow$ Built in-house, half & full cell cycling, pressure model w/ fitting and measurements coupled.

# Ad-Hoc fitting results in varying model performance

Three different popular parameter sets, each with variations in construction:

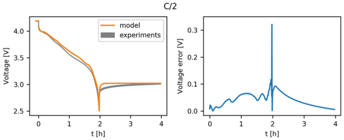
- Chen2020 $\rightarrow$ teardown + imaging, half & full cell cycling, refit parameters
- Ecker2015 $\rightarrow$ teardown + spectrometry, half & full cell cycling, refit parameters
- Mohat2020 $\rightarrow$ Built in-house, half & full cell cycling, pressure model w/ fitting and measurements coupled.

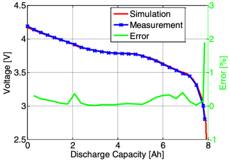
*Uniqueness*?

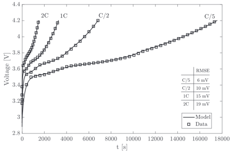
# Ad-Hoc fitting results in varying model performance

Three different popular parameter sets, each with a different creation method:

- Chen2020 $\rightarrow$ DFN w/ trial and error fitting (RMSE), parameter variation between 12% to 1800%
- Ecker2015 $\rightarrow$ DFN, variation from 45% to 154%
- Mohat2020 $\rightarrow$ MPMe, variation from 0% to ?%



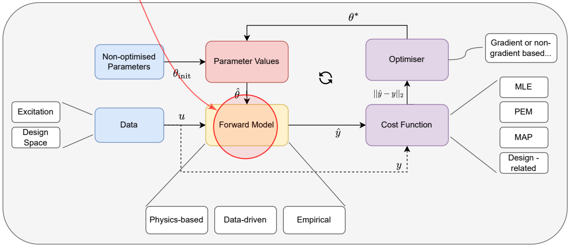
A package to standardised parameter identification and optimisation that provides:

- An API that offers complexity to advanced users while guiding new users
- A research platform to improve parameter identification and battery optimisation
- Performance, both in iter/s and convergance *guarantees*
- Open-source!

# Function calls that align with intuition

The high-level API is,

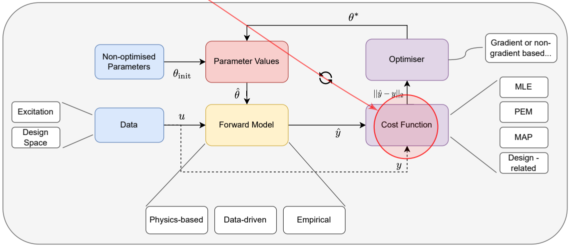
```
# Generate problem, cost function, and optimisation class
problem = pybop.Problem(model, parameters, dataset)
cost = pybop.SumSquaredError(problem)
optim = pybop.Optimisation(cost, optimiser=pybop.GradientDescent)
x, final_cost = optim.run()
```



# Function calls that align with intuition

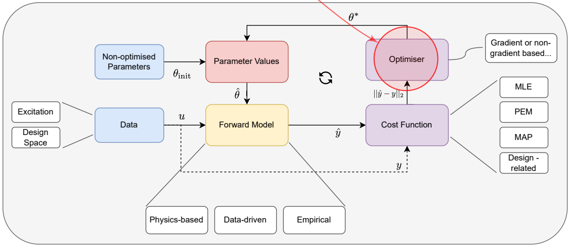
The high-level API is,

```
# Generate problem, cost function, and optimisation class
problem = pybop.Problem(model, parameters, dataset)
cost = pybop.SumSquaredError(problem)
optim = pybop.Optimisation(cost, optimiser=pybop.GradientDescent)
x, final_cost = optim.run()
```



The high-level API is,

```python
# Generate problem, cost function, and optimisation class
problem = pybop.Problem(model, parameters, dataset)
cost = pybop.SumSquaredError(problem)
optim = pybop.Optimisation(cost, optimiser=pybop.GradientDescent)
x, final_cost = optim.run()
```



CMA-ES[1] provides a robust solution to challenging (noisy, multi-modal, etc.) cost landscapes, at the exchange for performance without requiring hyperparameter tuning. The algorithm is defined by the following steps,

1. Sample candidate solutions:

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)}\mathcal{N}\left(0, \mathbf{C}^{(g)}\right), \qquad \text{for } g = 1, ..., \lambda$$

[1] I have a blog post that expands on this: bradyplanden.github.io

2. Construct the Evolutionary Paths:

$$\mathbf{p}_{\mathrm{c}}^{(g+1)} = (1 - c_{\mathrm{c}})\mathbf{p}_{\mathrm{c}}^{(g)} + h_{\sigma}^{(g+1)}\sqrt{c_{\mathrm{c}}(2 - c_{\mathrm{c}})\mu_{w}}dy$$

$$\mathbf{p}_{\sigma}^{(g+1)} = (1 - c_{\sigma})\mathbf{p}_{\sigma}^{(g)} + \sqrt{c_{\sigma}(2 - c_{\sigma})\mu_{w}}dz$$

$$h_{\sigma}^{(g+1)} = \begin{cases} 1, & \text{if } \frac{\left\|\mathbf{p}_{\sigma}^{(g+1)}\right\|^2}{1-(1-c_{\sigma})^{2\cdot(g+1)}} < (2 + 4/(d+1))\,d, \\ 0, & \text{otherwise} \end{cases}$$

# Cont.

3. Update the Search Distribution:

$$\mathbf{m}^{(g+1)} = \mathbf{m}^{(g)} + c_{\mathrm{m}} \sum_{i=1}^{\mu} w_i \left( \mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)} \right)$$

$$\sigma^{(g+1)} = \sigma^{(g)} \exp \left( \frac{c_\sigma}{d_\sigma} \left( \frac{\left\| \mathbf{p}_\sigma^{(g+1)} \right\|}{\mathbb{E}\|\mathcal{N}(0,\mathbf{I})\|} - 1 \right) \right)$$

$$\mathbf{C}^{(g+1)} = \left(1 - c_1 - c_\mu \sum w_j\right) \mathbf{C}^{(g)} + c_1 \underbrace{\mathbf{p}_{\mathrm{c}}^{(g+1)} \mathbf{p}_{\mathrm{c}}^{(g+1)^T}}_{\text{rank-one update}} \underbrace{+ c_\mu \sum_{i=1}^{\lambda} w_i \mathbf{y}_{i:\lambda}^{(g+1)} \left( \mathbf{y}_{i:\lambda}^{(g+1)} \right)^T}_{\text{rank-}\mu\text{ update}}$$

# Example: Fitting SPM Parameters from Pulse Data

## Active Material Volume Fractions

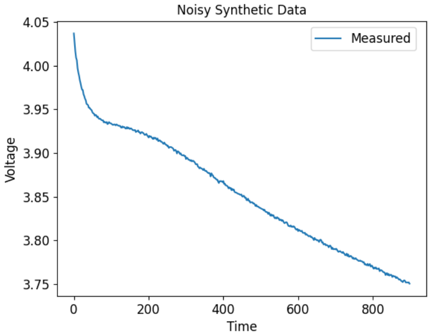
This example fits the active material volume fractions, $\{\epsilon_k\}$ for $k \in \{n, p\}$, using a sum of square errors cost function,

$$\sum_{i=1}^{k} \left(Y_i - \hat{Y}_i\right)^2$$

a Chen2020 parameter set, and a constant applied current of 1C.

The ground truth parameters are:

$$[\epsilon_n = 0.75, \epsilon_p = 0.665]$$



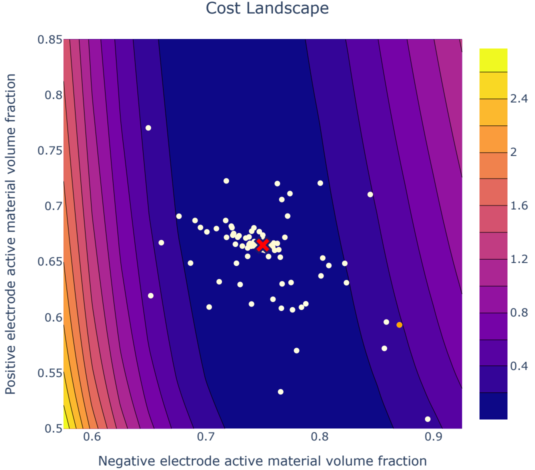
# Example: Non-Gradient based MLE

CMA-ES

We define the problem, cost, and corresponding optimisation with CMA-ES

- $x0_p \sim \mathcal{N}(0.53, 0.01)$
- $x0_n \sim \mathcal{N}(0.825, 0.01)$
- bounds $= [0.6 - 0.9, 0.5 - 0.8]$
- Max iters: 500

Traverses the landscape quickly at the beginning.



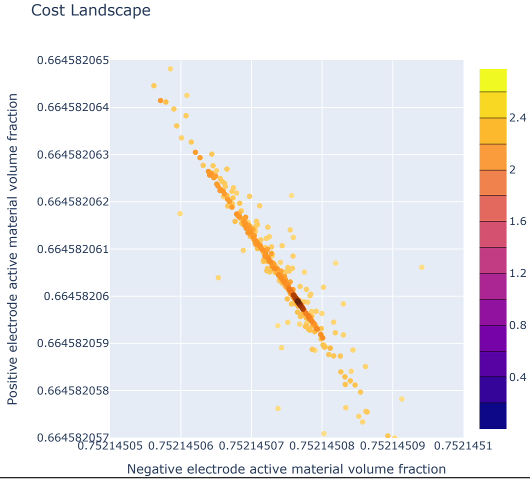
# Example: Non-Gradient based MLE

## CMA-ES

Quickly finds an optimal & spends a large amount of time around this point.

Optimal at:

$$[\epsilon_n = 0.7508, \epsilon_p = 0.6650]$$



# Example: with Gradient based MLE

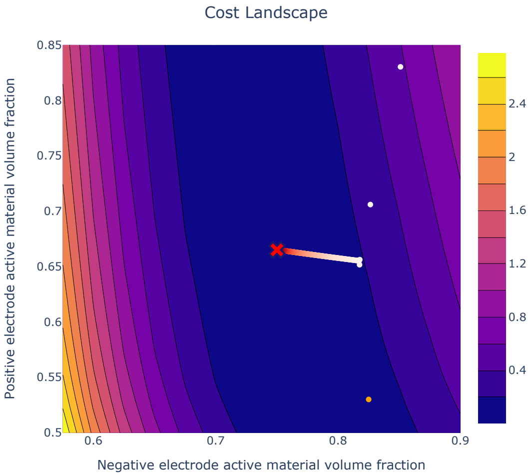
## Gradient Descent ($\eta = 0.01$)

Gradient descent[2] to minimise the cost function, passing $\frac{\partial f}{\partial \theta}$ from the forward model to Pints'.

- $x0 \sim \mathcal{N}(\mu, \sigma^2)$
- Max iters: 500
- $\eta$: 0.01

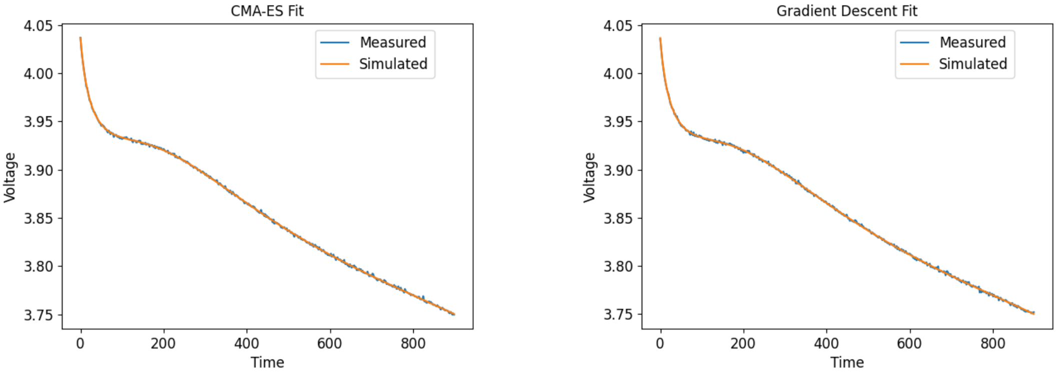
Calculating $\frac{\partial f}{\partial \theta}$ is expensive, and results in slower fitting.

$$[\epsilon_n = 0.7504, \epsilon_p = 0.6651]$$




[a]We run this daily on the cloud!

# Example: Time-Series Comparison



We are currently working on the following:

- Adding bayesian methods: HMC, ABC, etc.
- Implement a version of PEM (Injecting terms into the model structure before discretising)
- Methods to acquire Hessian information
- More workflows & benchmarks for users

# Automated testing is a step change in performance

Consider adding a testing suite to your individual research repositories. It solves a considerable amount of problems:

- Catches bugs / mistypes in your code before publishing results
- Tells you right away when you broke something
- Can be used for benchmarking
- Can format your code for consistency
- It's parametric, and free!