

Python Battery Mathematical Modelling (PyBaMM)

Valentin Sulzer (adapted from Martin Robinson)

October 14th, 2019

What is PyBaMM

- ▶ *Python Battery Mathematical Modelling* (PyBaMM) solves continuum models for batteries, using both numerical methods and asymptotic analysis.
- ▶ Designed as a Common Modelling Framework for the Multiscale Modelling Faraday project. Facilitates sharing and distribution of **model** and **numerical methodologies** .
- ▶ Not a general purpose solver (e.g. Comsol), focuses specifically on battery modelling.
- ▶ Use it for:
 1. Running a battery simulation using one of the pre-built models, either as a single simulation or within parameter estimation
 2. Inserting your own model, discretisation or solver. Compare with the pre-built pybamm components, or to share with other PyBaMM users.
 3. Develop within the PyBaMM framework, or use any of PyBaMM's components within your own code

Organisation

► developers:

Valentin Sulzer



Scott Marquis



Martin Robinson



Robert Timms



Ivan Korotkin



Jamie Foster



Organisation

- ▶ developed as an open source (BSD licensed) Python library on GitHub:
 - ▶ <https://github.com/pybamm-team/PyBaMM>
- ▶ software engineering best practices:
 - ▶ Full suite of unit and integration tests
 - ▶ Automated testing on Linux, Mac and Windows using Python 3.6+
 - ▶ Generated API documentation and example notebooks/scripts demonstrating features
 - ▶ Project management, issue tracking and code review via GitHub

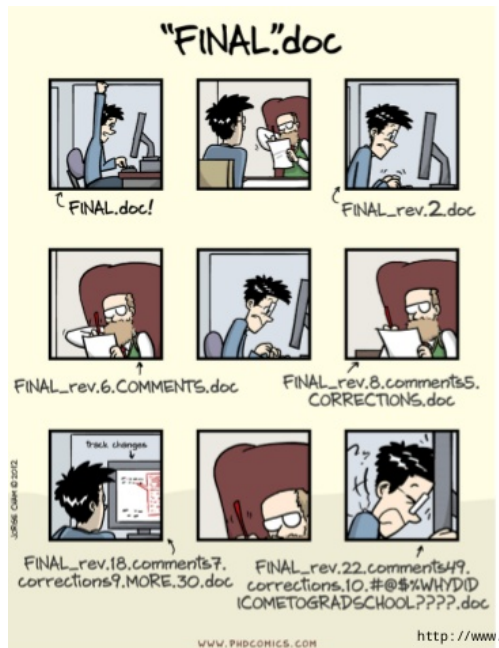
Why have a Common Modelling Framework?

- ▶ Don't reinvent the wheel
- ▶ Use everyone else's hard work
- ▶ Benchmarking: **directly** compare models and solvers
- ▶ Make it easy for others to use your work

Plus:

- ▶ Get feedback on your code and models
- ▶ Seeing other people's code makes you a better coder

Why use Git?



Why use Git?

- ▶ Backups!
- ▶ Version control
- ▶ Collaboration
- ▶ Project management
- ▶ Employability

Why use unit tests?

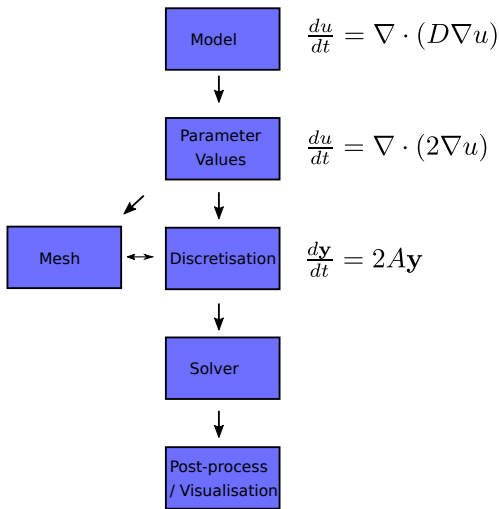
- ▶ Make sure code is doing what you think it's doing
- ▶ Confidence in changes
- ▶ Catch bugs early
- ▶ Simplify debugging process
- ▶ “Green tick” validation
- ▶ Helps figure out what to do next
- ▶ Saves time
- ▶ Very useful when:
 - ▶ codebase is too big to fit in your head
 - ▶ more than 2 people making changes
- ▶ (Not necessary for short-term, single-user/developer projects)

Current features of PyBaMM

- ▶ Lithium-ion battery models:
 1. Single Particle Model (SPM)
 2. Single Particle Model with Electrolyte (SPMe)
 3. Doyle-Fuller-Newman (DFN)
- ▶ Lead-acid battery models:
- ▶ Discretisations:
 1. 1D Finite Volumes (general purpose)
 2. 2 + 1D Finite Elements for current collector domain
 3. Control Volumes for electrode particle domains*
- ▶ Solvers:
 1. Scipy ODE solvers
 2. Scikits ODE & DAE solvers (SUNDIALS)
 3. KLU solver in C++ (in progress)
 4. Dae-Cpp DAE solver (in progress)*

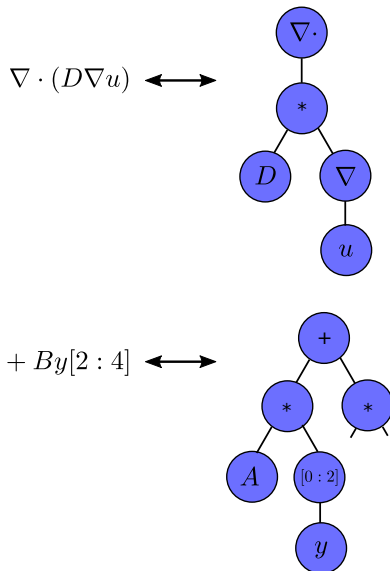
PyBaMM Pipeline

- ▶ PyBaMM's design separates the different stages solving a model, can develop or customise each stage separately
- ▶ Construction of a pipeline is a python script, reuse the stages as you see fit (e.g. comparing different models, re-solving with different parameters, custom plotting, ...)



PyBaMM's Expression Tree

- ▶ Communication between the stages is done via *expression trees* representing equations
- ▶ Different stages operate on these. E.g. Parameter Values walks through tree replacing symbolic parameters with scalars.
- ▶ Pybamm can take *derivatives* of expression trees (i.e. for Jacobians)



Use Case 1 - Adding your own model

1. Create a Python class representing your model¹
2. Define rhs, algebraic, boundary_conditions, initial_conditions, variables as **expression trees**
3. An expression tree is built from building blocks such as pybamm.Variable or pybamm.Parameter, and normal Python operators such as +. e.g., the expression

$$\nabla \cdot D(c) \nabla c + ac$$

written as:

```
c = pybamm.Variable('c')
a = pybamm.Parameter('a')
D = lambda x: pybamm.FunctionParameter('D', x)

expr = pybamm.div( D(c) * pybamm.grad(c) ) + a * c
```

¹see repository documentation for more details

Use Case 1a - Add/Customise your own submodel

- ▶ **Note:** In-built models in PyBaMM consist of sub-models (e.g. electrolyte diffusion or interface kinetics). Often simpler to replace/add a new sub-model rather than write a new model from scratch

Use Case 2 - Using your own parameters

- ▶ Each model has a set of default parameters that you can use, for example:

```
model = pybamm.lithium_ion.SPM()  
param = model.default_parameter_values
```

- ▶ You can change individual parameters in a python script:

```
param['Typical current [A]'] = 1.4
```

- ▶ ... or, the default parameter sets are defined as csv files in dimensional units, so can supply your own to specify an entire parameter set.

Use Case 3 - Plotting additional variables

- ▶ Each model has a list of variables that are calculated from the solution variables. These can be dimensional versions of solution variables or other useful quantities for comparing different battery models.

```
voltage = pybamm.ProcessedVariable(  
    model.variables['Terminal voltage [V]'],  
    solution.t, solution.y, mesh=mesh)
```

```
t = np.linspace(0,1,250)  
plt.plot(t, voltage(t))
```

- ▶ Users can defined *custom variables* using expression trees. These are discretised along with the other equations in the model (can therefore contain derivatives)

Other potential use cases

1. **Custom discretisation:** discretisation is set on a per-domain basis (electrolyte, positive electrode particle, etc.). Can supply your own discretisation for one or more domains. Can be as general purpose or as model-specific as you require.
2. **Parameter estimation:** Those parameters that do not affect the discretisation can be altered and a new solutions obtained just be re-running the solver (i.e. no need to re-generate and discretise your model)
3. **Others..?:** We are keen to hear your ideas! PyBaMM is in a formative stage so keen to receive and incorporate feedback into the library.

Live demo time!!

Summary

- ▶ PyBaMM is a library for describing and solving continuum battery models
- ▶ Exposes a decoupled pipeline design for solving a model, so that users can customise with new models, discretisations, parameterisations, solvers etc.
- ▶ We welcome new developers at this beta testing stage, and are keen to get your feedback, bug reports and suggestions!
- ▶ I'm happy to give a more detailed introduction
- ▶ Get the code, browse the documentation, or get in contact at:
<https://github.com/pybamm-team/PyBaMM>