Overview of Pack SPM Implementation in PyBAMM

*This documentation is very much work-in-progress…*

**Preface:**

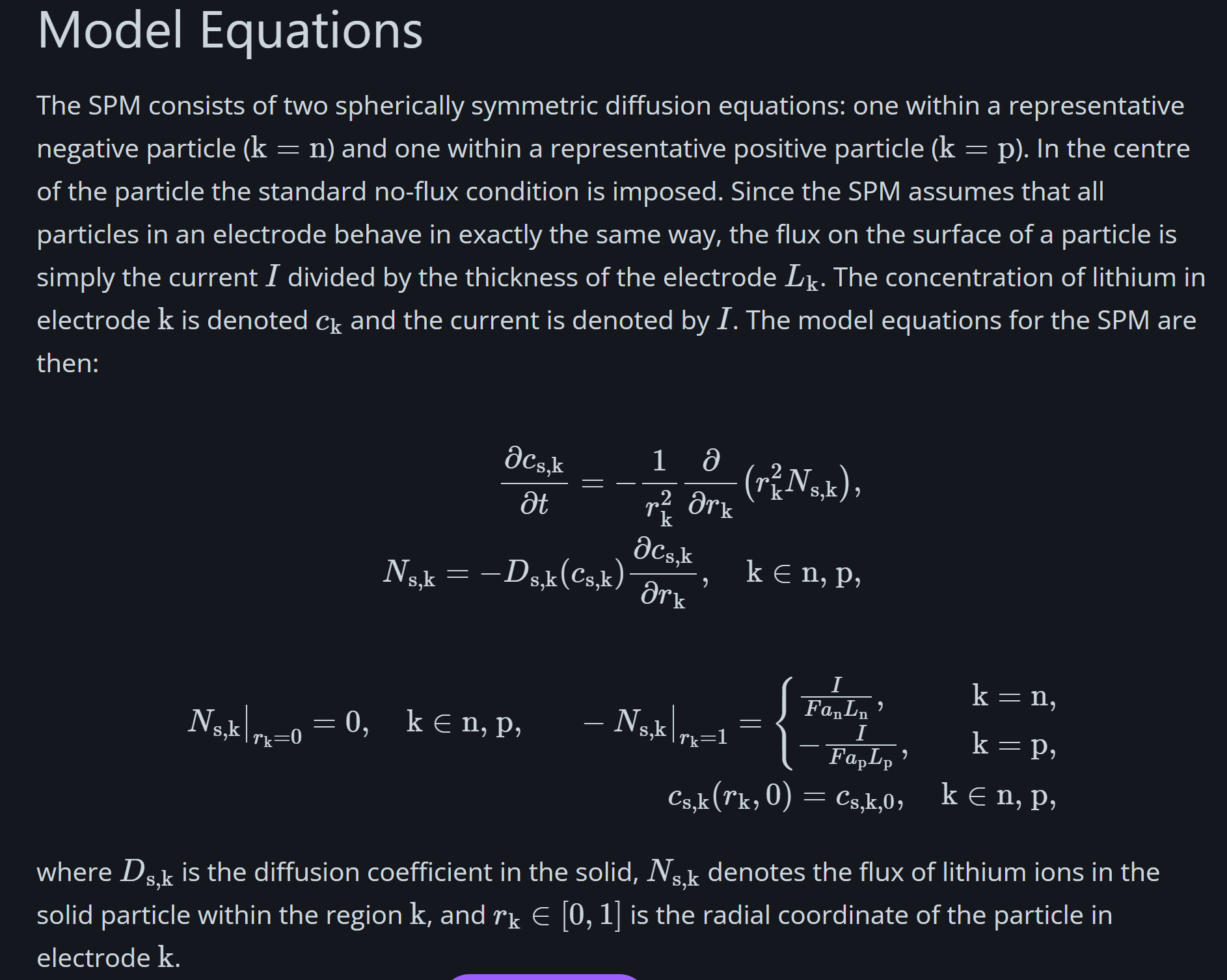
PyBAMM is a Python library for simulating battery models. More generally, the package provides an interface to DAE solvers – differential and algebraic equations can be specified with symbolic variables/parameters and solved using accessible function calls. Several common battery models (SPM, P2D, Lead-Acid) are provided by default in the package.

In this codebase, however, SPM is implemented “from scratch” – passing equations and parameters into the solver tools without using the existing abstractions. Two main reasons for this:

1. An educational modeling exercise – to gain an understanding of diffusion models, governing equations, kinetics, discretization, etc. Using the library’s fancy classes and objects hides most of the implementation details…
2. The PyBAMM API is frustrating! It is difficult to understand where parameters are being passed and how to easily change the model, relevant equations, etc

**Governing Equations:**

*Particle-level:*

**

) - )

*OCP (open circuit potential) given by built-in function*

*J0p and J0n (exchange current density) given by built-in function in marquis.py*

*Ln: electrode thickness*

*An: Area/Volume calculation (see ‘a\_term’ in single\_particle.py)*

**Pack-Relevant Source Code (Top-Down):**

*The code is organized in several layers of abstraction, developed in the process of scaling from particle-level to pack-level modeling. I will review it in a ‘top-down’ approach… hopefully this is an intuitive explanation of the repository.*

**mainmodel.py:** ’Main’ program which initializes Packs or Cells – runs cycling and any plotting/post-processing routines

* *Refer to the comments in this program*

**pack.py:** Builds X Cells in series, Y Cells in parallel (X, Y are input params).

* Builds the DAE for the pack – all algebraic equations for coupling
* Includes discretization method and cycling protocol

**cell.py:** Couples 2 Single\_Particles (which consistute one ‘Cell’)

* Cathode called Positive Electrode (+)
* Anode called Negative Electrode (-)

**single\_particle.py:** Defines model of a single electrode.

* Diffusion ODEs, boundary/initial conditions for Lithium concentration, and Butler-Volmer calculations for electrode potential
* Global meshing geometry, parameter structures are updated
* *Note that this program can be run standalone to validate the ‘charge’ / ‘discharge’ of a single particle.*

**‘Helper’ Source Code:**

**params.py:** Defines cell-level parameters (see ‘VariedParameter’)

* VariedParameter is a structure used for randomizing parameter values between cells. For example, this line of code:
* POS\_ELEC\_POROSITY = VariatedParameter.from\_percent(0.50, 5)
  + *The positive electrode’s porosity will be 0.5 +/- 5% across all cells in pack*

**marquis.py:** Copied from the PyBAMM library’s source code repository. It provides empirical functions for OCP and exchange current density – used to determine electrode potential (via Butler Volmer eq).

**consts.py:** Defines physical constants (like temperature, Faraday constant, gas constant, etc). *Can be ignored…*

**Some Footnotes:**

1. This repository/API for pack-modeling is meant to be parameter-set-agnostic. We are using ‘Marquis-2019’ – a set provided by PyBAMM for Li-Ion coin cells. Results from my code have been cross-checked by PyBAMM’s simulation output (for this parameter set). We can potentially validate against other sets to ensure robustness.