BUILD AND PUBLISH PYBAMM-COOKIECUTTER AS A TEMPLATE FOR NEW PYBAMM-BASED PROJECTS

GSoC Project proposal for PyBaMM by Santhosh Sundaram

Personal Details and Contact Information

- GitHub Username - santacodes
- Email - santhoshsundaram9650@gmail.com
- University - Anna University, Chennai, India
- Time-zone - UTC+5:30 (IST)
- Address - Chennai, India
- IRC nick - santacodes

Synopsis

This project aims to build and publish python-cookiecutter templates for new PyBaMM-based projects. This template, upstream at https://github.com/pybamm-team/pybamm-cookiecutter/, originally was a part of GSoC 2023 and is designed to simplify the setup process of the development environment for researchers and scientists interested in utilizing PyBaMM for battery modeling but may lack familiarity with managing Python environments or repositories. The project intends to enhance the accessibility and usability of PyBaMM for newbies and experienced users alike. Providing a standardized template with best practices and automation tools, would lower the barrier to entry for adopting PyBaMM in battery modeling projects. This would in turn make battery modeling more accessible, efficient, and collaborative for the research community which could potentially increase the user base of PyBaMM.

Potential mentors for this project would be Saransh Chopra, and Agriya Khetarpal, who were the initial contributors to the cookiecutter template repository, additionally Arjun Verma, and Ferran Brosa Planella as per the PyBaMM GSOC reference site.

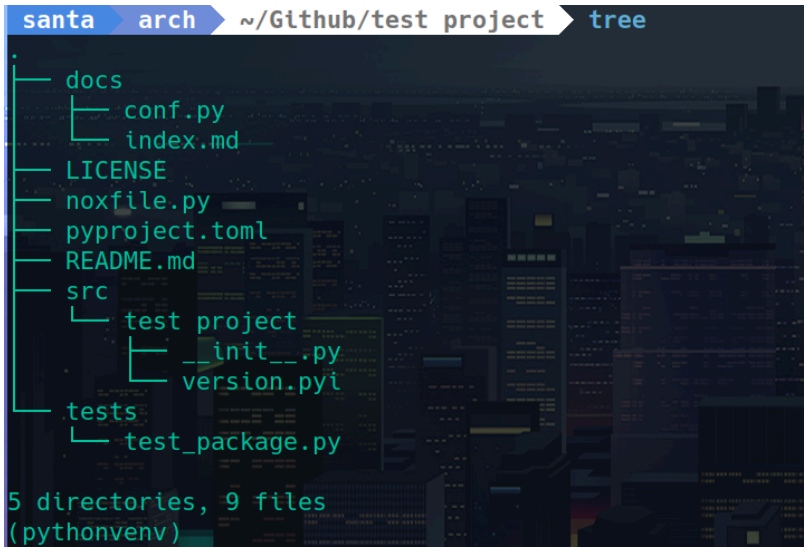
Benefits to the Community

PyBaMM’s cookiecutter template would help scientists and researchers, especially those with limited knowledge of Python environments or repositories find PyBaMM easier to use. It could potentially grow PyBaMM's current contributors' community and general usage as a result of its increasing simplicity and utility.
PyBaMM's already established automation infrastructure and workflows, and best practices, present in the template would make it easier to start and set up new projects. This would simplify the procedure and free up researchers to concentrate more on experimentation and modeling than wasting their precious time on setting up development environments.

The availability of a user-friendly template will help in future extensibility with PyBaMM as it will include features such as Model entry points which would allow community users to define their models using and use PyBaMM’s established GitHub infrastructure to run, test, and deploy their models.

Current Status of the Project

Currently, the PyBaMM cookiecutter template is a basic established template that works and can be considered as the groundwork. The base directory structure has been initialized and the basic configurations have been set up and look like this -



There is no provided documentation for installation steps to set up the development environment from the cookiecutter template.

Several additions and improvements can be implemented to make it a full-fledged ready-to-use template so that it can be packaged and utilized by the user base which is discussed in the following Goals section.

Goals

Currently, the upstream repository for the PyBaMM cookiecutter template lacks the following things, with some being major and some being minor. I would sort the following tasks based on their priorities -

Goal 1

- Add functionality for **Model entry points** (major)

  Originally mentioned here, the cookiecutter template should allow community users to create their own models with their own licenses and specifications.
  The lack of contributors adding models to PyBaMM stems from complexities in its submodel structure, uncertainty around ownership when adding models directly to PyBaMM's repository, and potential disagreements. To eliminate these issues, it was proposed to implement "Model entry points", allowing community contributors to create and share models of their repositories using the cookiecutter template without directly adding them to PyBaMM. This would not only let community contributors retain ownership and choose license terms but also grant flexibility to the PyBaMM team in supporting models. Including all of GitHub's functionality and infrastructure contained within PyBaMM in the template repository, including issues, discussions, and pull requests, would build a collaborative community. The implementation would involve making necessary changes to PyBaMM to support model entry points, defining entry points for models, and creating a cookiecutter repository for streamlined setup, testing, documentation, and versioning, laying the groundwork for future developments separating the model and PDE code within PyBaMM. Model entry points then need to be bootstrapped using a mapper to the models in `pyproject.toml` similar to PyBaMM's parameter sets.

- Adding **Documentation** (major)

  Pybamm cookiecutter template lacks documentation for the usage and setting up of the development environment for users. The documentation can be implemented through either `sphinx or jupyter notebooks` to directly initialize the development environment. As originally stated in - #issue 5, the documentation should provide technical details such as - Installation steps, command line prompts that could be linked to additional functionalities and procedures, a user guide to provide details on styles and

naming conventions, data paths linked to a directory within the template (originally discussed at Data path discussion), guide to parameter sets and third-party parameter sets, etc. In addition to that details surrounding the infrastructure and testing should also be added and also for deployment models such as docker if decided upon execution.

Goal 2

- Improvements in the directory structure (major)

  The present directory structure is quite simple and does not resemble the original proposed directory structure for the project mentioned in #issue 1. Optional directories can be flagged using the command line prompts and a more organized directory structure should be incorporated into the project template.

- Publishing, CI/CD pipelines, and testing (major)

  The template currently only resides in the pybamm-cookiecutter repository and is not published on PyPI. It also lacks CI/CD testing and deployment infrastructures. Multi-container testing for unit and integration and pipelines for publishing the template to PyPI testing should be implemented.

- Support cruft and copier (major)

  cruft already fully supports cookiecutter templates while copier might be made to support cookiecutter templates using its own project specifications which use YAML instead of JSON as mentioned here.

Goal 3

- Creating better support for build systems (minor)

  Hatch is the default backend builder in the current cookiecutter template and other backed builders like flit and setuptools could be implemented and used by selecting through command line prompts.

- Optional dependencies prompts (minor)

  During the template project setup procedure, command line prompts can be added to flag optional dependencies under [dev] and [docs]. This would make the template more customizable and a user can minimize the project defaults as per their needs.

Deliverables

The main goal of this deliverable model is to get the important things done and running. Once established a strong base, features, and other improvements should be implemented in a waterfall-like model.

Deliverable 1 -

- Making **model entry points** with a defined directory structure
- Initial **documentation** with installation and setting up procedures

Deliverable 2 -

- Supporting systems for **cruft and copier** templating tools
- **CI/CD Pipelines** for Publishing into **PyPi** and for **Unit & Integration testing**

Deliverable 3 -

- **Command line prompts for cookiecutter template** for more customizability and flagging optional dependencies
- Adding more **backend build systems** and implementing bootstrapping methods

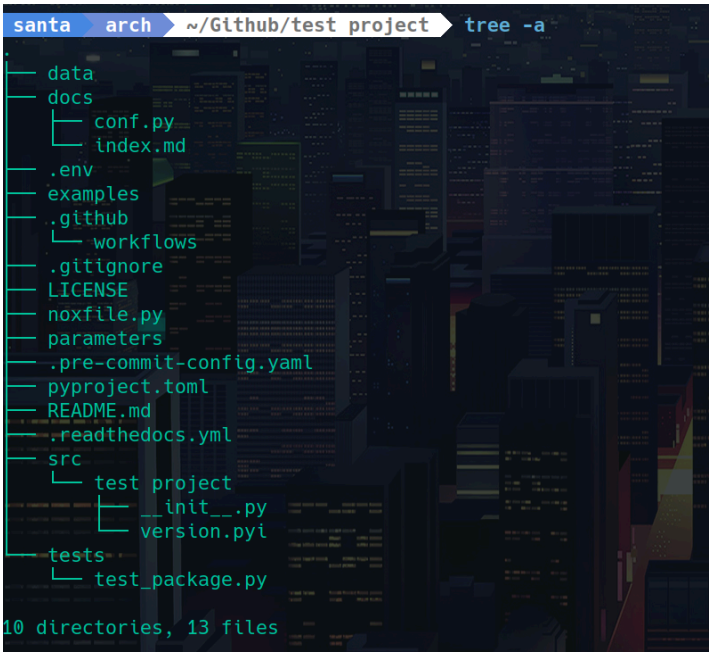
Expected Results

Upon completion of the project at the end of the coding period, PyBaMM community can expect the following outcomes -

- A **Model entry point** for the PyBaMM community for the users to define and implement their own models.
- A **complete cookiecutter template** for PyBaMM, packaged and deployed to PyPi.
- A **user guide and documentation** for the usage of the template.
- **CI/CD pipelines and workflows** to not only check the correctness of the template but to also test the models added to it by the contributors.

Approach

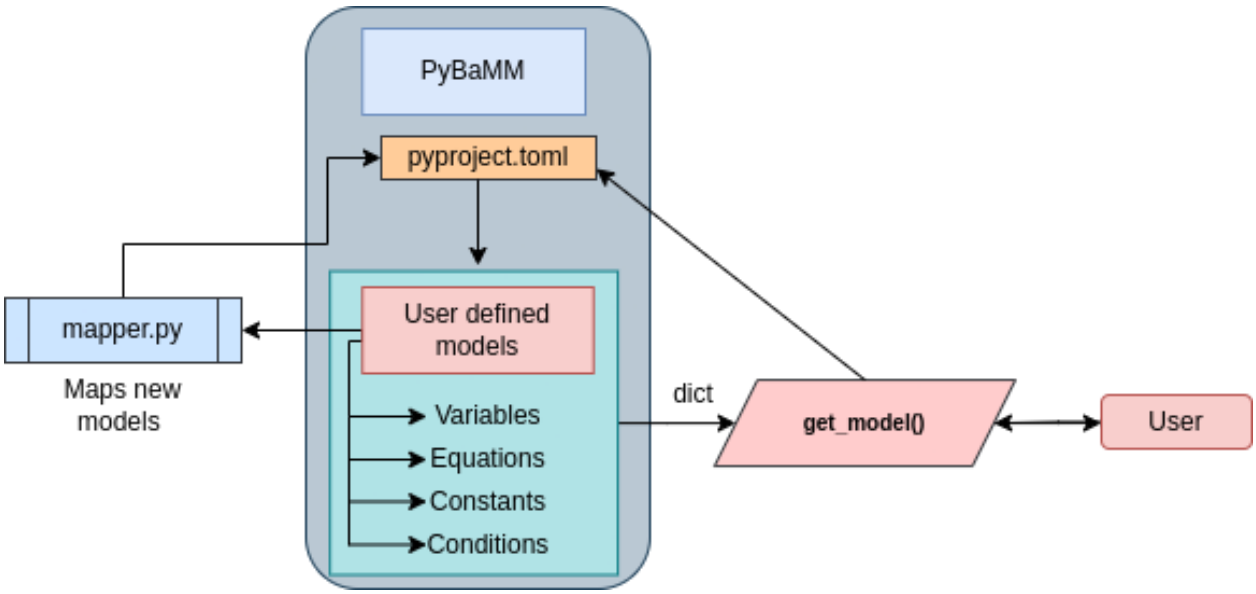
The first and foremost operation to perform would be to create a well-defined directory structure for the project such as the one mentioned below.



The project template should use PyBaMM as a dependency for the community users to use it and define their model entry points. The project clones should be runnable across different systems with the same dependency stack, therefore making it portable. This would be the first building block of a cookiecutter template as it would just provide a defined structure for the development environment of model entry points and then further bootstrapping it to be shared within the community.

With the template structure defined, we could implement it in cookiecutter with `pyproject.toml` as we could define a directory structure like this to make it more organized and extensible for the community users. By adding prompts in `cookiecutter.json` we could also make some directories optional for example the test directory.

Implementing Model Entry Points (PoC)



Creating entry points for the model -

```
[project.entry-points."pybamm_models"]
Example_model =
"pybamm.models.full_battery_models.test_model.test_model:get_model"
```

Getting the models from `pyproject.toml`

```
importlib.metadata.entry_points(group=group_name)
models = dict()
for entry_point in self.get_entries("pybamm_models"):
            self.models[entry_point.name] = entry_point
```

Loading the model from entry points

```
model.load = models[model_name]
# this would either return a dict() or a json object
```

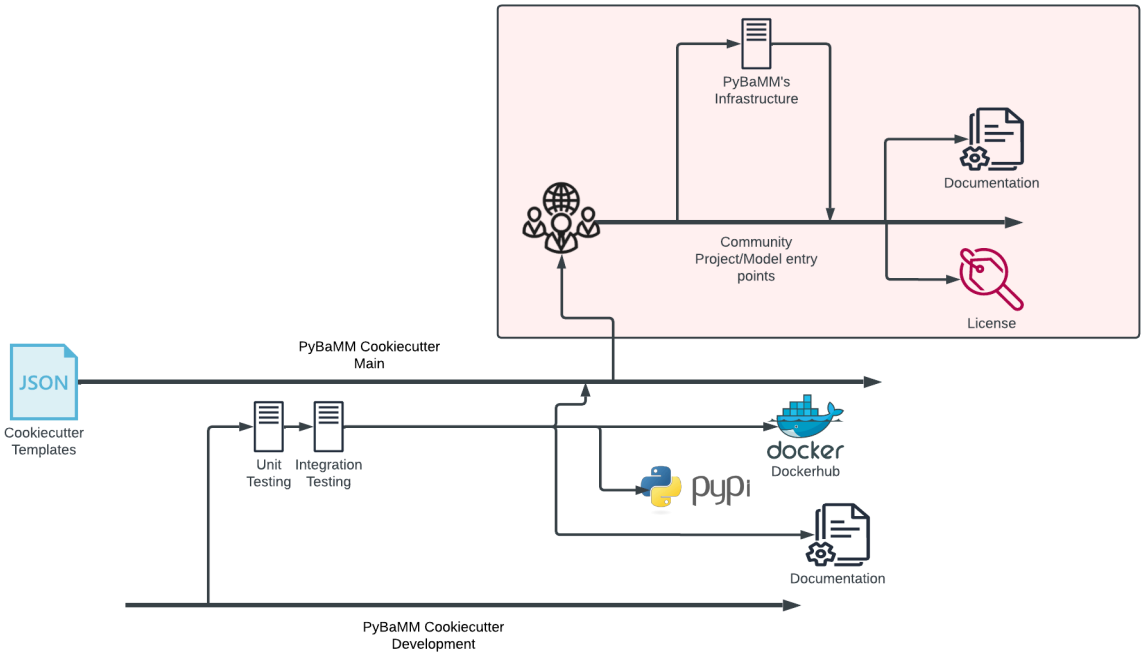
Documentation could be implemented through `sphinx` and `nbsphinx/myst-nb` for both usage and installation documentation as well as user-defined documentation for portraying use cases and examples for their own battery models (model entry points). The documentation builders are already defined in `pyproject.toml`. As for the user guide for the cookiecutter template, the documentation can be either added to PyBaMM documentation or implemented separately in the PyBaMM cookiecutter documentation.
The user guide would contain initialization procedures for installing and using `python-cookiecutter`, followed by grabbing and using PyBaMM's cookiecutter template for personal use. Example -
`$ cookiecutter pybamm-cookiecutter/`

Model entry points could be implemented similar to the parameter sets in PyBaMM. Using self-contained model classes in the project template, which can be further used by community users.
Calling the model with the author name or the model name would return a `JSON` object or a `Python Dictionary` with key and value pairs as discussed here.
Similar to the parameter sets we could create a directory to define the models which could be further mapped in `pyproject.toml` to declare the models.
This could serve as a bootstrapped model for the community users to define their models and use the models within the community without them being part of the parent PyBaMM repository.

Further after defining the template structure, CI/CD pipelines and workflows should be implemented to package and publish the cookiecutter template to `PyPi`. For testing, `pytest` could be used as a parallel GSOC 2024 project that is involved in migrating PyBaMM's testing infrastructure to pytest this year in the parent repository. The testing infrastructure should not only accommodate the existing test cases but also ensure the community users define their test cases for their models in the GitHub workflow infrastructure.

Other minor changes may include adding user-friendly prompts to the cookiecutter template to make it more extensible and customizable as per the user's need. For example, the user could choose the backend build systems out of the options `hatch, flit, setuptools,` and available licenses. A user could also be made to choose to install optional dependencies such

as `jax or odes solvers`. Other customizable prompts might also include `mypy,pre-commit hooks, ruff, ruff format, local/pygrep hooks, blacken-docs etc.`
For type checking, `PyBaMM's pre-commit hooks` can be included within the project template by default or maybe a command line prompt to include it can be implemented for better customizability.



Proposed Project Architecture

Timeline

Period	Task
After proposal submission	• Taking relevant issues and solving them to get familiar with the code base better. • Getting familiar with parameter sets to implement similar features in `Model entry points`.
May 1 - 26 Community Bonding Period	• Knowing project requirements for initial `packaging`. • Setting up the requisite structure for implementing `Model entry points` inside a project.
Week 1 and 2 May 27 - June 10	• Adding the initial documentation for installation and setting up the development environment. • Getting to know the changes that need to be incorporated to implement `Model entry points`.
Week 3 and 4 June 10 - June 24	• Working on implementing model entry points and documenting the user guide for it. • Add guidelines to define and declare model sets and map them in `pyproject.toml`
Week 5 and 6 June 24 - July 8	• Writing tests for the template using `pytest`. • Incorporating unit and integration tests in GitHub workflows.
Mid-term evaluation period	July 8 - July 12
Week 7 and 8 July 12 - July 26	• Publishing the template as a package on `PyPi` using CI/CD pipelines. • Supporting `cruft and copier` (alternate template builder tools).

Week 9 and 10 July 26 - August 9	• Supporting other backend builders and adding more customizability to the template.
Week 11 and 12 August 9 - 26	Concluding the project working on missing minor features and issues and delivering the template and its documentation for community usage.
Final student evaluations	August 26 - September 2

As per the NUMFOCUS guidelines I will also be posting a total of `6 blogs`, i.e., one blog each in detail every two weeks on my personal website to record the progress, changes I made, and the next goals I will be tackling. If I were facing any issues, I would also document them and explain them with technical details.

About Me

I am Santhosh Sundaram, pursuing Bachelor of Engineering in Computer Science at Anna University, Chennai. Programming has been my long-driven passion since I was in high school, from my early projects written in BASIC to me contributing to open-source projects, my trait of accepting and adapting to knowledge has stayed persistent. This trait has led me to use the opportunity to work with various kinds of technologies.
In my high school and freshman years, I was an active freelancer on Fiverr and used to make applications and automation scripts using Python and JavaScript. Later in my sophomore year, I worked on a cross-modal deep learning model for the re-identification of a person between IR and RGB images and wrote a research paper on it. At the end of the last semester, I made a server monitoring dashboard for IBM UNIX-AIX servers for an organization's data center and implemented asynchronous techniques in .NET MVC to reduce the latency and quicker update times and wrote shell scripts to fetch and relay server health data.

I am relatively new to the PyBaMM community and familiar with the workflow. I have yet to get acquainted with the core PyBaMM code base but I have familiarised myself with the infrastructure and contributed and solved some relevant issues -

Contribution	Status
PR: Migrated docker image from miniconda to manylinux2014	Closed and drafted another PR due to this issue
PR: Migrated from miniconda to official python bullseye docker image	Drafted and Open
PR: Added pybamm user id to 1000 and set its group to root	Open and work in progress
PR: Added a weekly schedule on needs-reply workflow	Merged
PR: Improved distribution specific documentation for linux and removed old dependencies	Merged
Issue: Update source installation instructions for linux to include more distro-specific instruction	Closed

My plans after GSoC

GSoC was a great opportunity to find reasons to contribute to open-source. Once I had started contributing I immediately fell in love with the process and it kept on driving me to contribute more. I plan to keep on contributing to open-source and OSS and not limit myself but adapt more to upcoming technologies to make the code base better with each commit.

Why you chose to work with PyBaMM

Mathematical computing and computational research in any relevant field have always been interesting to me. The sheer complexity of heavy mathematical modules was always intriguing from my perspective. PyBaMM provides a medium of combining battery modeling and mathematical computation to simulate battery models and their parameters which would not only help me acquire knowledge about the code base itself but also the principles of working battery models which I am not acquainted with yet. The potential knowledge I could gain from contributing to PyBaMM would help me progress, learn, and push my limits of aptitude by tackling new problems. Technical skills such as CI/CD, packaging, and deployment would also benefit me in the long run as they would contribute a lot to my professional career ahead.

How you plan to stay on track and finish the project successfully

I would allot myself up to 30 hours per week to work on this project and commit to it. The mentioned duration for this project is 175 hours and I ideally would wish for my deliverables(mentioned here) to end before those allotted hours as it would give me additional time to improve and refine the features. I would stick to the timeline and abide by the waterfall model to keep checking the goals one after one until full completion.

The amount of time you would be able to devote to the project

In total, I would be able to devote 30 hours a week to this project and will be available for calls or meetings from 6 PM (18:00) IST to 12 AM (00:00) IST that is, from 12:30 PM (12:30) GMT to 6:30 PM (18:30) GMT. I have not yet received my end-semester examination schedule from my university which might hinder my availability slightly during 2-3 weeks in between the community bonding/coding period. I would work extra hours to compensate for that before and after my examinations.

Other commitments during the summer, if any.

Currently, I do not have any commitments during the summer, if any I'll make sure to notify.