The Journal of Open Source Software

PyProBE: Python Processing for Battery Experiments

Thomas Holland[1,2], **Daniel Cummins**[3], **and Monica Marinescu**[1,2]

1 Department of Mechanical Engineering, Imperial College London, United Kingdom **2** The Faraday Institution, United Kingdom **3** Research Computing Service, ICT, Imperial College London, United Kingdom

DOI: 10.21105/joss.07474

Software

- Review
- Repository
- Archive

Editor: Sophie Beck

Reviewers:

- @TomTranter
- @ritesh001
- @jakobhaervig

Submitted: 16 September 2024
Published: 04 February 2025

Summary

PyProBE (Python Processing for Battery Experiments) is a Python package to process experimental data in a standardised and reproducible way. Recognising that battery experiments are often derivatives of standard procedures, PyProBE simplifies data processing to improve performance and transparency. It was written with the following objectives:

1. Ease of use for those with little Python experience, with a natural language interface.
2. Accelerate battery data exploration with quick plotting and data visualisation, using Polars under-the-hood for rapid DataFrame manipulation and a graphical user interface (GUI).
3. Encourage transparent storage of battery data, with human and computer-readable README-files and a readable API syntax.
4. Host a library of post-processing methods, so techniques can be easily added and compared within a standardised code structure.

Statement of need

Multiple researchers have published the tools that they have developed to perform analysis of experimental data. cellpy (Wind et al., 2024) reads data from multiple battery cyclers and includes built-in methods for techniques like Incremental Capacity Analysis (ICA) and summarisation. It filters data by passing specific cycle and step numbers as arguments to methods, but other filtering must be done manually using Pandas (The pandas development team, 2020).

BEEP (Herring et al., 2020) enables efficient filtering of experimental data with their "structuring" approach to assemble summary statistics for machine learning. It does not include methods for more detailed analysis, such as plotting ICA curves or performing degradation mode analysis. DATTES (Redondo-Iglesias et al., 2023), in MatLab, is able to read from multiple cyclers and includes analysis functions such as ICA and anomaly detection. The user interface of DATTES is a single function that takes character codes such as 'GC' (for graphing capacity) to access functionality.

In the Python community for battery research, PyBaMM (Sulzer et al., 2021) has gained large developer support as an open-source environment for physics-based battery modelling. While Python packages for battery experimental data processing clearly exist, none have yet gained similar support. PyBaMM was written from the ground up to provide an open, modular framework to battery researchers, whereas cellpy and BEEP were written as tools for specific projects. DATTES being written in MatLab (a proprietary development ecosystem) inevitably limits its attractiveness to the open source community.

Filtering a dataset to the section of interest is the first step of all data processing tasks, but can be time-consuming and cumbersome. Researchers often write new scripts for each experiment,

in tools like MatLab, Python Pandas or Excel. These scripts are often not intended for sharing, so they may be difficult to read by others, which slows down the exchange of data and methods between researchers. Like PyBaMM, PyProBE has been written to be more user-friendly than existing tools, making it usable for those with little Python experience.

PyBaMM includes a library of interchangeable models, that allows users to test different approaches. There is no equivalent for interchanging methods for battery data processing, causing duplication of effort among researchers. A need therefore exists for an open-source data processing package where researchers can develop new analysis tools within a single framework. PyProBE's **analysis** module is written to be modular and intuitive, with a consistent data structure and built-in data validation with Pydantic (Colvin et al., 2024). As new methods are developed, they can be added and instantly compared to existing approaches.

PyProBE Operating Principles

Importing and Filtering Data

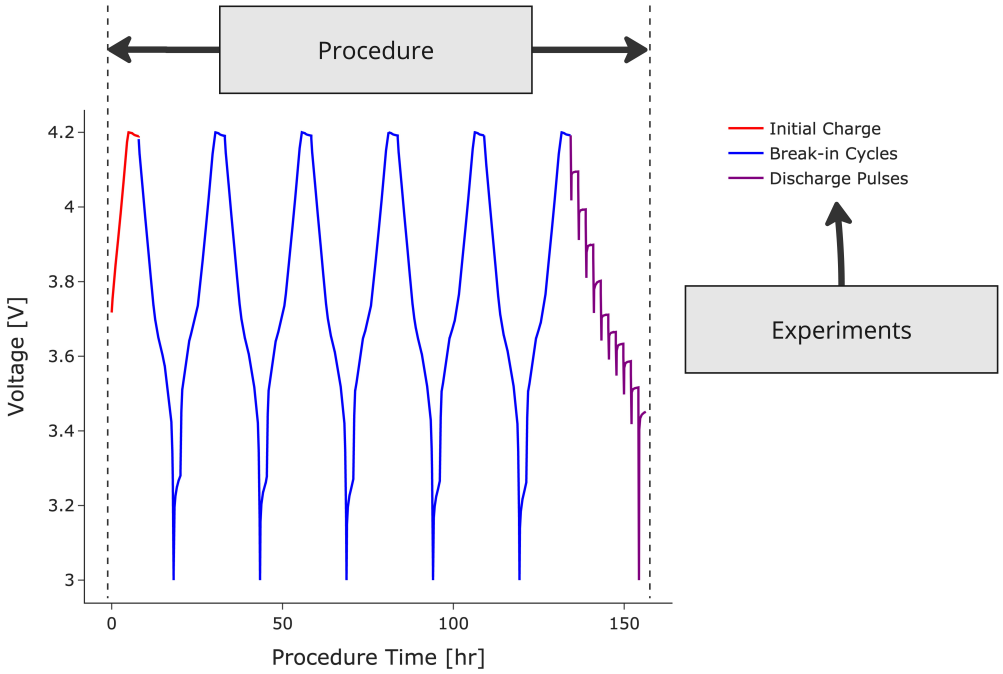

Figure 1: The definition of *procedure* and *experiment* in PyProBE.

PyProBE has a hierarchy of objects containing experimental data. A **Cell** object contains all the data associated with a physical battery that has undergone electrochemical testing. A **Procedure** contains data associated with a particular programme run on a battery cycler. It usually represents data from a single results file produced by the battery cycler. The details of the procedure are included in a README.yaml file stored alongside the experimental data.

Including a README file alongside experimental data is good research practice in accordance with the FAIR principles (Wilkinson et al., 2016). The PyProBE README includes descriptions of cycler processes in PyBaMM Experiment format (Sulzer et al., 2021). This means it is human-readable and enables integration with PyBaMM for running simulations that correspond with the experimental data. The procedure can be split up into **Experiment** objects which can then be referenced in PyProBE's hierarchical filter structure. An example README file is shown in Figure 2.

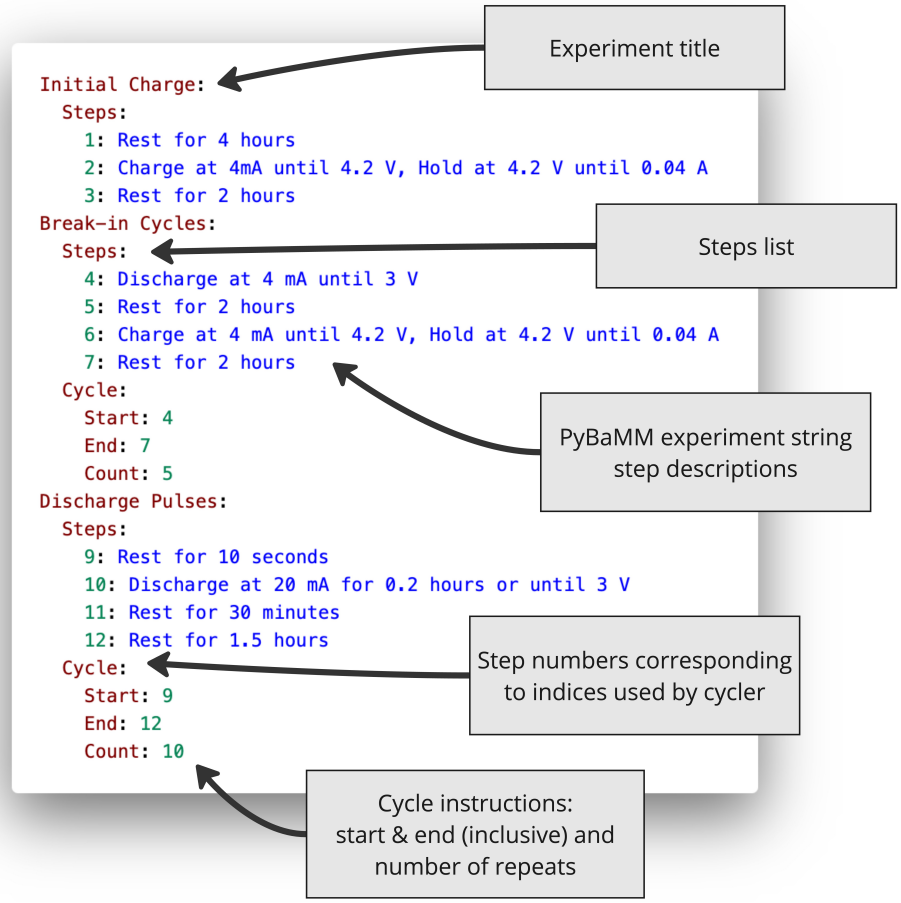

Figure 2: An example README.yaml file for the procedure in Figure 1.

Once imported into a PyProBE **Procedure**, individual cycles and steps can be indexed, with separate methods for accessing charge, discharge, rest etc. processes. All filtering methods produce a **RawData** object type which can be used for further analysis. The complete structure of PyProBE can be seen in Figure 3.

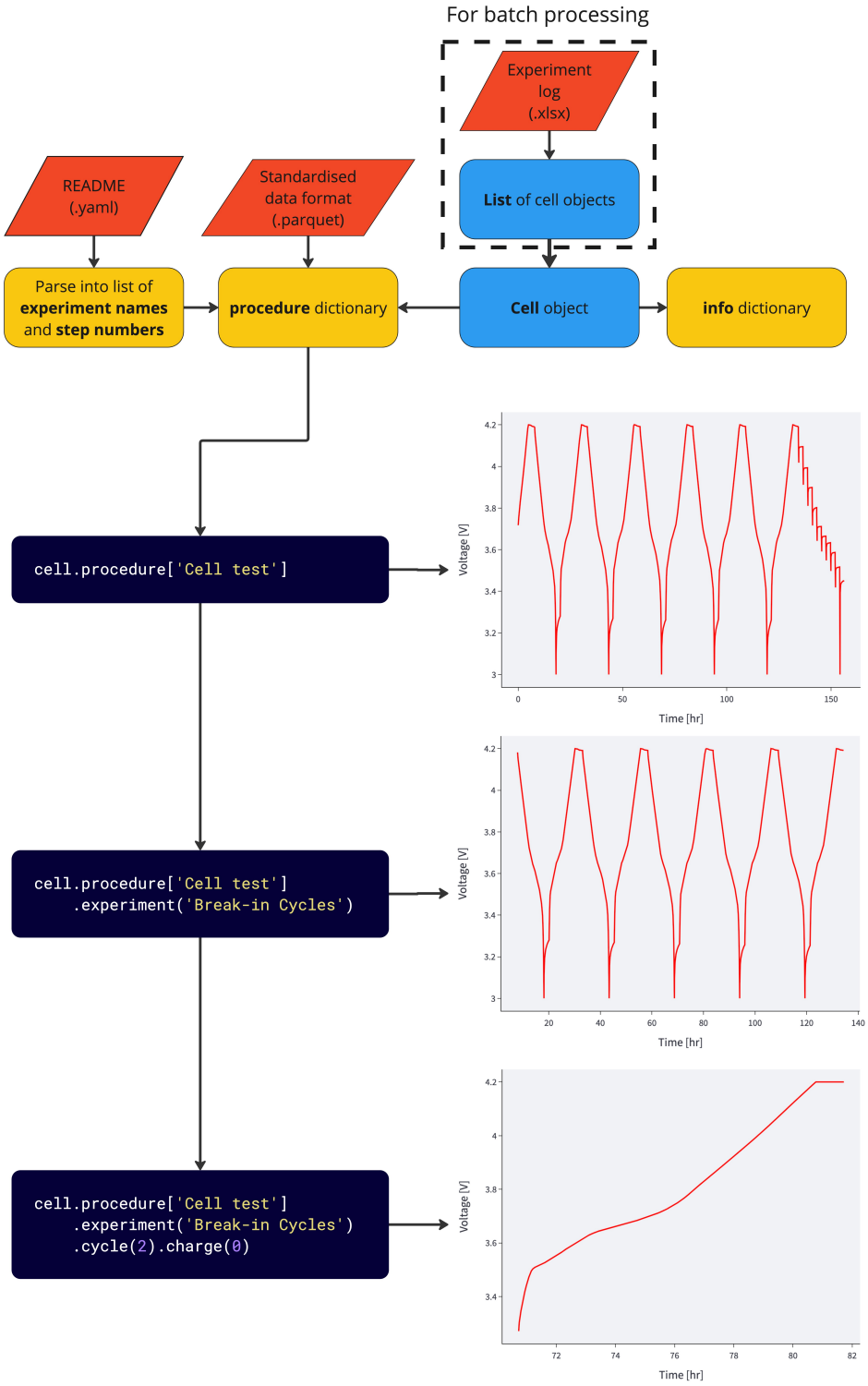

Figure 3: The structure of PyProBE, showing the imported data and setup files (red), PyProBE objects (blue and yellow) and example filtering queries.

Post-processing tools

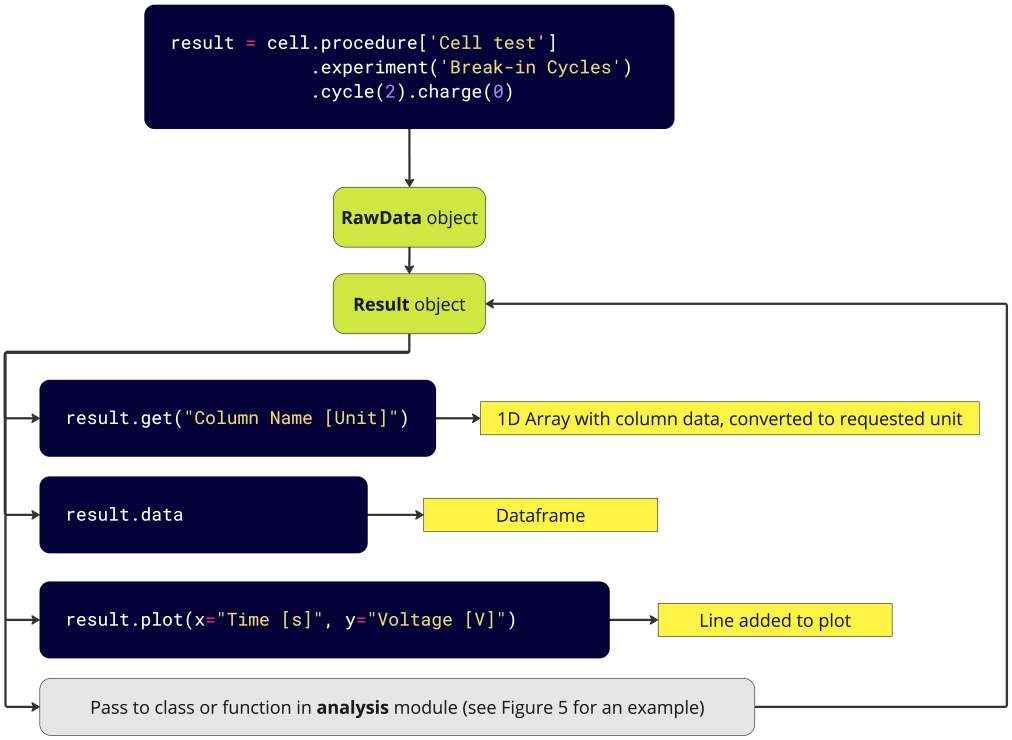

Figure 4: Some of the methods that can be called on **Result** objects, and the objects that they return.

Figure 4 shows how PyProBE **RawData** and **Result** objects can be used. **RawData** DataFrames contain only the columns of the PyProBE standardised format, while **Result** DataFrames contain any data columns produced from further analysis.The data stored in PyProBE **Result** object can be returned as a NumPy (Harris et al., 2020) array or Polars dataframe for further manipulation. They can also be immediately visualised with built-in plotting methods for matplotlib (Hunter, 2007) and hvplot (Rudiger, 2024), or passed to an included wrapper for seaborn (Waskom, 2021).

The analysis module contains classes and functions which, when passed a **Result** object, enable additional functionality. The steps to smooth voltage data before differentiation are described in Figure 5.

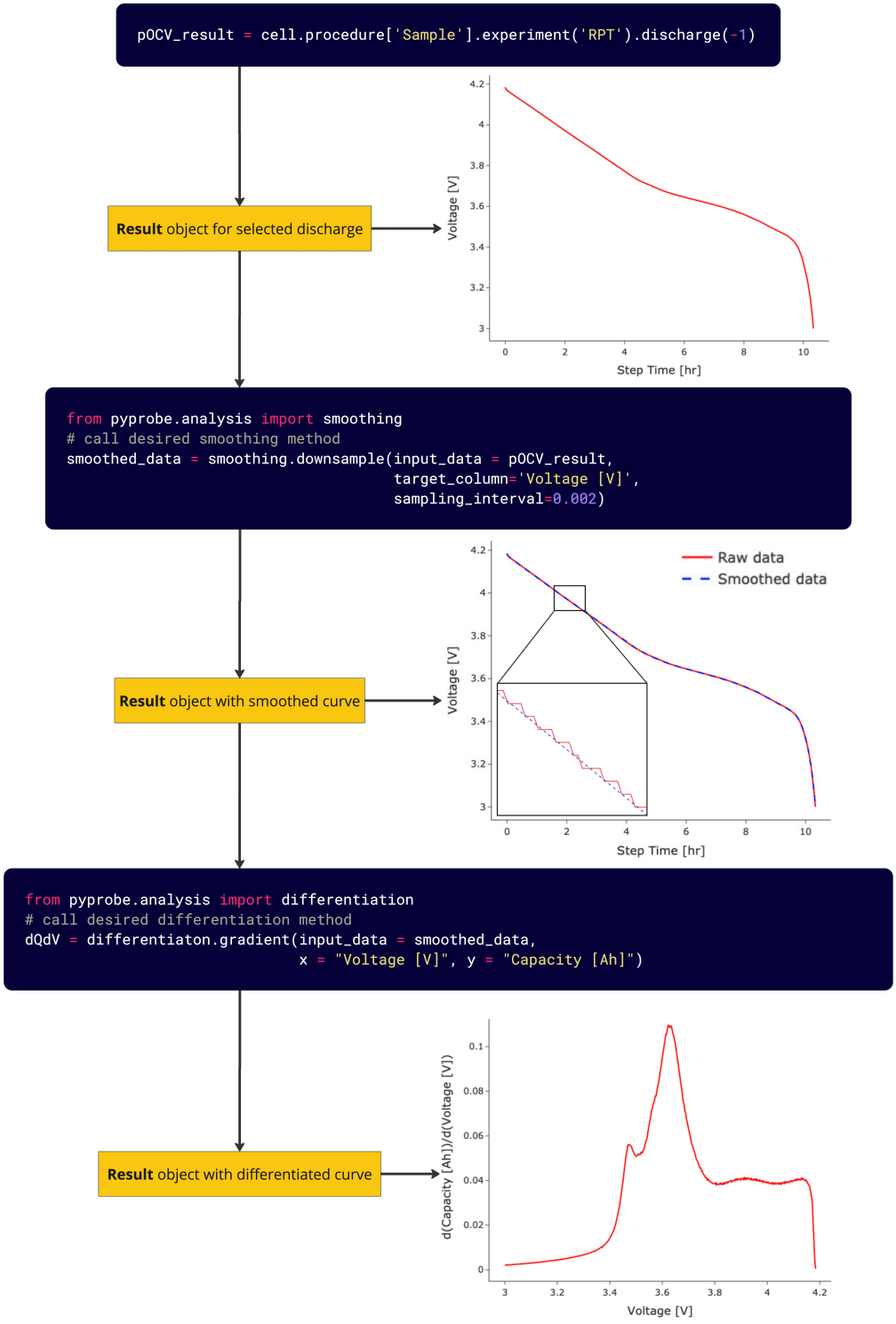

Figure 5: The workflow for smoothing and differentiating voltage data in PyProBE. The complete code can be seen in the "Differentiating voltage data" example.

Performance

PyProBE is faster than manual filtering with Pandas. PyProBE uses Polars (Vink et al., 2024) for DataFrame manipulation and .parquet files for data storage. Polars allows for Lazy

computation, which optimises execution of DataFrame manipulation by delaying until a subset of data is requested by the user. Figure 6 shows how for a week of data PyProBE is 2.7x faster than Pandas v2.2.2 at filtering to a particular discharge in a sample dataset, but for a year of data this increases to 52.7x. This was averaged over 100 runs on a 14 inch MacBook Pro with M2 Pro and 16Gb RAM. The code for this benchmark can be found in the "Comparing PyProBE Performance" example.

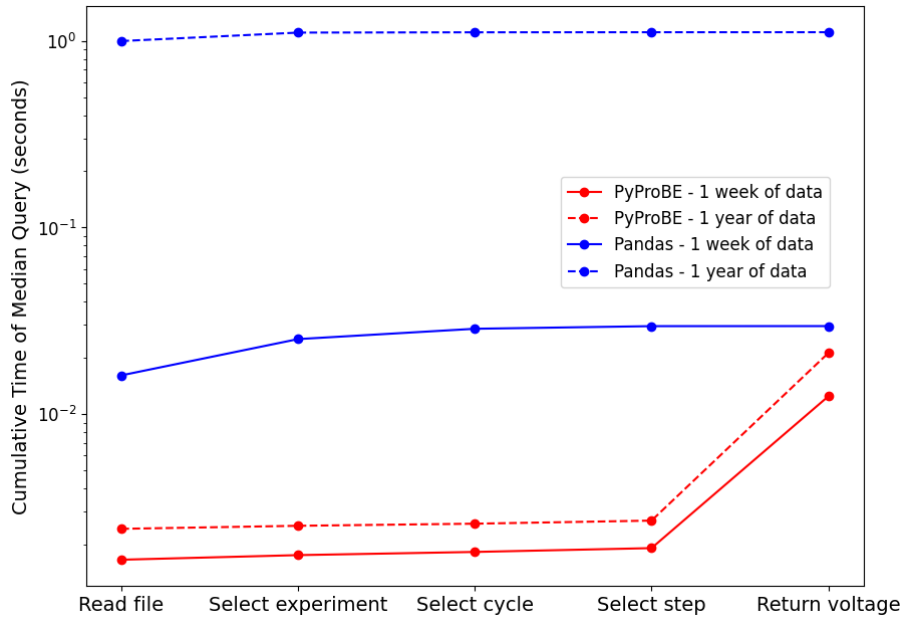

Figure 6: Comparison between PyProBE and Pandas for importing, filtering and returning a DataFrame when reading from a `.parquet` file with an average sampling rate of 1 Hz.

Acknowledgements

This work was generously supported via an EPSRC CASE (EP/W524323/1) award by Rimac Technology, as well as the Faraday Institution Multiscale Modelling project (EP/S003053/1, grant number FIRG025).

References

Colvin, S., Jolibois, E., Ramezani, H., Badaracco, A. G., Dorsey, T., Montague, D., Matveenko, S., Trylesinski, M., Runkle, S., Hewitt, D., Hall, A., & Plot, V. (2024). *Pydantic* (Version v2.10.4). https://docs.pydantic.dev/latest/

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herring, P., Balaji Gopal, C., Aykol, M., Montoya, J. H., Anapolsky, A., Attia, P. M., Gent, W., Hummelshøj, J. S., Hung, L., Kwon, H.-K., Moore, P., Schweigert, D., Severson, K. A., Suram, S., Yang, Z., Braatz, R. D., & Storey, B. D. (2020). BEEP: A Python library for Battery Evaluation and Early Prediction. *SoftwareX*, *11*, 100506. https://doi.org/10.1016/j.softx.2020.100506

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Redondo-Iglesias, E., Hassini, M., Venet, P., & Pelissier, S. (2023). DATTES: Data analysis tools for tests on energy storage. *SoftwareX*, *24*, 101584. https://doi.org/10.1016/j.softx.2023.101584

Rudiger, P. (2024). *Holoviz/hvplot*. HoloViz. https://github.com/holoviz/hvplot

Sulzer, V., Marquis, S. G., Timms, R., Robinson, M., & Chapman, S. J. (2021). Python Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, *9*(1). https://doi.org/10.5334/jors.309

The pandas development team. (2020). *Pandas-dev/pandas: pandas* (Version 2.2.2). Zenodo. https://doi.org/10.5281/zenodo.3509134

Vink, R., Gooijer, S. de, Beedie, A., Gorelli, M. E., Guo, W., Zundert, J. van, Peters, O., Hulselmans, G., nameexhaustion, Grinstead, C., Marshall, Burghoorn, G., chielP, Turner-Trauring, I., Santamaria, M., Heres, D., Mitchell, L., Magarick, J., ibENPC, … Brannigan, L. (2024). *Pola-rs/polars: Python Polars 1.4.1*. Zenodo. https://doi.org/10.5281/zenodo.13208786

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L. B. da, Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 160018. https://doi.org/10.1038/sdata.2016.18

Wind, J., Ulvestad, A., Abdelhamid, M., & Mæhlen, J. P. (2024). Cellpy – an open-source library for processing and analysis of battery testing data. *Journal of Open Source Software*, *9*(97), 6236. https://doi.org/10.21105/joss.06236