User-Friendly, Robust Data Analysis Software for Cryogenic Molecular Beam Data (CryoDAS)

Emily Pan
Mentors: Nick Hutzler and Arian Jadbabaie

Abstract

Currently, there are no dedicated data analysis systems designed for the cryogenic molecular beam data source, and existing analysis lacks modularity and a user interface. CryoDAS is an object-oriented, robust system that allows users to apply computations and plot data interactively. These computations include sorting and averaging, filtering, and integrating data, which can be applied to the data in any sequence. CryoDAS uses widgets to improve the user experience and plot readability, including sliders to change plotting bounds, checkboxes to indicate the computations done on the data, and buttons to save plots. CryoDAS also speeds reading and writing saved data by saving the data as objects that can be read directly. CryoDAS is a resource that any group analyzing cryogenic molecular beam data can use.

CryoDAS: Motivations

A cryogenic molecular beam is a stream of very cold gaseous molecules. While the creation and observation of cryogenic beams of atoms is a relatively well-studied area, we choose to create beams of polyatomic atoms as a way to better understand charge conjugation parity (CP) violation. Both the standard model of particle physics and the theory of relativity fail to explain CP violation, so we can leverage the properties of ultracold, polyatomic molecules as a platform to seek new physics or place bounds on existing physics (*PolyEDM Electron EDM Search*, n.d.).

The study of cryogenic molecular beams is a burgeoning field. Importantly, however, data analysis systems for the cryogenic molecular beam source are crude and underdeveloped. Ultimately, this hinders understanding of the data and slows the progression of research. Thus motivates the development of CryoDAS with 3 primary goals. First, data acquisition, storage, and analysis must be designed with an object-oriented approach. Next, methods must be implemented to carry out both general data pre-processing as well as common computations. Lastly, the data must be visualized and there must be a helpful user interface. Ultimately, CryoDAS succeeded in fulfilling these goals and providing a consistent working framework. Furthermore, CryoDAS is easily adaptable to any lab studying cryogenic molecular beams with similar data acquisition setups. However, there remains room for further work in computational efficiency and aesthetics.

Methods

Cryogenic Molecular Beams

We base CryoDAS around the 4K YbOH beam source as a model for our data. Other ongoing experiments employing cryogenic molecular beams take similar data, so by designing our data analysis around the 4K system, we ensure that our analysis will be robust enough to enable compatibility with other beam data.

Cryogenic YbOH beams are generated through a process known as cryogenic buffer gas cooling. As can be seen in Figure 1, in a copper cell, a solid target is laser ablated and generates gaseous YbOH, which is rapidly cooled by surrounding helium gas to around 4K. Then, the YbOH molecules are allowed to leave the cell through a small opening, thus creating the molecular beam (Jadbabaie et al., 2020). Once a stream of YbOH molecules are generated, laser probes target the YbOH molecules to understand molecular geometry and behavior. These lasers take absorption and fluorescence data and form the data set that CryoDAS analyzes.

Object-Oriented Analysis

When designing objects to hold the data, it was best to reflect the actual form of the raw data collected. In its raw form, the data from each experiment is packaged as a folder of text files. Each text file represents a scan and contains a metadata header that describes the parameters of the experiment and a collection of data values indexed by time. Since many lasers are probing the beam at the same time, there are many “channels” of time series data within each file. In order to incorporate this nested design, we created a TimeSeries object to hold each “channel” of time series data, and a FileData object to hold a collection of TimeSeries objects. Since the TimeSeries objects are often isolated from the FileData object for computation, we chose to associate the header metadata with the individual TimeSeries objects rather than with the FileData object. While this increases storage space and slightly strays from the true form of the raw data, it allows for easier computation as well as code usability.

For the ScanData object, we chose to sacrifice sticking to the form of the raw data in exchange for a similar but slightly different array-like format. The ScanData object held an array of TimeSeries objects, where the first index of the array indicates which channel the TimeSeries came from and the second index indicates the file number. We chose to implement ScanData objects this way because we often analyze the values from a single channel over multiple files, and the relationship between the same channel over multiple files was much more important than the relationship between channels within a single file. Thus, we felt that the benefits outweighed any potential confusion that the new form might bring. In fact, good code documentation should alleviate these problems. In order to encapsulate the entire experiment folder in an object, the AllScansData object returns to the nesting form by simply holding many ScanData objects.

Data Processing and Computation

When approaching data processing and computation, we had 3 primary objectives: to improve modularity, implement basic pre-processing on all data, and allow enough robustness to allow many different independent variables to be studied.

The first solution to improving modularity was to bypass string parsing when possible. The Python pickle library (*Pickle — Python Object Serialization — Python 3.9.7 Documentation*, n.d.) was used to save and load class objects so that at any point in data analysis, computations can be directly applied without the need to format the data into a raw state and go through the entire parsing process again. We also changed the data acquisition code in LabView so that the file headers include directly executable Python code. This allows changes in the metadata to be dynamic because without it, directly editing the parsing code would be required to accommodate any changes. The addition of more parameters occurs frequently, and changing it at the data acquisition level is much preferred to the analysis level.

There are some basic processes that all data must undergo. Namely, all data is filtered, and any linear drift is removed. A lowpass Butterworth filter was selected. When removing linear drift, it must be noted that since both fluorescence and absorption data are taken, the log values of the absorption data should actually be used. This difference was noted using an absorption "tag," which indicates that the data must be treated differently.

The common computations that were useful to implement are integration and averaging. One feature that was important to these methods were that the input object had to match the output object. In existing analysis, the computations had to be applied to the data in a rigid, prescribed order. However, in our implementation, the data can be fed into the computations in any order and allow results to be viewed at any step.

In order to accommodate the many independent variables studied, we had to use generalized get and sort parameter functions. This way, any independent variable can be used to reformat the data into a sorted array, which then can generate a new object with sorted data.

Data Visualization and User Interface

In order to visualize data, we employ the matplotlib library (*Matplotlib 3.4.3 Documentation*, n.d.) to quickly generate plots of the data. Specialized methods for each data object are implemented so that objects can directly be plotted.

For a user interface, we use the ipywidgets library (*Widget List — Jupyter Widgets 8.0.0a5 Documentation*, n.d.) to add interactive widgets. There exists a dropdown to allow the user to choose and display independent variables, and to show the plot of a selected channel. A slider helps the user set plotting and computation bounds. In current analysis, bounds were generated through a process of estimating and adjusting. However, the slider allows the bounds to just be dragged to the right place, cutting out the hassle of estimating the bounds. Also included on the plot itself are buttons that allow the user to return to past plot views, drag the plot around within the axes, resize the axes, save the plot locally, and return to original zoom settings.

In the future, full documentation of CryoDAS will be created so that anyone using cryogenic molecular beam data can quickly understand how to use and modify CryoDAS to their specifications. More aesthetic features, such as color pickers and more interface options can be included. Lastly, we seek to further improve performance such as multiprocessing compatibility (*Multithreading vs Multiprocessing in Python*, n.d.).

References

Multithreading vs Multiprocessing in Python (n.d.). Retrieved August 16, 2021, from https://levelup.gitconnected.com/diy-multithreading-vs-multiprocessing-in-python-fb93698ca7f3

Jadbabaie, A., Pilgram, N. H., Klos, J., Kotochigova, S., & Hutzler, N. R. (2020). Enhanced molecular yield from a cryogenic buffer gas beam source via excited state chemistry. *New Journal of Physics*. https://doi.org/10.1088/1367-2630/ab6eae

Matplotlib 3.4.3 documentation. (n.d.). Retrieved September 13, 2021, from https://matplotlib.org/stable/contents.html

Pickle—Python object serialization—Python 3.9.7 documentation. (n.d.). Retrieved September 13, 2021, from https://docs.python.org/3/library/pickle.html

PolyEDM Electron EDM Search. (n.d.). Retrieved September 11, 2021, from https://sites.google.com/site/hutzlerlab/research/polyedm-electron-edm-search

Widget List—Jupyter Widgets 8.0.0a5 documentation. (n.d.). Retrieved August 16, 2021, from https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html

Acknowledgements

I would like to acknowledge my mentor Nick Hutzler, my co-mentor Arian, as well as the entire Hutzler Group for continuous support throughout the project. Furthermore, I'd like to thank Caltech SFP for providing me with the SURF opportunity.