Calibration Program Documentation

Tristan Anderson

April 28, 2019

Contents

1 A Note

This set of classes are to be executed with Python 3.6.7 **NOT** in python's interactive mode. I am currently implementing a GUI as of 03/27/2019, but there is no guarantee of its completion. Two classes were made to work in tandem here, the "*slifercal*" class inherits the "*thermistor_profile*" class, the former deals with the overall data-parsing and graphing, while the latter deals with the calibrations. I don't really have formal training in working with large data sets. I took an introductory level python programming class in 2016 - proceed formally.

2 slifercal

2.1 Foreword

It is imperative, **David,** (or LabVIEW manager) to make sure the columns of the data file passed to the *slifercal* class is in the proper format - so labview should be outputting a datafile in the format of something along the lines of:

Time (seconds or datetime*)	**AB.M1**	**...**	**CCS.F1**	**...**	**Comments**
3635332758.31034	163.412	**...**	2032.2	...	**ipsum lorem**
3635332772.31121	163.393	**...**	2035.2	...	**ipsum lorem bacon**

The program only analyzes the data underneath the headers that have "Time", "AB.", "CCS.", and "Comments" in them.

2.1.1 Initialization

```
instance = slifercal(file_location=None):
```

The objective of this class was to create a persistence based thermistor calibration program using pickle for IO. The user can initialize the class with a data file location: *file_location* but providing a path to a data file is not necessary. All the class does when initialized is take whatever location provided, and store it as a self variable.

2.2 Prebuilt Methods

2.2.1 Loading Previous Data

```
slifercal.load_data(data_location=None):
```

This method searches for the most recent "keeper_data_original_YYYY_MM_DD_HHMMSS.pk" where Y, M, D, H, M, S is the year, month, day, hour, minute, and second respectively specifying the file's time of generation. One does not have to provide a file name or location so long as the name of the file remains unchanged. If one wishes to use older pickled parsed data - so be it. Simply call the method with the appropriate file location & or name. The GUI (If implemented should should allow you to select

2.2.1.1 Usage

As the parent-title of the subsection above mentions, this method is used to load previously parsed (analyzed) data into the instance of a class.

2.2.2 Analyze Data

```
slifercal.find_stable_regions(rangeshift=1, n_best=10)
```

This will read the data from the file location provided during the instance's initialization. If no file was provided, it will assume the data file is present in the directory that the class was initialized from. It will take several minutes to calculate the calibration points depending on the slice size, step size, and data-file size. The program will periodically notify the user the program's whereabouts in the parsing process. The program prints to the console every time it parses a quarter of a column. Typically in the form of:

"n" of x Ranges averaged.

This method works by using Pandas to slice data from one column at a time. The program then calculates the average value within that slice of data, and calculates its standard deviation. It skips over any slice of data that contains a zero, or NaN values. This information, including the range that the slice originated from is saved, and is written to a dictionary. This dictionary of standard deviations, averages, and ranges is then written to a file, and analyzed. One of the private methods then write the dictionary with averages and ranges to a binary pickle file using cpickle.

2.2.3 General Plotting

```
slifercal.plot_calibration_candidates(n_best=10, dpi_val=150, plotwidth=1000,
    plot_logbook=False, data_record=True):
```

Once the data has been analyzed and the top n most stable regions of data have been saved to that pickled dictionary, the program then garbage-collects all non-critical variables to reduce memory usage. Once the garbage is collected, it initializes the plotting process[†] and begins to plot the n most stable regions.

2.2.4 Analysis Suite

```
slifercal.analysis_suite(rangeshift=1, n_best=10, range_length=None,
    dpi_val=150, logbook=False):
```

This pre-built method combines two methods from sections General Plotting and Analyze Data

2.2.4.1 range_shift

Modifies how the slices of data are selected from the data file. The default is 1, which shifts the range by 1 in the Analysis Suite section above.

[†]Which should probably be turned into its own class due to object format

2.2.4.2 range_length

Specifies the size of the range slice that the program takes as it is parsing the original data file.

2.2.5 Graphing

```
slifercal.plot_calibration_candidates(n_best=10, dpi_val=150, plotwidth=1000,
    plot_logbook=False, data_record=True)
```

It takes the n most stable‡ regions of data that were stored in the "keeper data" dictionary, and plots the sections of data with comments that correspond to the same slice of the data.

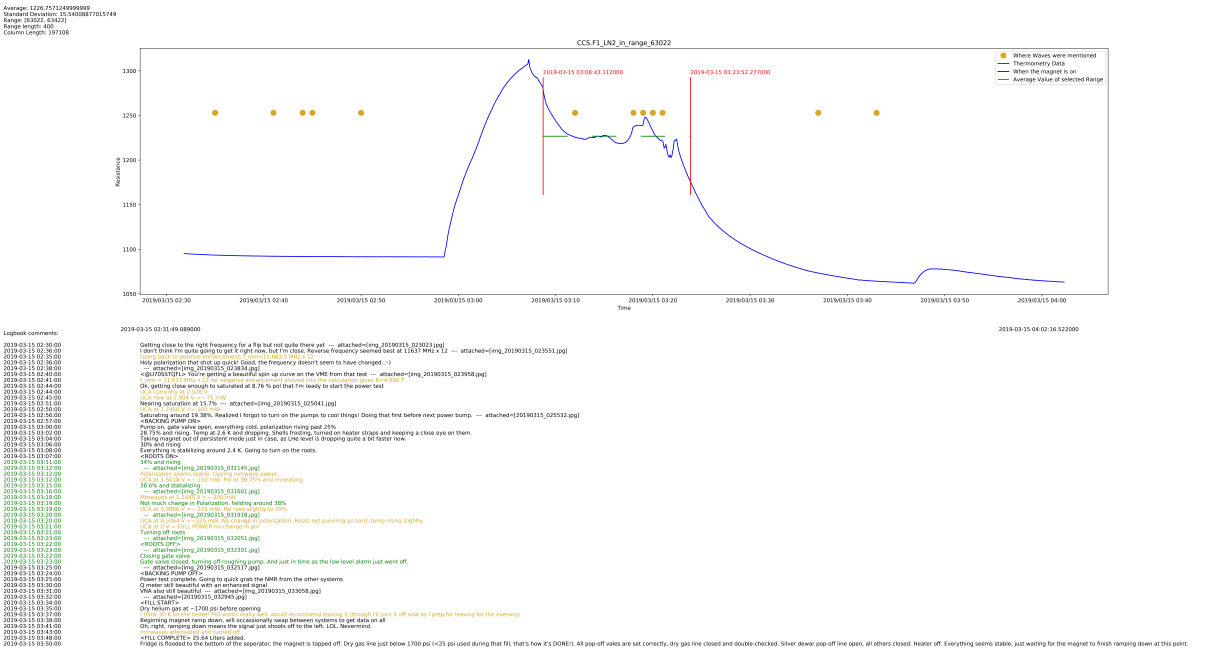

Figure 1: Equipped with a legend, the graph displays lots of useful information in the form of highlighted text, and color-coded markings. On the upper left of the figure, range information is written, below the actual graph are comment entries from the logbook itself. On my laptop, the pdf looks awful on the page that this is printed. If you zoom to 200% in the PDF, you should be able to read the comments, and see the graph a little more clearly.

2.2.5.1 n_best

If no "thermistor calibration points" dictionary was found during the calling of this method, the method will try to find the most recently saved pickling of the parsed data. The program

‡By standard deviation

will then take the n most stable regions per thermistor per temperature range and plot them.

2.2.5.2 dpi_val

Specifies the DPI resolution for the plot. By default the plot is 32"x18". I encourage you to use a DPI of at least 150 so you can read the comments on the bottom half of the graph.

2.2.5.3 plotwidth

The naming of this argument is misleading This argument determines the length of **EACH** "wing" on both sides of the region of data that the program has determined as "flat". Expect a total range width of Plot Width $\approx 2 * plotwidth + range\_length$

2.2.6 Why the hell is it taking so long

Pandas is a powerful library because it's backbone is Numpy arrays. They're high performance and written at the C level; *however*, due to the quantity of data being parsed, this pandas is fairly slow at data slicing. The second slowest executed command in the *slifercal* and *thermistor_profile* classes are executed on the order of 20 ms. The longest command in any loop that is not a data slice takes no more that 5 ms to execute. I promise I have tried to vectorize as much as I possibly could to make this faster. I blame pandas. I also don't want to rewrite this in a different language (yet).

3 Thermistor Profile

3.1 Foreword

This class is a data-structure that handles the negative temperature coefficient thermistor calibrations that are installed on the superconducting solenoid and helium evaporation refrigerator in the Dynamic Nuclear Polarization system. The ones that are tracked by this program have the functional form of:

$$T_{Kelvin} = a + b * \exp\left(\frac{1000 * c}{R_{\Omega}}\right) \tag{1}$$

Other thermistors like Cernox, pre-calibrated Cernox, and pre-calibrated carbon ceramic thermistors are not managed by this class, therefore they will be ignored if the column labels contain exactly: "AB." and "CCS." The *curve_coefficent__data.csv* should never be manually edited, and should remain in the following format:

Name	**AB.M1**	**...**	**CCS.F11**	**...**	**Other pre-calibrated thermistors**
a	10.6901	**...**	-2.660	**...**	...
...		**...**	...	**...**	...
c	-28.279...	**...**	4.735	**...**	...
RT	142.001	**...**	917.6	**...**	...
...	...	**...**	...	**...**	...
Methanol		**...**	1048.03	**...**	...

It is a tab delimited file. If you need to open it for any reason, do so - but again **don't manually edit this file.**

3.2 Initalization

```
from thermistor_profile import thermistor_profile
thermistors = {thermistor_name : thermistor_profile(thermistor_name) for
    thermistor_name in list_of_thermistors}
```

This class is meant to be inherited by the slifercal class. *Each instance of this class is, in itself a representation of the calibration profile of a particular NTC-Thermistor.* Each instance is a column of the *curve_coefficent__data.csv* file.

3.3 Prebuilt Methods

3.4 Back end / "Private" methods

3.4.1 None