frankenplot 2.0: An Improved Quick 'n' Dirty MRCAT XRF Plotter

Jason Petsod
Physics Division, BCPS Department, Illinois Institute of Technology, Chicago, IL 60616, USA
(Dated: 9 May 2010)

frankenplot is a Python application for plotting MRCAT x-ray fluorescence mapping data. Its appeal stems from its simplicity and ease-of-use: users do not have to manually construct and format the plot as frankenplot is tailored for visualising such fluorescence data. However, its functionality is limited. During this Spring 2010 semester, a more powerful and flexible version of frankenplot was designed and implemented based on user feedback and typical use cases.

Conventions

Monospaced text preceded by a dollar sign ($) indicates a particular column in the data file; for instance, $corr_roi3_2 signifies the "corr_roi3_2" column. Monospaced text preceded by a percent sign (%) indicates a group (see below); as example, %corr_roi_1 refers to the group named "corr_roi_1".

I. INTRODUCTION

frankenplot is an application for quickly and easily plotting MRCAT x-ray fluorescence mapping data. Since frankenplot understands and is tailored for MRCAT data formats, users can visualise such data in a useful manner using a single command without having to move data files around or manually format a plot.

II. MOTIVATION

frankenplot was first written in December 2006 by Ken McIvor, but prior to Spring 2010, its functionality was limited: it could only plot a region of interest (ROI) from a data file, save the plot, or print the plot. Further, customisation of the plot was limited to toggling plot normalisation by I_0, changing the colormap of the plot, and changing the columns used. In order to implement custom functionality such as plotting different functions or specific columns, users had to edit the data file or edit the application itself. In order to address this lack of flexibility, frankenplot 2.0 was born, aiming to present to the user a much more powerful plotting application for XAFS mapping data.

III. DESIGN PLAN

For the new version of frankenplot, three overall modes of operation were proposed which would align with typical use cases: Fluorescence Mode, Transmission Mode, and Colormap Mode. In Fluorescence Mode, the user would be able to plot the fluorescence data from the file: ROIs or specific channels; in Transmission Mode, either the sample or reference transmission would be plotted; and in Colormap Mode, the user would be able to plot any column of the data file or an arbitrary expression comprised of mathematical operations on any number of columns.

To implement these functions in an maintainable manner, an underlying mode-agnostic expression system would need to be implemented; that is, the GUI frontends for these three modes would construct an appropriate expression to be plotted and this expression would then be processed by the plotting routine that has no knowledge of the current mode of the application. By emphasising this separation between presentation and data, it makes it easier to create arbitrarily complicated plots of the data.

Users would also be able to create "groups" comprised of specific columns in the data file and other groups. These groups would simplify user interaction by providing a means for easily performing mathematical operations in aggregate on a number of columns in the data file. Groups would persist throughout the session so the user could continue to use these groups. Additionally, the notion of groups abstracts away columns in the data file if the user does not need that level of granularity. For instance, the %corr_roi_1 group could contain the columns for all of the channels in the first corrected ROI.

In addition to the three modes and the underlying abstractions, various user interface improvements were identified. Since in Colormap Mode, users have full flexibility in constructing arbitrary expressions, an intuitive graphical "Expression Builder" would be desirable to help users construct complicated expressions that are valid and plottable without them having to learn the expression syntax themselves.

Minor enhancements identified were the ability to have multiple windows open at a time, each showing a different plot and/or different data file; having frankenplot remember the settings for each data file such that when a user re-opens a data file, his previous plot parameters for that file are restored; functionality to edit the plot title; and the ability to set the number of significant figures displayed.

A. Fluorescence Mode

In Fluorescence Mode, the user would be presented with options to plot the fluorescence data for a given ROI, a specific channel from an ROI, or some user-defined parameters describing I_0 and I_f: these modes being "sum mode", "single-channel mode", and "user-defined mode" respectively.

In single-channel mode, only the contents of a single channel from the selected ROI are plotted. For instance, if the user wants to plot channel 8 from corrected ROI 2 of the data file, `frankenplot` plots the data in the column `$corr_roi8_2`. Users would be able to choose the desired channel from a drop-down menu or switch between channels with back/forward buttons. Specific channels could be enabled or disabled, where disabling a channel excludes it from being summed over in sum mode (see below). Disabling channels would be useful, for example, in the case where the detector reported erroneous results for only particular channels.

When `frankenplot` is in sum mode, each data point is calculated by summing over all of the enabled channels in a specific ROI. For instance, if a data file contains the channels $0, \ldots, 15$ and ROI 3 is selected, the plot will be formed by taking the sum `$corr_roi0_3` + `$corr_roi1_3`$+\cdots+$`$corr_roi15_3` at each data point. If any channels were disabled in single-channel mode (see above), then the associated columns would be excluded from the sum. This sum would be an example of an expression passed to the plotting routine.

Two options would control the behavior of the sum and single-channel modes, with both enabled by default. The user can toggle the using corrected/uncorrected ROIs, using the `$corr_roi*` columns versus the `$roi*` columns; and the user can toggle normalisation, where the expressions are divided by the incident intensity I_0 (`$Io` by default) before being plotted.

Finally, in user-defined mode, the user would be able to completely control the behavior of the fluorescence plot by specifying arbitrary columns, groups, or expressions for both I_0 and I_f. This mode would utilise the Expression Builder to present the user with a powerful but easy-to-use interface for constructing a custom fluorescence data plot.

B. Transmission Mode

In transmission mode, the user can visualise the transmission of the beam through the sample, transmission of the beam through a reference sample, or some user-defined expression describing the transmission.

When plotting sample or reference transmission, the XAFS transmission equation,

$$T = \ln \frac{I_0}{I_t}, \tag{1}$$

is used to determine the plot. For sample transmission, I_0 = `$Io` and I_t = `$It`; for plotting the reference transmission, I_0 = `$It` and I_t = `$Iref`. Additionally, the user would be able to specify an expression for I_t and I_0, or alternatively, construct an arbitrary expression for the transmission altogether.

C. Colormap Mode

In Colormap Mode, the user has full flexibility to plot an arbitrary colormap of data columns, groups, and mathematical operations on such. The Expression Builder would also be available to facilitate construction of complicated expressions.

IV. RESULTS

`frankenplot` is written in Python using the wxPython GUI toolkit. As its primary users are expected to be running Debian Lenny or Squeeze, the code has only been tested using the standard versions of libraries on Debian Lenny (see Appendix B).

A basic distutils `setup.py` build and installation script was written and tested locally. However, no Debian package has yet been created.

The expression system was implemented in an object-oriented manner, where expressions are represented by objects. When the user requests a plot, the GUI callbacks specific to the mode execute and form an expression based on the plot parameters. For instance, in Fluorescence/Single-Channel Mode, when the user switches channels, a new `ChannelExpression` object is created with the `roi` and `channel` attributes set appropriately. This object inherits from the `FluorExpression` class which, in turn, inherits from the abstract `Expression` class. The data for the plot is generated by calling the `xdp.Data.evaluate()` method on some string, so when the time comes to create the actual plot, the string representation of the `Expression` objects needs to be computed. Thus, we require `Expression.__str__()` to be implemented by classes inheriting from it. In our example, `ChannelExpression.__str__()` creates an expression of the form `$corr_roi`<*channel*>`_`<*roi*>. As another example, `ROIExpression.__str__()` (representing a plot of an ROI in Fluorescence/Sum Mode) generates an expression in the form `%corr_roi_`<*roi*> which is then expanded into its constituent data columns before plotting.

The majority of the desired functionality for the Fluorescence and Transmission Modes was accomplished this semester. As shown in Figure 2 and Figure 3, the sum and single-channel modes were implemented, allowing users to view individual channels and enable/disable channels for sum mode. Additionally, a separate "Plot Controls" window was created in order to not clutter up the main plotting window. The drop-down menu at the

top of the Plot Controls window serves to both change the operating mode of the program and display the plot parameters specific to the mode.

In Figure 4 and Figure 5, we see the behavior of `frankenplot` in Transmission mode. Users can choose from the predefined Sample Tranmission or Reference Transmission or create their own custom transmission mode plots using the drop down menus or entering an arbitrary expression. However, the Expression Builder was not implemented this semester so at current, users will need to be familiar with the XDP expression syntax to use the most advanced mode.

Finally, in Colormap Mode (Figure 6), the user is currently only presented with a box for an arbitrary expression. Once the Expression Builder interface is created, this mode is expected to become much more useful and functional for users.

V. FUTURE WORK

As discussed, the Expression Builder interface was not constructed during this semester and should be a major priority in the future as "custom" modes in `frankenplot` are unintuitive without its functionality.

Support for having multiple plot windows has also not yet been implemented. For user ease-of-use, the behavior of the Plot Controls inspector in a multiple-window scenario should mimic that of many Mac OS X applications: there is a single "inspector" panel which changes its state to reflect the currently focused or most recently focused plot window. In this way, we would reduce the clutter created by having a separate Plot Controls window for each plot window and present (Mac) users with functionality they are already familiar with.

Additionally, the ability for `frankenplot` to "remember" previous plot settings was not done during the semester. This could be implemented using something akin to Vim's `~/.viminfo` file to store mappings between data file paths and the most recent plot parameters for each mode. Similarly, some sort of `frankenplot` configuration file (e.g., in `~/.frankenplot.conf`) would be useful for users as well.

Finally, direct user studies and solicitation of users' suggestions and feedback was not conducted during this semester and would be recommended before full-scale deployment. And as with any other software project, ongoing bug-hunting and documentation efforts are needed.

Acknowledgements

I would like to thank Ken McIvor and Professor Carlo Segre for their guidance and insight during this project.

Appendix A: Source Code

Source code for `frankenplot` is available on Gitorious at `http://gitorious.org/frankenplot/frankenplot`.

Appendix B: Dependencies

The following were the libraries and versions used to test `frankenplot` (versions in Debian Lenny as of this writing). Debian package names are included in parentheses.

- Python 2.5 (`python`)
- wxPython 2.6.3.2.2 (`python-wxgtk2.6`)
- matplotlib 0.98.1 (`python-matplotlib`)
- wxmpl 1.3.0 (`python-wxmpl`)
- xdp 1.5.10 (`python-xdp`)

Appendix C: Screenshots

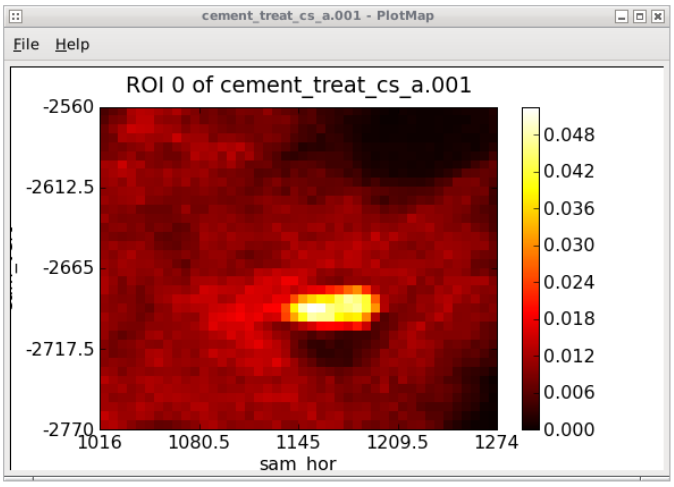

FIG. 1: `frankenplot` prior to the Spring 2010 semester

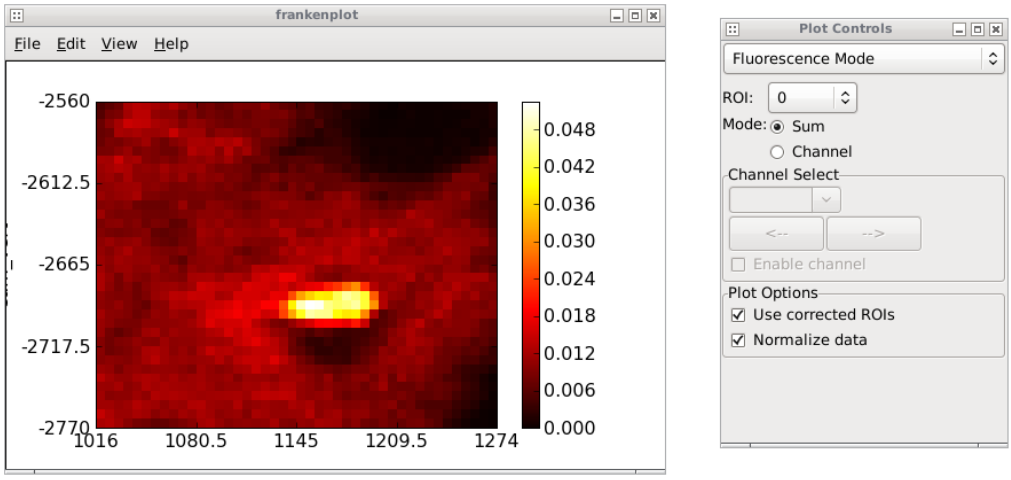

FIG. 2: Fluorescence/Sum Mode plotting ROI 0

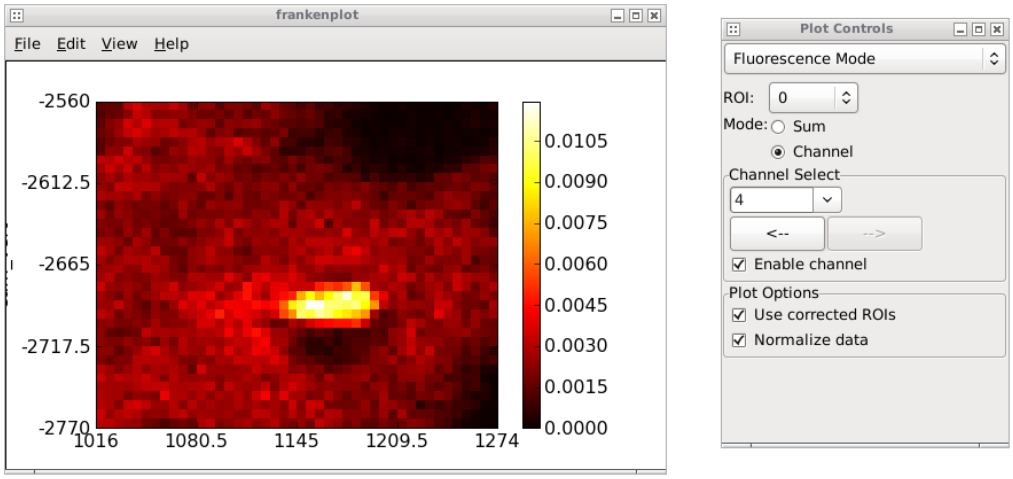

FIG. 3: Fluorescence/Single-Channel Mode plotting ROI 0, Channel 4

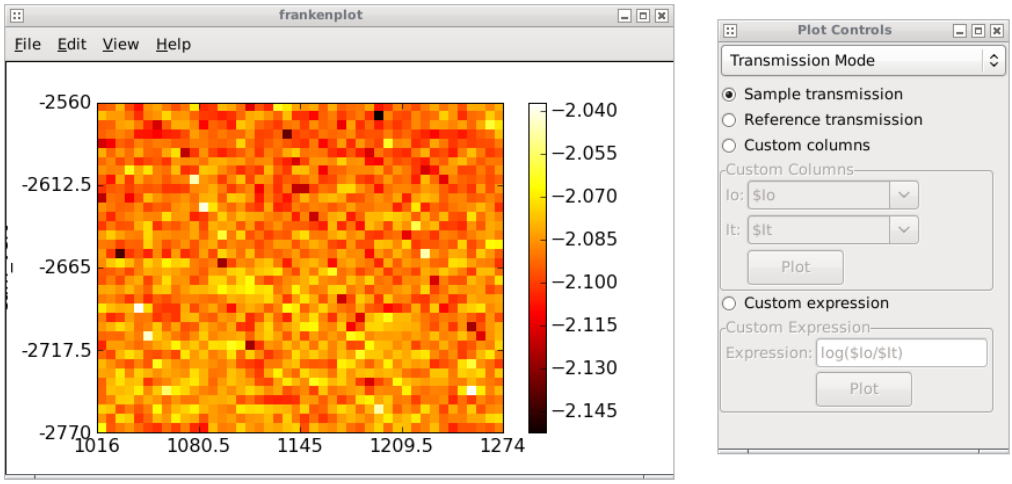

FIG. 4: Transmission/Sample Mode

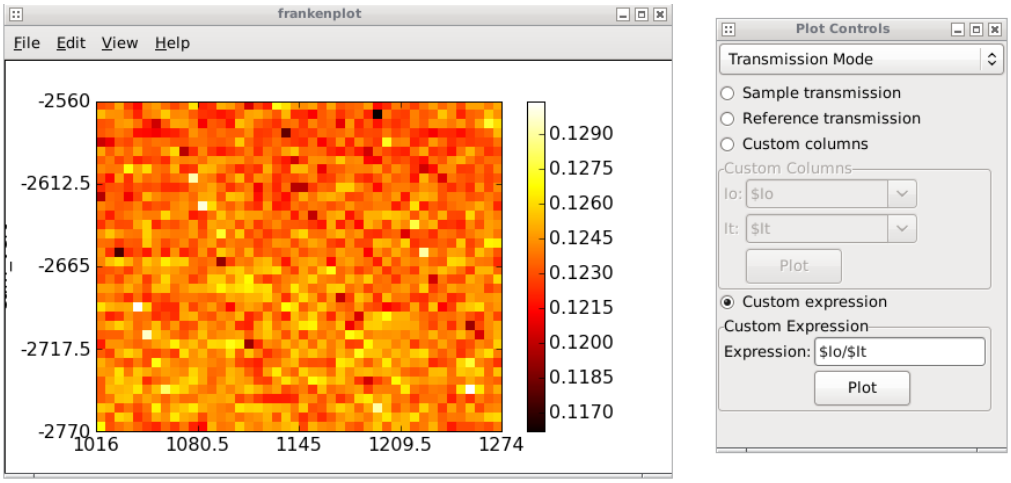

FIG. 5: Transmission/Custom Expression Mode plotting the user-defined expression `$Io/$It`

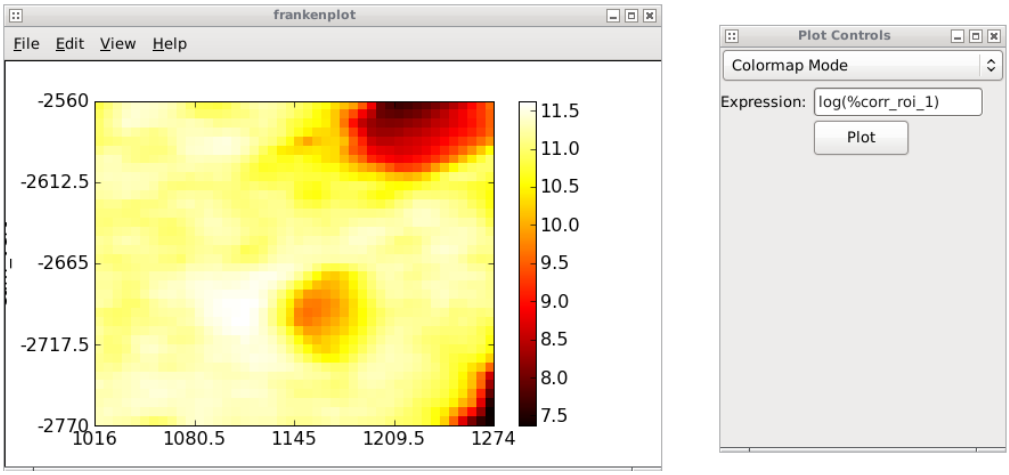

FIG. 6: Colormap Mode plotting the user-defined expression ln(`%corr_roi_1`)