Expresso Analysis Toolbox (EAT) User Guide

Sam Whitehead
email: scw97@cornell.edu

September 18, 2017

1 Introduction

The following is a brief guide for the Yapici lab's Expresso data analysis suite, the Expresso Analysis Toolbox (EAT). Expresso is a novel, automated feeding assay for precisely measuring real-time feeding bouts in *Drosophila*. For more details about the overall use of Expresso, see [Yapici et al., 2016]. This document is concerned with the post-processing of data that is acquired using the Expresso system. The data comes in the form of time series of volume levels in a calibrated glass capillary; the goal of our post-processing with EAT is to identify bouts of feeding within these data sets, and extract them for further analysis.

This software is open source, and available upon request to anyone who would like to use the Expresso system. To cite EAT, please contact us at either scw97@cornell.edu or ny96@cornell.edu (our first publication using this analysis package is forthcoming).

2 Getting set up

This analysis package for Expresso data is implemented in Python, and thus requires a working Python installation to function. The code should be compatible with either Python 2.7 or >=3.4. **NB:** if you do not already have Python installed on your machine, we recommend the Anaconda installation of Python, which by default includes many of the package dependencies required to run the analysis code. Download and installation instructions for Anaconda can be found at https://docs.anaconda.com/anaconda/.

2.1 Dependencies

The following are Python packages that need to be installed before running the Expresso analysis code and that are not included in the Anaconda distribution:

- `changepy`: a Python implementation of the PELT change point detection algorithm [Killick et al., 2011] written by Rui Gil. The code can be installed via pip (i.e. in the terminal, type `pip install changepy`). Details about the code repository can be found on Rui Gil's GitHub (https://github.com/ruipgil/changepy.
- `PyWavelets`: a package for wavelet analysis in Python. This can be installed via conda (`conda install pywavelets`) or pip (`pip install pywavelets`). More info at https://pywavelets.readthedocs.io/

The following are other Python packages that are necessary for the code, that are included in the Anaconda installation, but are somewhat non-standard, and thus may require user installation.

- `h5py`: a package for dealing with .hdf5 files, which is the default output for data from the Expresso system. Can be installed via pip. More information and installation instructions at http://www.h5py.org/
- `statsmodels`: a general package for statistical models in Python. Can be installed via pip. Documentation and additional installation instructions at http://www.statsmodels.org/stable/index.html
- `openpyxl`: a Python library for reading and writing Excel 2010 xlsx/xlsm/xltx/xltm files. Can be installed via pip. Documentation and additional installation instructions at https://openpyxl.readthedocs.io/en/default/

The remaining dependencies are the standard bunch: `numpy`, `scipy`, `matplotlib`, `Tkinter`, etc. For a full list of dependency installation instructions that can be copied and pasted into the terminal, see Appendix A.

3 Example usage walkthrough

Once you have a working installation of Python with the proper dependencies, you are ready to run the analysis GUI! To start the GUI, run the function `expresso_gui_rough_mk2.py`. This can be done either via command prompt or interactive development environment (IDE), e.g. Spyder. The rest of this section will focus on using the GUI.

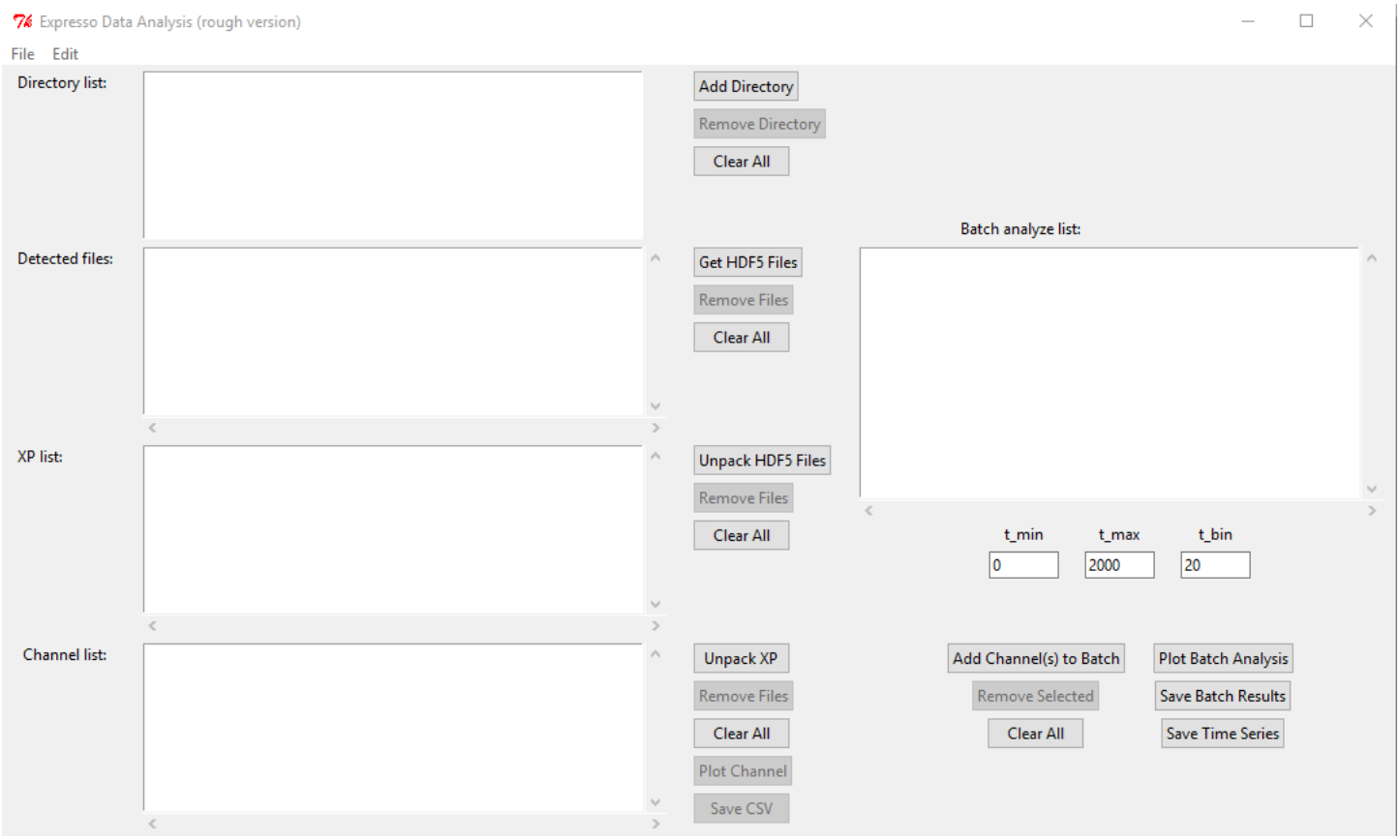

Figure 1: GUI Window. The window that opens upon running `expresso_gui_rough_mk2.py`.

3.1 Loading data

Before getting into the GUI usage, a quick note on the data structure output by the Expresso system. Feeding data is simultaneously collected from Expresso channel banks, each of which hold up to five feeding capillaries. In the setup used by the Yapici Lab, data is collected from two banks, meaning that there are ten total data channels being recorded at a time. The output of this data collection is a .hdf5 file, whose groups correspond to the bank identity (typically labeled *'XP02'* or *'XP04'*), and each of these groups contains five data sets, corresponding to the data from the bank's five capillaries, which are labeled *'channel_1'* through *'channel_5'*. In addition, each .hdf5 file has a field labeled *'sample_t'*, which is the clock for the system (i.e. it gives time at which each data point is collected). For an illustration of this structure, see Appendix B.

The first function of the GUI that we'll go over is navigating this data structure. The four boxes on the leftmost side of the GUI window (Figure 1) correspond to the four levels of data organization depicted in the above directory tree structure. The labels for these four boxes are found to the left, and the buttons used to perform operations on the box elements are found to the right.

Step 1: Specifying a data directory. To begin, we select a folder (or set of folders) that contain .hdf5 files from Expresso. Press the Add Directory button, which will open a standard file navigator window for your OS. Navigate to and select a folder containing Expresso data files, and click OK. This can be done multiple times to populate the *'Directory List'* field. To remove directories from this box, either highlight them individually and click the Remove Directory button, or clear the entire box contents with the Clear All button.

Step 2: Finding .hdf5 files in a directory. After filling in the *'Directory List'* box with the directories you'd like to access, we can then grab all of the .hdf5 files within those directories (Note: this will grab *all* .hdf5 files, so we recommend keeping

Expresso data files in directories that are separate from .hdf5 files that do not pertain to Expresso). To pull out the .hdf5 files from a directory and populate the *'Detected files'* field, **first highlight a single directory in the *'Directory List'* box**. Then press the Get HDF5 Files button. This will fill the *'Detected files'* box with all of the .hdf5 files in the highlighted directory. Again, elements in this field can be removed selectively or all together with the Remove Files and Clear All buttons, respectively. Note that you can fill the *'Detected files'* field with files from multiple directories by repeating the above operation.

Step 3: Unpacking the groups from the .hdf5 files. Next, we can pull the groups (which correspond to the different Expresso sensor banks) from the .hdf5 files. This is accomplished by highlighting one or multiple files in the *'Detected files'* box (Shift + click to highlight many files), and then pressing the Unpack HDF5 Files button. This populates the *'XP list'* field with the different sensor banks, denoted XP02, XP04, etc. At this point, the box items get a little complicated, because they contain the full path of the data file, in addition to the bank label. This added test is to make sure that data files from different experiments can be kept separate when necessary. Again, removing items from this box is accomplished via the Remove Files and Clear All buttons.

Step 4: Unpacking the channels from the hdf5 groups. Continuing on, we can separate out the individual channels from each sensor bank by highlighting one (or multiple) elements of the *'XP list'* and pressing the Unpack XP button. This populates the *'Channel List'* field with the different channels, each corresponding to one capillary from a given experiment. Again, the full path is kept to differentiate data sets. Also, as before, elements of the *'Channel List'* field can be removed with the Remove Files and Clear All buttons.

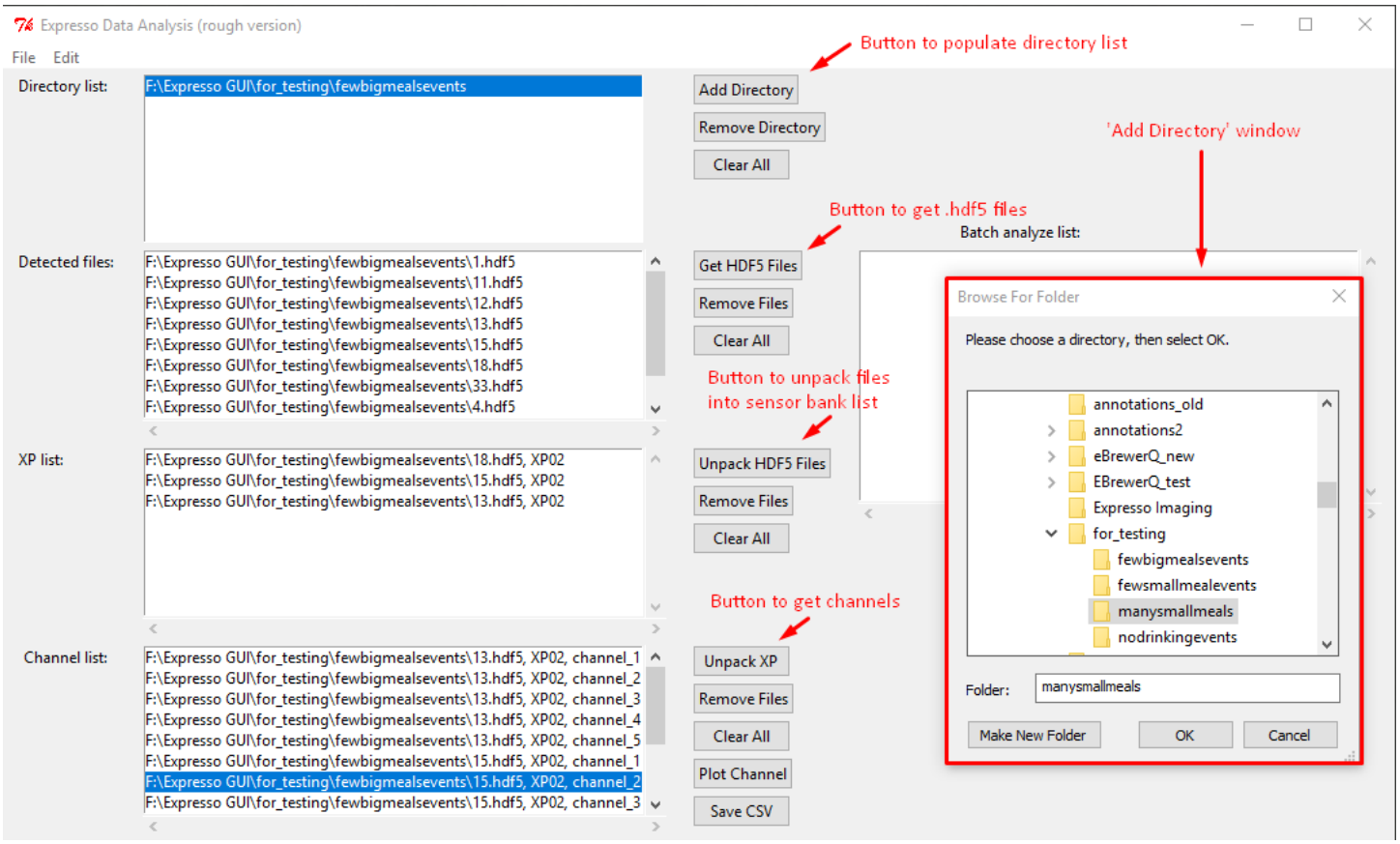

Figure 2: Populated list boxes. Example screenshot of the GUI with the various data fields populated.

Once these steps are completed, we now have access to the data that we'd like to analyze, which we can update at any point. An example is shown in Figure 2. The next sections will address what to then do with the data.

3.2 Plotting single-channel results

Once we have channels populating the *'Channel List'* field, we can now access the data for analysis. As a first pass, we can plot the data from a single channel. This is accomplished by highlighting an individual element in the *'Channel List'* field and clicking the Plot Channel button (immediately to the right of the *'Channel List'* field). Note that this only works for a single channel (capillary).

Pressing the Plot Channel button will open a new window with two plots stacked on top of each other. Both correspond to the same data, but the upper plot is the raw input, while the lower plot is smoothed/denoised. Both plots are time series of capillary volume, with volume in nanoliters on the y-axis and time in seconds on the x-axis. The data is plotted in blue, with detected feeding bouts indicated by red lines and gray vertical bars.

Because this is a standard `pyplot` figure window, the normal tools of zooming, saving, etc. should all be available. This is important to note because saving the individual channel plots requires you to use the saving mechanism in the figure window. In addition to this toolbar, the bottom of the figure window has two sliders, which allow the user to adjust the x-axis range and midpoint. While these can be used whenever, I find that they're really only useful when looking at long time series data; otherwise, the standard zoom tools usually get the job done.

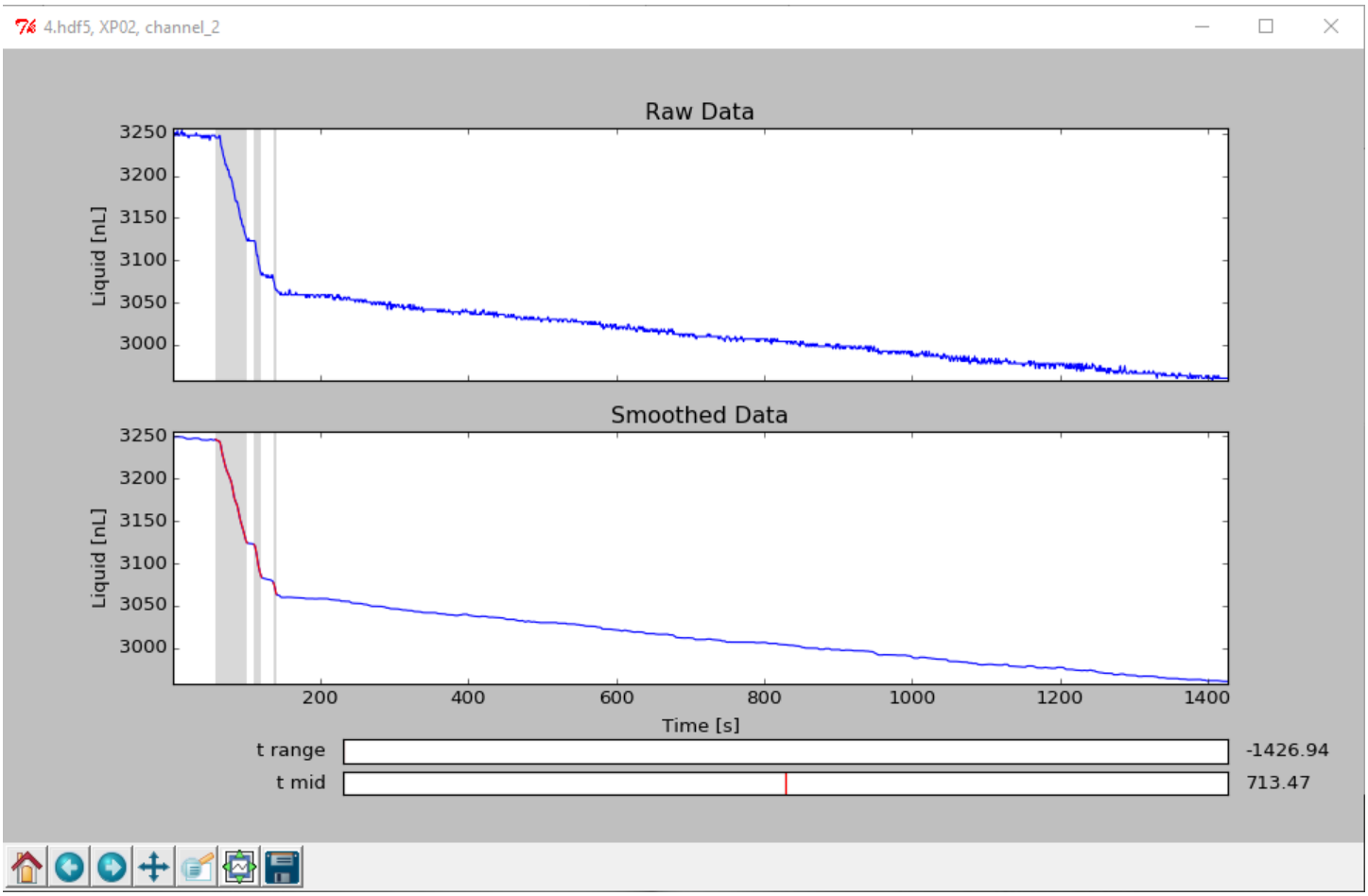

Figure 3: Single data channel plot. Example plots from a single data channel. Upper and lower plots show raw and smoothed data, respectively. The period over which data collected was ∼1425 seconds, and the analysis code detected three feeding bouts, indicated in red. The standard matplotlib tool bar is found in the lower left corner of the window. Above that are the two sliders to adjust range and midpoint of the x-axes.

3.3 Saving single-channel data

In addition to plotting the analysis of a single channel, we might also want to save the results of our analysis for future use. To do so, highlight the channel that you'd like to analyze and save results for in the *'Channel List'* field, and then click the Save CSV button. This will open a file navigator window (native to your OS), wherein you can save the analysis results as

you would any other file. Note that the file name and save location of the .csv file is determined by the user, so while the .csv file will contain information about the original data file, bank number, and channel number within, it will not necessarily have any external marker indicating its origin (long-winded way of saying that the GUI doesn't impose any organizational structure to the saved outputs–that's up to you!).

The Save CSV button saves a list of detected feeding bouts from the selected channel, with each bout described by its order in the times series, its start and end points in both array index and seconds, and the volume of food consumed during the bout in nanoliters. Thus, a typical .csv output will look like the example in Figure 4.

4.hdf5, XP02, channel_2					
Bout Number	Bout Start [idx]	Bout End [idx]	Bout Start [s]	Bout End [s]	Volume [nL]
1	57	100	58.81100082	102.4329987	122.6504306
2	108	119	110.5439987	121.6999969	39.24552219
3	134	139	136.9190063	141.9889984	14.57114335

Figure 4: Single data channel summary. Example saved output from a single analyzed channel with three feeding bouts. Data here is the same as in Figure 3. The upper left corner gives the file name, bank number, and channel number of the analyzed data. In the next row are labels for the data, including Bout Number, Bout Start/End in both array index and real time (seconds), and Bout Volume in nanoliters.

3.4 Batch analysis

In addition to analyzing single-channel data, the analysis GUI allows data from multiple channels to be analyzed in batch. This is accomplished via the batch portion of the GUI, located on the right of the window. To designate data channels for batch analysis, highlight one (or many) items in the *'Channel List'* field, and then click the Add Channel(s) to Batch button. This populates the *'Batch analyze list'* box. Similar to the other list boxes, elements can be removed from the batch analysis list box with both the Remove Files and Clear All buttons.

Similar to the *'Channel list'* field, the *'Batch analyze list'* field gives the option to both save analysis results and plot batch analyses. In this case though, the subject of analysis is whatever is in the *'Batch analyze list'* field, regardless of what is highlighted. Important to these functions are the three input fields immediately below the batch analysis list box, labeled t_min, t_max, and t_bin. Typing numeric values into these three boxes set the minimum and maximum times (in seconds) over which the data sets will be analyzed, as well as the time bin size (also in seconds).

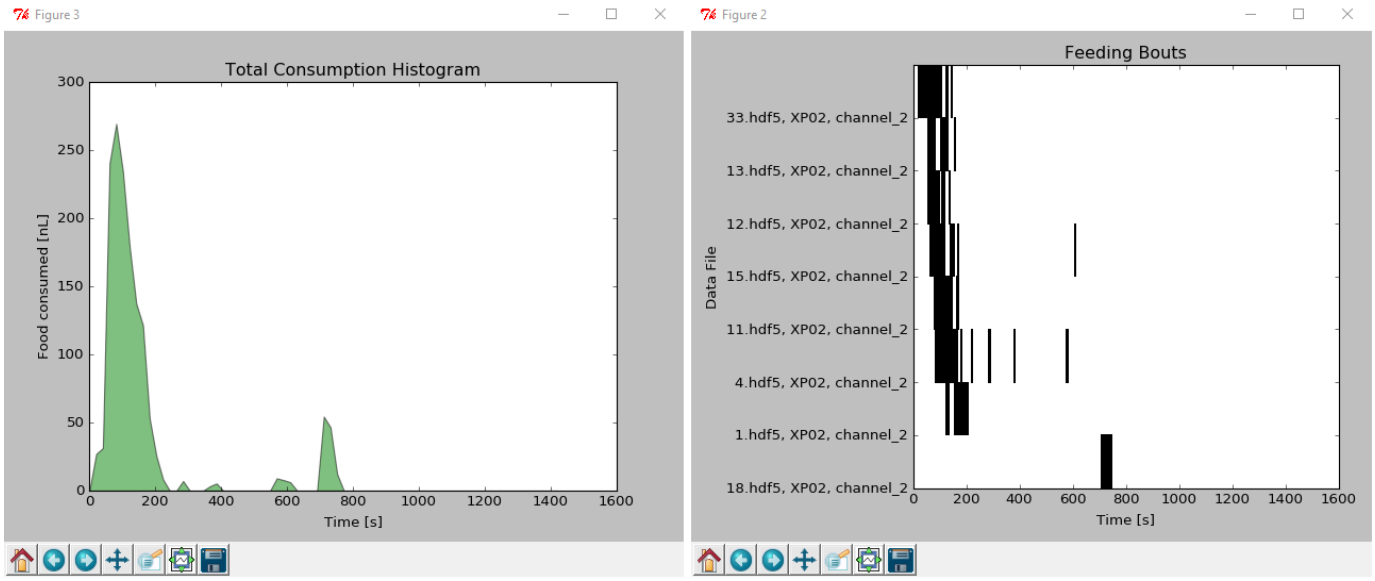

Figure 5: Batch analysis plots. Left: example histogram of good consumption across eight channels from 0 to 1600 seconds, binned in 20 second intervals. Right: same data plotted as a raster plot. Each row is a channel, with the x-axis giving the time. Black regions of the plot correspond to feeding bouts, while white regions are non-feeding times.

Plotting batch results. To plot results of batch analysis, use the Plot Batch Analysis button. This produces two plots: 1) a raster plot of feeding bouts wherein the rows correspond to data channels and the columns correspond to time points, and 2) a histogram that shows food consumption per time bin across data channels. The time range and binning for these plots is determined by the t_min, t_max, and t_bin input fields. The rows of the raster plot are sorted according to the latency to first feeding bout, with the channel with the earliest first meal on top. As with the single channel plots, these can be saved using the toolbox within the figure window. Examples shown in Figure 5.

Saving batch results. Results from batch analysis can be saved in one of two ways: 1) in summary form (as in the single-channel data), which gives information on the detected bouts, their start/stop points, and food consumption, or 2) in time series form. These are done using the Save Batch Results and Save Time Series buttons, respectively. Saving batch results in time series form with the Save Time Series button entails creating a separate .csv file for each data channel in the *'Batch analyze list'* field. Pushing the button opens a file navigator window, which prompts the user to give a directory to save the resultant .csv files in. The files are names in the form <file name>_<bank name>_<channel name>.csv, and contain columns corresponding to index, time (in seconds), raw capillary data (in nanoliters), smoothed capillary data (in nanoliters), and a boolean value indicating whether the fly is feeding or not at that time point.

4 Errors, bugs, issues, and suggestions

If you run into any issues with the analysis code, or have any suggestions for improvement, please do not hesitate to contact Sam Whitehead (scw97@cornell.edu)! We will try to resolve issues in as timely a manner possible.

References

[Killick et al., 2011] Killick, R., Fearnhead, P., and Eckley, I. A. (2011). Optimal detection of changepoints with a linear computational cost. *Journal of the American*.

[Yapici et al., 2016] Yapici, N., Cohn, R., Schusterreiter, C., Ruta, V., and Vosshall, L. B. (2016). A Taste Circuit that Regulates Ingestion by Integrating Food and Hunger Signals. *Cell*, 165(3):715–729.

A Code dependencies

For **Anaconda** users, run each of the following commands in the Anaconda terminal to install the EAT dependencies.

- `pip install changepy`
- `conda install pywavelets`

For **other Python installations** run each of the following command in the Python terminal to install the EAT dependencies.

- `pip install changepy pywavelets h5py statsmodels openpyxl`

To install **all** dependencies, in the event that your Python installation lacks the standard scientific Python libraries, run each of the following commands in the Python terminal to install the EAT dependencies. Note that this requires the ability to use `pip` in your Python terminal; to set up `pip` for installing Python packages, see https://packaging.python.org/tutorials/installing-packages/. However, note also that `pip` does not function very well with Windows, so if you are using a Windows machine and have a Python installation that does not include standard packages like `numpy` or `matplotlib` (to test these, open your Python terminal and type/run `import numpy` to see if you have e.g. `numpy` available) we *strongly* recommend installing a Scientific Python distribution, which can be found at https://www.scipy.org/install.html (we use the Anaconda distribution).

- `python -m pip install -upgrade pip`
- `pip install -user numpy scipy matplotlib ipython jupyter pandas sympy nose`
- `pip install changepy pywavelets h5py statsmodels openpyxl`

All other packages imported by EAT are part of the Python Standard Library.

B Directory structure

Below is a diagram illustrating the typical data directory structure for Expresso output, including the groups and fields within the output .hdf5 files.

```
<data directory>
├── example_data_file_01.hdf5 ..... Example data file output from Expresso
│   ├── sample_t ..... System clock values
│   ├── XP02 ..... Bank 1
│   │   ├── channel_1 ..... Individual data trace
│   │   ├── channel_2
│   │   ├── channel_3
│   │   ├── channel_4
│   │   └── channel_5
│   └── XP04 ..... Bank 2
│       ├── channel_1
│       ├── channel_2
│       └── ...
├── example_data_file_02.hdf5
│   ├── sample_t
│   ├── XP02
│   │   ├── channel_1
│   │   ├── channel_2
│   │   └── ...
│   └── XP04
│       ├── channel_1
│       ├── channel_2
│       └── ...
└── ...
```