LiKa v1.1

Automated Analysis Routines for Voltage Imaging

Danxun Li[1,2], Kaveh Karbasi[1,2], Alison Walker[2], Evan Miller[2]
[1]Undergraduate Research Apprenticeship Program
[2]University of California, Berkeley

June 8, 2015

1 Introduction

The human brain is an amazingly complex organ, governing basic motor movements to higher order thought processes. There exists around 10^{12} synapses in our brain, making us truly unique individuals. While we have seen dramatic advances in our understanding of the human brain, we are yet limited by existing tools in our investigation of the brain circuitry. In order to address the challenge of providing a direct measure of electrical dynamics across multiple neurons, the Miller Lab is developing voltage sensitive fluorescent indicators, which output neuronal activity (changes in membrane potential) via changes in fluorescence intensity. to characterize a new dye, it is essential to understand the exact relationship between fluorescence intensity and voltage. We perform this characterization in HEK293T cells, where changes in membrane potential are induced via a patch pipette and record a timeseries of the fluorescence changes. We seek to automate the image processing procedure by identifying the cell of interest, reliably calculating the ΔF/F as a function of the voltage applied, thus streamlining and standardizing the characterization of a new voltage-sensitive dye. We describe here our automated routine for image analysis.

The image files, the timeseries of fluorescence intensities, are input as tiff stacks, which are represented in MatLab as 3D matrices. We can think of each stack as a timeseries of single frames, where the 3 dimensions are height, width, and number of frames. The previous two parameters together define the object's position in space and the the last describes a position in time. Every element in the matrix contains a value that represents its own fluorescence intensity, which we can then manipulate in order to calculate the ΔF/F as a function of the voltage applied of a region of interest. In a characterization timeseries, voltage is applied to the cell at certain time points, causing a change in the cell's voltage and fluorescence intensity for a short interval before returning to the holding voltage–and baseline fluorescence. This is repeated at regular intervals with a gradient of voltages, during which the cell exhibits different fluorescence changes at these response steps. By traditional methods , one loads these movies into an image processing program such an Image J (Figure 1), select and plot the z-axis (3rd dimension: number of frames) profile of an ROI, then estimate the fluorescence level at each response peak and the baseline and background fluorescence, input these numbers in a data-processing document such as Excel, carry out the necessary computations and finally arrive at a relationship between ΔF/F and applied voltage. This process is not only long, inefficient and irksome, but also contains an element of human error or habit that may inject more than an insignificant variation into the characterization procedure.

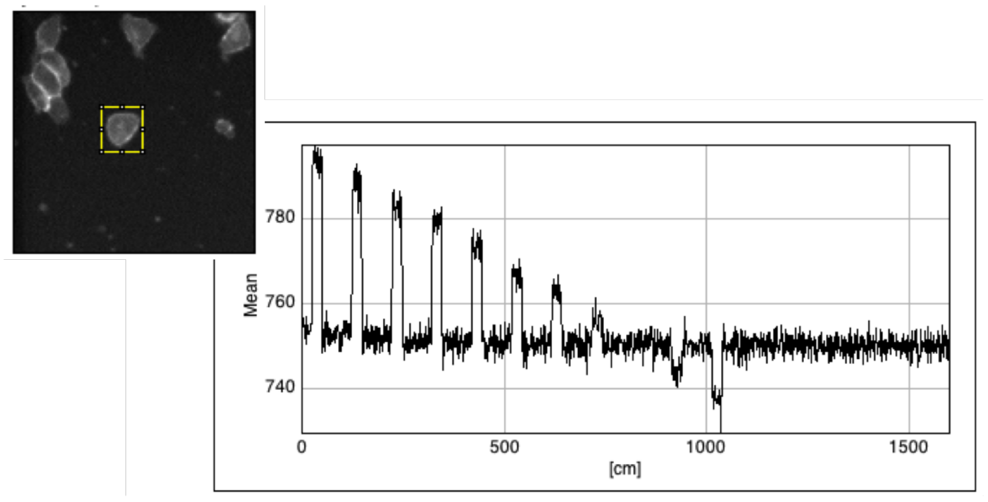

Figure 1: Image processing in the traditional way.

In our analysis routine, we seek to automate the identification of the patched cell, as well as the computation to extract the change in fluorescence during these response steps and correlate with the applied voltage. As these steps are automated, we can streamline and standardize the characterization protocol and reduce variations. We present two master scripts, `mastersc2` and `mastersc3`. The first has the user input a protocol that contains the timepoints that correlate to response steps, whereas the second calculates these timepoints uniquely for each timeseries by detecting rising and falling edges. We commence with constructing the mask, followed by describing the strategies for finally calculating the ΔF/F. These procedures are the same for characterization of the dye in both an experimental range, from -100mV to +100mV, and in a physiological range, from -300mV to +300mV.

2 Constructing the Mask

Stuff here by Kaveh about how `makeCellResponderMask` works.

3 Selecting a Background

After a mask of the cell of interest is generated, the user is prompted to select a background ROI. The region is freehanded rather than defined by a specific shape in order to take into consideration that areas available for background subtraction calculations may not be so permissive for regular shapes.

The stimulation frame (first frame with detectable response) is displayed in grayscale in a new figure, on which the user can draw a desired background region. A mask is created of the object `h` and saved in `binarybgimg`, which returns a binary image that is the same size as the input image, `img`, with pixels recording `1` inside the ROI object `h` and `0` everywhere else.

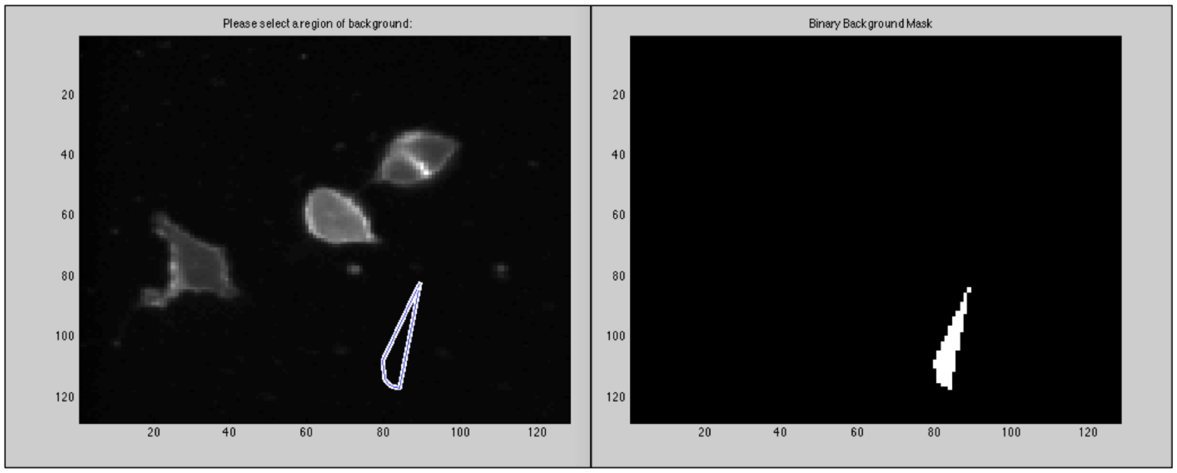

Figure 2: Selecting a background. On the left, `img`. On the right, `binarybgimg`. `h` is the handle given to the region selected. The axes represent position space.

Once `binarybgimg` is generated, it is sent, along with the input tiff file, the mask generated of the cell of interest, and the number of response steps to `epochs`, which identifies the indices that correspond to a change in fluorescence beyond a threshold set by the user.

4 Generating edge indices

In order to calculate average and baseline fluorescence intensities, we begin by generating the indices that correspond to the rising and falling edges of each response step beyond a certain threshold in `epochs` and "clean up" this set of indices via `clusterdxl`.

4.1 Detecting all rising and falling edges

`epochs` takes in the image file (`imgfile`), the mask of the cell of interest (`mask`), the mask of the background ROI (`bgmask`), and the number of response steps (`num`) and outputs:

`ind`: an array of all indices that correspond to a change in fluorescence beyond a threshold
`averageint`: average fluorescence of the cell of interest as a timeseries
`averageintbgsub`: `averageint` with background subtracted

First we run through `imgfile` and apply `mask` to each frame. This is assigned as a new tiff stack matrix to `newmat`. `newmat` contains the timeseries of the fluorescence changes only of the cell of interest; everything outside of the masked cell is black, represented as `0`, and everything within is its original value. (Since $x * 0 = 0$ and $x * 1 = x$.) In order to obtain the background subtracted matrix, we apply `bgmask` to each frame, take the average of each background ROI, and subtract this background average from the corresponding frame in `newmat`. This tiff stack matrix contains only the background subtracted fluorescence time-series of the cell of interest and is assigned to `newmat2`. Once again everything outside the masked cell is `0`. Next we define a threshold, `tit`. The strategy is to string together a matrix of values, `amat`, corresponding to the change in fluorescence intensity from one frame to the next: most elements will be small, whereas large values are expected to correspond to the frames where the cell shows a rapid change in voltage–and fluorescence–profile: at the edges of the response steps. We want to gather all the indices that correspond to a change beyond this previously defined threshold, which may span more than a couple of values and may not necessarily be consecutive. The values that are written into the function are tried and tested, however, guidelines for troubleshooting are included as well.

As before, we run through `newmat` and subtract each frame (`priframe`) from its next frame (`nexframe`), saved as a temporary matrix in `maxchg`. Keep in mind that at this point `maxchg` is a single frame, a 2-dimensional matrix containing the difference in fluorescence intensities only in the cell ROI, as `newmat` is a masked matrix. We then average the nonzero elements in each `maxchg` frame which represents the average fluorescence change of the cell of interest between frames. We save this value in `chg`, and construct a matrix, `amat` of all `chg` values as we run through all the frames, plotted below.

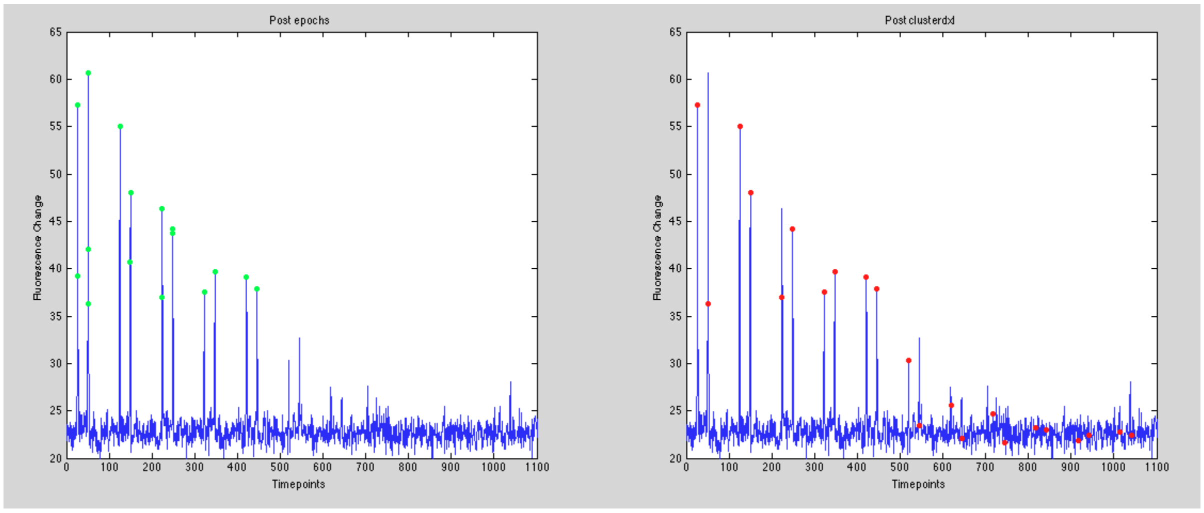

Figure 3: Matrix of fluorescence changes between frames. `amat` is plotted in blue, whereas `ind`, the values beyond a user-defined threshold are plotted as points. On the left, the points represent indices saved after `epochs`. On the right, after clustering.

As we construct `amat`, we compare each element to the previously defined threshold, and if it is larger, we save its index in `ind`. At the end of the for loop that runs through `newmat`, `ind` contains the indices of all the values in `amat` that are larger than a threshold. Ideally this would give us the single indices of the edges of every response step, but that does not happen. What `ind` contains in the best situation are the indices that correspond to the edges of the larger response steps, and we lose the information about the smaller response steps. In addition, for some of the edges that we detect, especially the earlier ones, we obtain multiple indices that correspond to the same edge, as the change in fluorescence cannot be instantaneous, and thus may span multiple frames, all of which are greater than the threshold value, and gets registered in `ind`. This is represented by the plot on the left in Figure 2, in which some of the peaks have multiple points, and some of the peaks (post 500) don't have any. In order to obtain an array of single indices of all the edges of the response steps, we send `ind` to `clusterdxl`. Finally, we obtain an average of all the non-zero fluorescence intensities of each frame in `newmat` and `newmat2`, which are saved in `averageint` and `averageintbgsub`, respectively.

4.2 Clustering

The strategy in clustering is that we would like to select a single index for the edges that are represented by more than one index, and reconstruct the edge indices that were not picked up after thresholding in `epochs`. Since all the indices that represent a single edge will be within a range smaller than the distance between any two adjacent edges, we can set a radius and cluster all the indices in `ind` that fall within a narrow range of each other via `clusterdxl`. This function inputs the number of steps, `numsteps` and `ind`, and outputs:

`singleindexarray`: `ind` but with only one index representing an edge
`arr`: the completed array of indices for every response edge

`ind` is clustered in cell arrays, which are indexed data containers called cells. Each cell could contain a different size, or type of data, which is ideal for clustering, as all indices in a cluster are stored, and indexed, in cell elements and each cluster is stored in a cell, making up a cell array, assigned to `clusterarr`. We can then select, within each cell which contains all the indices that correspond to a single edge, the single index that will allow us the largest sampling of a plateau response region. Briefly, since all odd-numbered cells are the rising edges and all even-numbered cells are the falling edges, we pick every first element of an odd cell and every last element of an even cell in order to span the entire response peak. These single indices are saved in `singleindexarray`.

Finally, we reconstruct. From `singleindexarray`, we can calculate an average `interval`, which is the width of a response peak, and an average `period`, which is the length between the start of two responses. For each index in `singleindexarray`, we test whether it is within a reasonable range of either the interval or the period with the previous element, and if it is, we keep that index, and if it is not, we insert a new index by adding either `interval` or `period` to the previous element. We iterate this check until we have reconstructed an array such that all the edges are represented by an index. This array is stored in `arr` and will have a number of elements that is two times the number of response peaks (each peak has a rising and a falling edge). The panel on the right in Figure 2 plots the single indices of all the response edges in red. The points may not be plotted on the highest peak as we're selecting the indices that correspond best to the start of the response, not neccessarily where there was the largest change in response. And thus, we can send our array of indices, `arr`, to `edgestxt`, which will compile everything and calculate the ΔF/F.

5 Calculating ΔF/F

At this point `mastersc2` and `mastersc3` converge: whereas `mastersc3` has calculated the edge indices, which will be unique to every tiff stack, `mastersc2` inputs a protocol with predefined edge indices, which, although will not match up perfectly with the actual edges of the current timeseries, can reduce computation times, especially when applied to a batch of characterization timeseries with the same experimental parameters. Note that `mastersc3` runs `edgestxt2`, which is essentially the same as `edgestxt` that `mastersc2` runs, except the former loads the average raw and background subtracted fluorescence intensities, `averageint` and `averageintbgsub` that we've previously calculated in `epochs`, whereas `edgestxt` calculates those values in `mastersc2`, which does not run `epochs`.

We adjust the edge indices by a small number, `adj`, in order to ensure that we are solidly in the range of the response plateau. This new edge index array is saved in `matprot`, which we send, along with `averageintbgsub` to `avgintensity2` in order to obtain the average fluorescence intensity for each response step. This is stored in a new array with as many elements as the number of response steps, in `avgint`. A similar procedure is used to obtain the average baseline fluorescence intensity for each interval after a response step via `subbaseF2`, the only exception being that we had to first construct indices for the baseline intervals from `matprot`. The array of baseline edge indices is saved in `sbmat` and that of the average intensities is saved in `sbF`. Since we've been using `averageintbgsub`, any fluorescence intensites we work with henceforth are all background subtracted unless otherwise stated. We then send `sbmat` and `matprot` to the function `intervaldx`, which calculates the median index between two edge indices.

Once we have `avgint` and `sbF`, which are arrays of the average fluorescence intensities of the response steps and the baseline intervals respectively, we can obtain ΔF, stored in `delf`, and ΔF/F, stored in `delfof`. We store the full (unaveraged) ΔF and ΔF/F intensities in `delfint` and `delfofint`. They act as an immediate visual verification upon plotting that the averages are close to what they should be. Other points of reference that we have written into `edgestxt(2)` are that we have converted the edge indices in `matprot` and `sbmat` into their binary on-off steps counterparts, `binarymat` and `binarysbmat`, with a raised "on" (`1`) step being that the fluorescence information contained within was considered. In this way, if an unideal ΔF/F result stems from the determination of edge indices, we can immediately see if the calculation for a response step also took a non-response interval into consideration, and vice versa. In addition, we also plot the average fluorescence at each response step, the baseline fluorescence after each response interval, and the final ΔF/F result directly on the traces so that it is immediately clear if they deviate from an expected value that the user can estimate from the traces.

Finally, we calculate the signal-to-noise ratio of the response step with a 100-mV change.

As we now have the ΔF/F, stored in `delfof`, and we know the applied voltages at each response step, we can now plot ΔF/F as a function of applied voltage, fit a line to the data and output the R-squared value. After a run through our function `edgestxt(2)`, we end up with two figures: one features four panels with the raw F, background-subtracted F, ΔF, and ΔF/F traces with the previously described on-off steps and averages and the second with the ΔF/F versus voltage applied information along with a best-fit line and SNR. Both figures are saved automatically in the same folder as that containing the movie files.

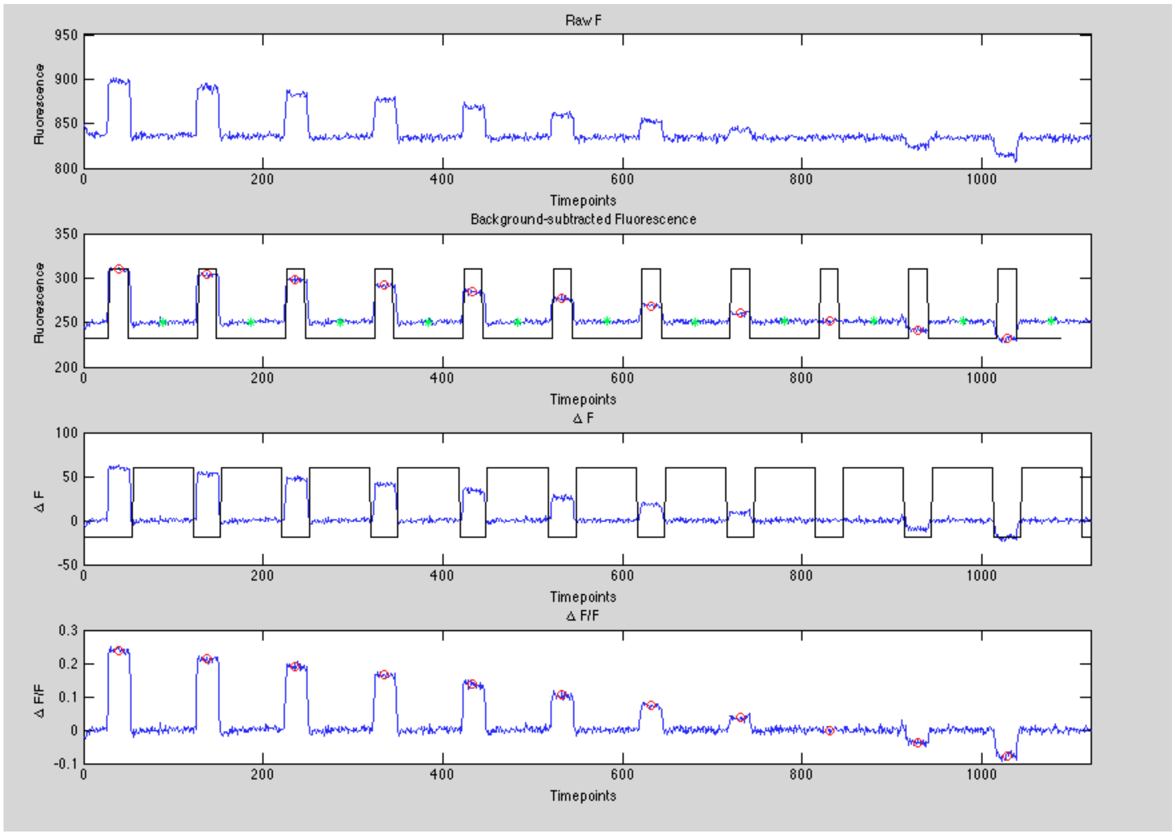

Figure 4: Four panels of fluorescence information; the on-off steps for `matprot` are plotted on the second panel; for `sbmat` on the third panel, along with the averages of that interval.

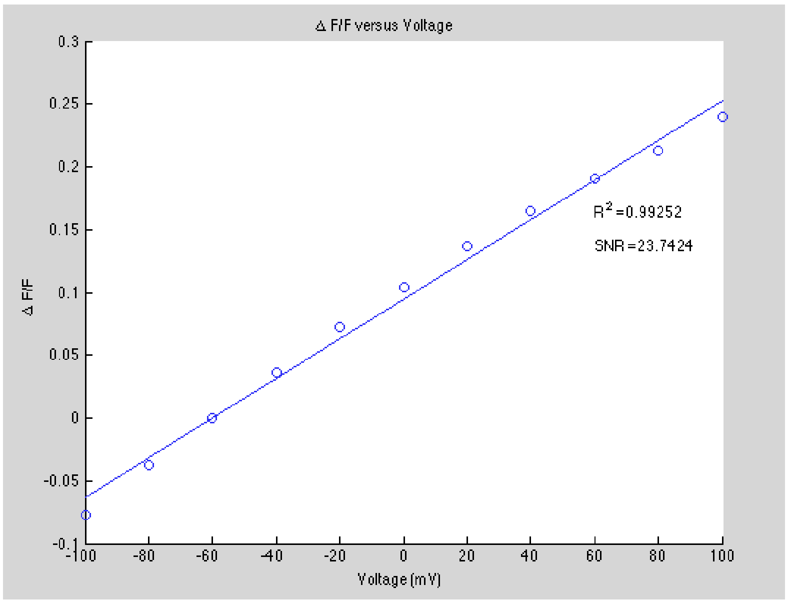

Figure 5: Timepoints data is converted to applied voltages, and plotted against ΔF/F, as well as a linear fit and the SNR for the -40mV step.

In summary, `edgestxt(2)` takes the array of indices that were either input in a protocol or computed in `epochs` and `clusterdxl` and obtains the relevant fluorescence information to ultimately plot ΔF/F as a function of applied voltage, which gives us quantitative information about the characteristics of the dye in terms of how its fluorescence changes at a certain voltage. With this procedure, we can then standardize and streamline all voltage-sensitive dye characterizations and be able to objectively and quantitatively compare one dye to another. All calculated variables, including fluorescence, edge indices, SNR, R-squared, etc. are saved in a data matrix and can be retrieved at will simply by calling the `load` command.

6 Physiological Range

The applied voltage range can be extended and applied from -300mV to +300mV. At either extremes we observe that the relationship between ΔF/F and voltage is no longer linear and tapers off: here is the maximum change in fluorescence on either ends; a larger applied voltage difference will not be able to elicit a larger change in fluorescence. Characterizing the dye in this range is useful as it allows us to verify that our experimental range, from -100mV to +100mV falls within the linear region. Both `master2` and `mastersc3` are able to run characterization movies in the extended range, and can carry out the appropriate course of action from the user input of the number of response steps.

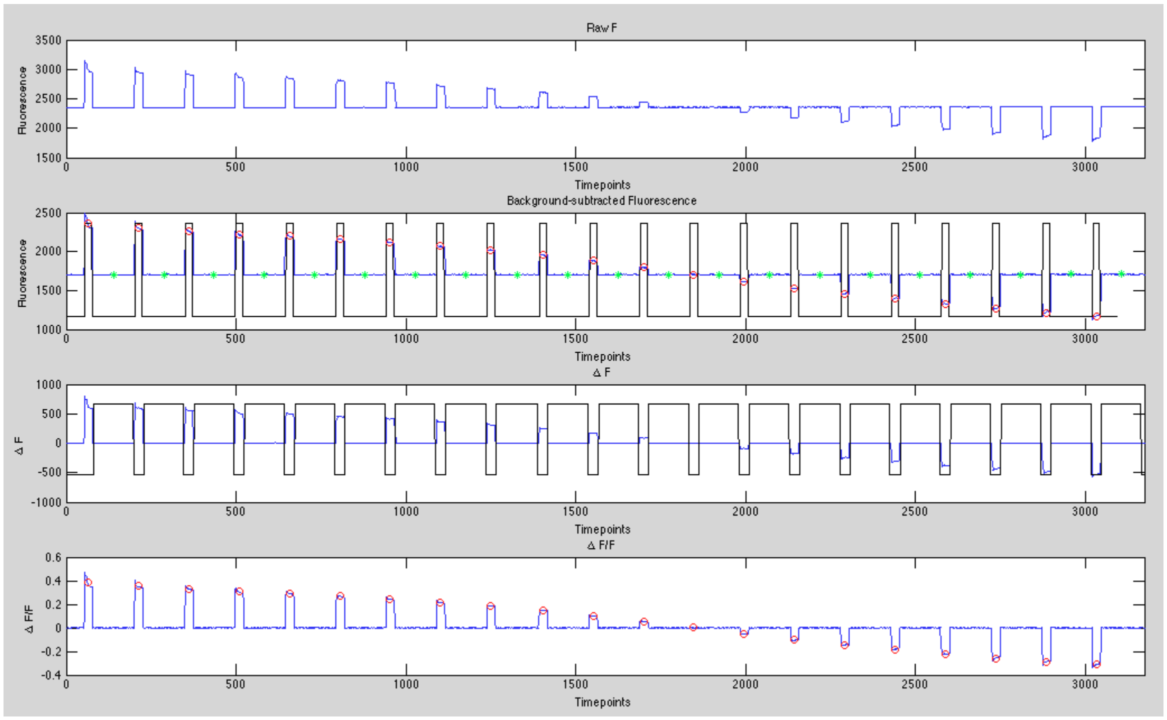

Figure 6: Four panels of fluorescence information for the voltage range -300 to +300mV.

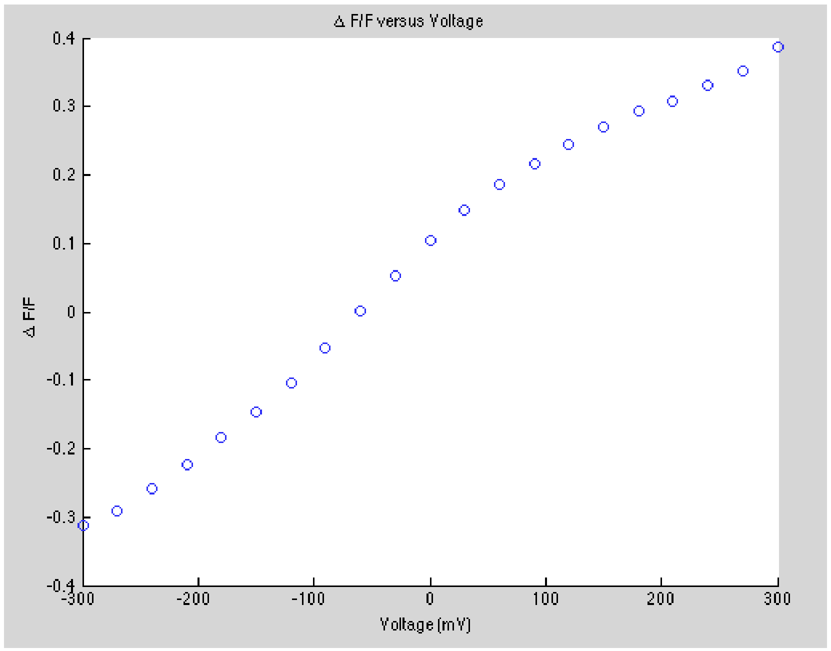

Figure 7: Timepoints data is converted to applied voltages, and plotted against ΔF/F. We see that the experimental range from +100 to -100mV is not perfectly linear.

7 Discussion

Having described the computation behind the master scripts, we can now make a comparison between the two. We run the same tiff stack through both scripts, and compare the two protocols with each other. That from `mastersc2` is always plotted on the left, whereas that from `mastersc3` is plotted on the right. We explore two different scenarios. When the pre-loaded indices in `mastersc2` describes the edges well, which would be immediately clear in the figure with the four panels, the two scripts have comparable accuracy–their R-squared values of the best fit line are both larger than 0.99–and comparable SNR.

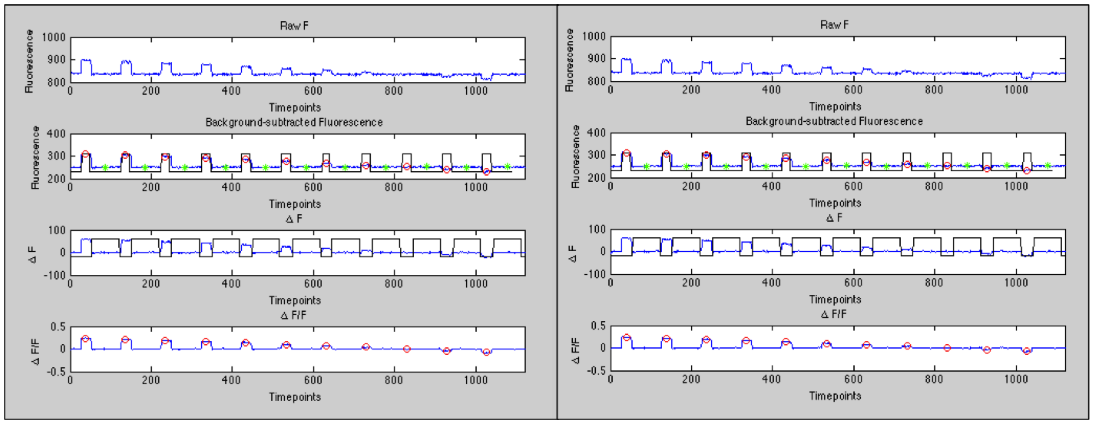

Figure 8: The protocol from `mastersc2` corresponds well with the experiment regime and defines edges as well as `mastersc3` detects them.

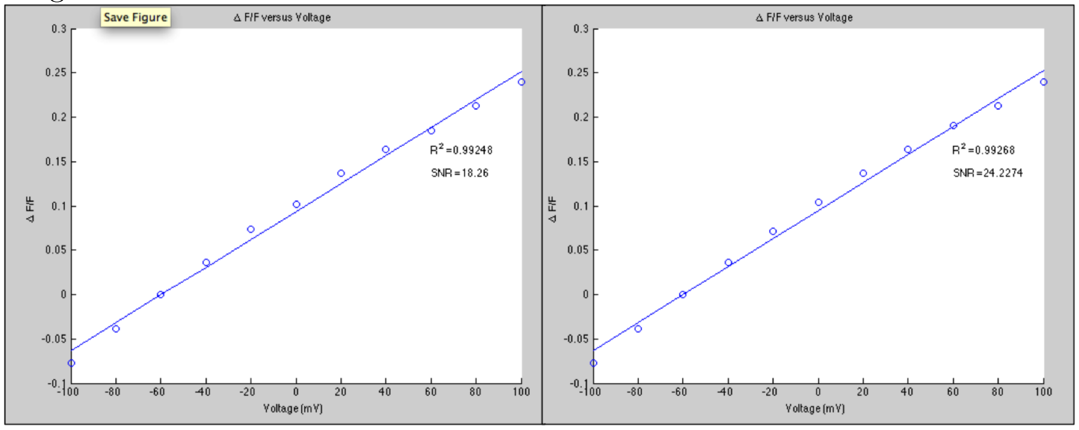

Figure 9: Both scripts have comparable goodness-of-fit and SNR.

On the other hand, sometimes different experimental parameters may be used for a characterization timeseries: the length and frequency of the response intervals are variables that could render a pre-loaded protocol ineffective. In these scenarios, however, `mastersc3` is the only reliable script that can still compute the characterization quantifications with precision and accuracy.

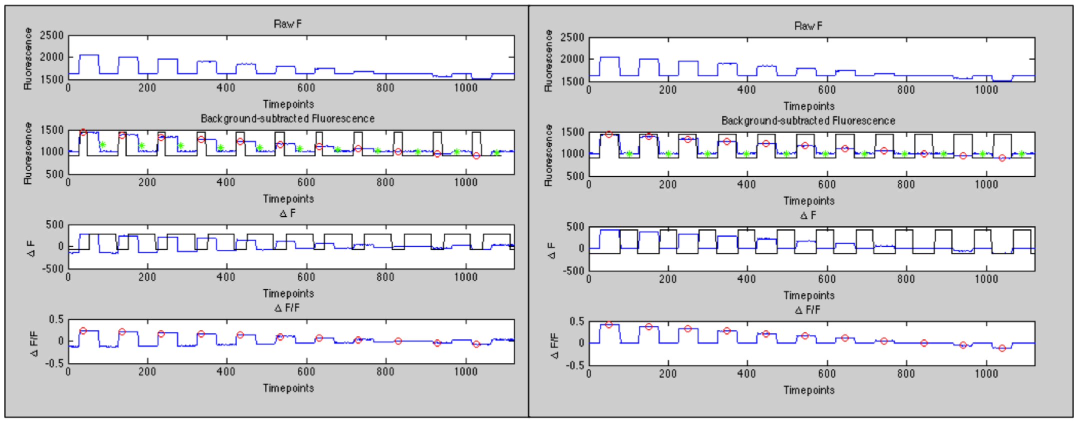

Figure 10: The "on" indices in the protocol from `mastersc2` corresponds well with the rising edges of each peak, but the off-indices are premature, as are the on-indices for each baseline fluorescence interval, thus resulting in an average baseline fluorescence that is not true to value. `mastersc3`, however, can still recognize all edges with precision.

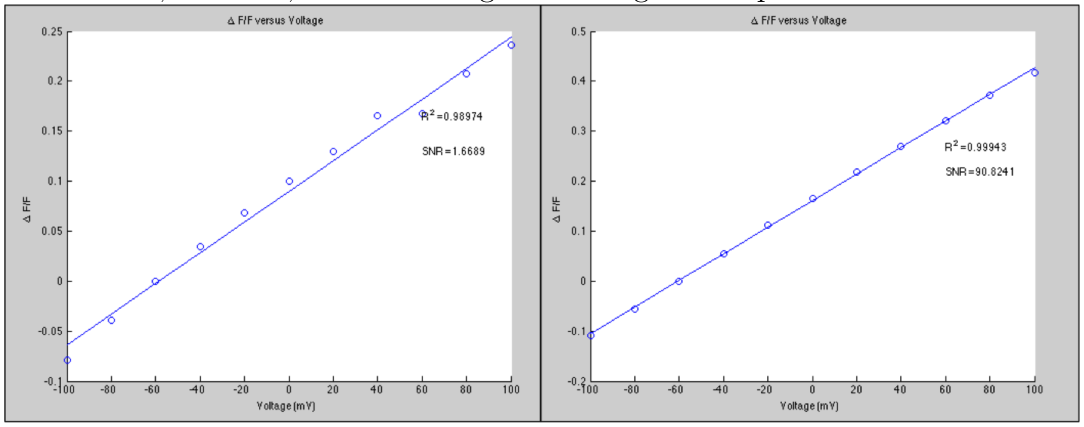

Figure 11: The difference in goodness-of-fit and SNR is more pronounced, and `mastersc3` once again provides reliable analysis.

8 Conclusion

TBW