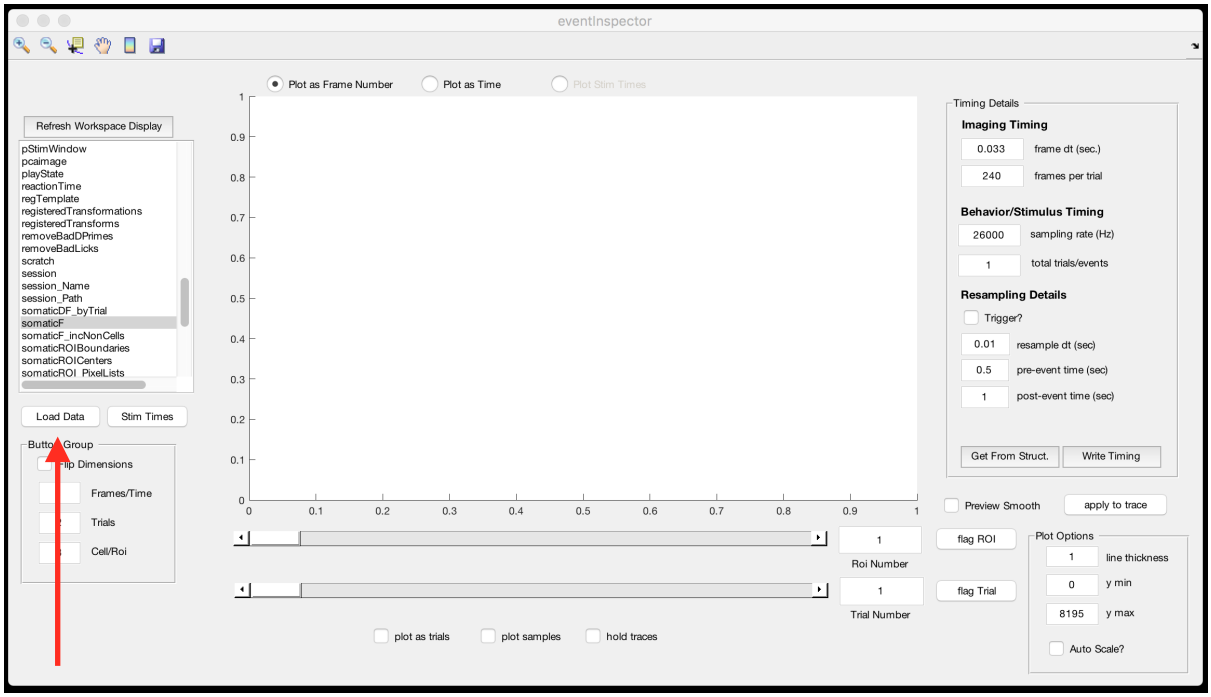
eventInspector
documentation started: 4/10/2016; updated: ; cdeister@brown.edu

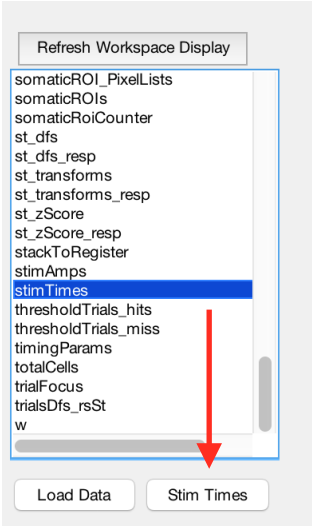
The point of eventInspector is to look at your data in the context of 'events.' Events are most often a stimulus or cue time, but could be anything. The main thing you need to make use of this is a matrix of data, a vector of event times, and some basic knowledge of how your data and events were collected/ generated. Optionally, you can configure and use a struct with relevant timing info using fields we will discuss later. For now let's start working through some data.

To open eventInspector, make sure imageAnalysisGui and all subfolders are in your path. If you care about how *eventInspector* works you probably made it this far already.

type 'eventInspector' in the matlab command line. You should see this:



This screenshot is from version 2.9 of the package, so it may not look exactly like this, but it will be close. The first thing you should do is select a matrix from the 'workspace browser.' Here I have selected a matrix that I have extracted with extractor from a bunch of somas called 'somaticF', this is the default from *extractor*, so you probably have one of these too. eventInspector works best with a 2-D matrix that contains M(rois) x N(frames), but you can do the opposite, and let eventInspector know, by toggling the 'Flip Dimensions' button that is slightly obscured by my red arrow. Once you know what you want to load hit the 'Load Data' button that I've pointed to above with the red arrow.

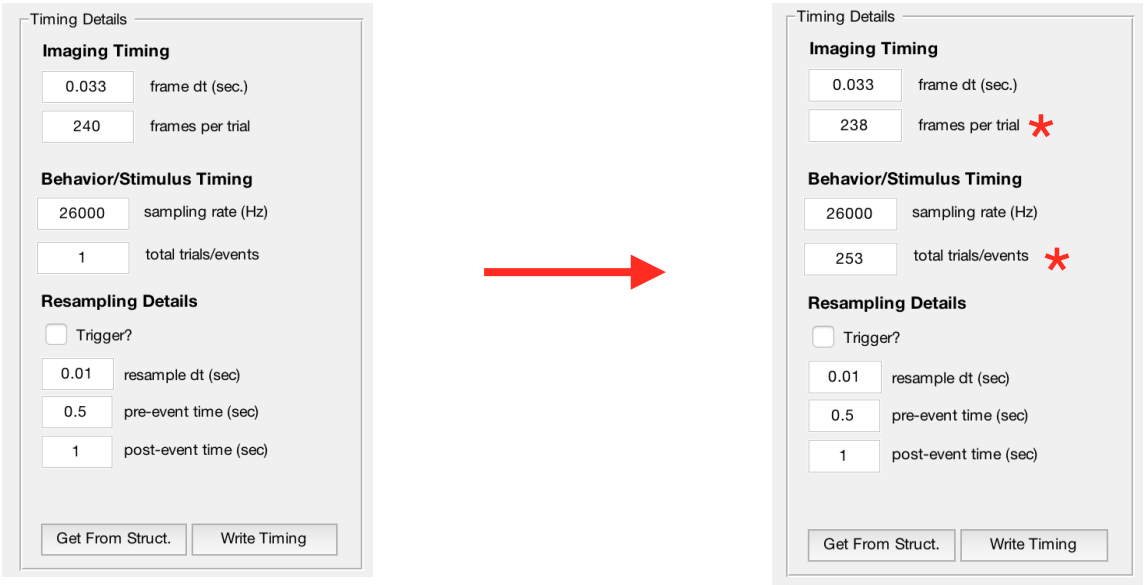


When you do this, you will see a trace plotted for the first roi and all frames. From then on, eventInspector understands what you are working on so you can choose something else from the workspace browser. Go ahead and now select a vector of event times, and then click the button called 'Stim Times' next to the load button:

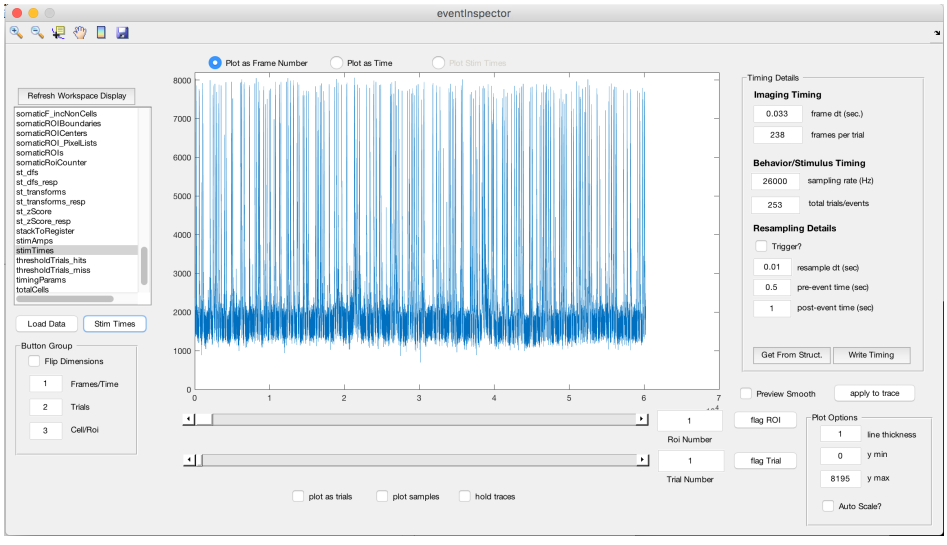
The screenshot to the right shows the button called out in red. Once you click 'Stim Times' this will load the times into a struct in your workspace called 'scratch.' The whole point of scratch is to store relevant temp variables, that

you don't need to worry about. Point is eventInspector now 'knows' these times. It then tries to help you out and make some assumptions, or figure out some of your timing information for you. Don't worry if it gets it wrong you will be able to modify things. Speaking of which let's turn our attention to that real quick.

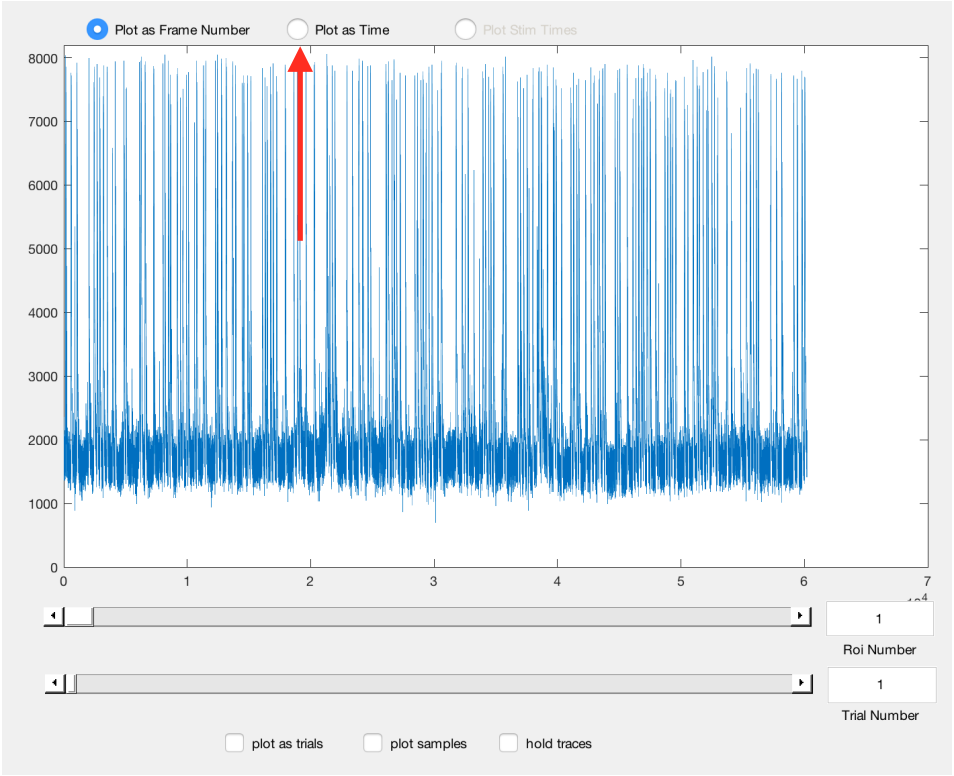
On timing. At this point the necessary timing details you need to see something are minimal, but here is what happens when you load stim times. The left image below shows the initial state of the 'Timing Details' portion of the program. Once I load my 'stimTimes' vector, which is a 253 element vector of times, some fields update as you can see on the right.



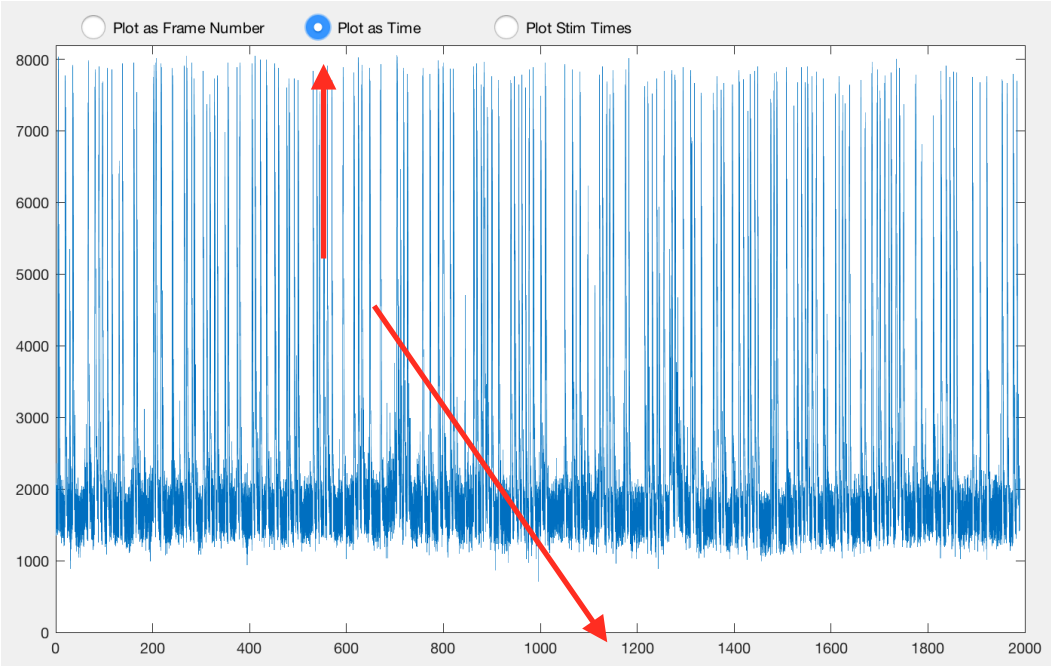
You can, and likely, deliver your stimuli differently, but what I say should generalize. I collect trials of images whose onset are synchronized to a behavior control computer that uses standard AD/DA boards to generate and deliver stimuli. I have 8 second long trials in this example, but within the 8 seconds the stimuli came on at variable times, which are listed in 'stimTimes'. So trial 1 will have a time of 1.23 seconds, but trial 2 might be 2.6 seconds. eventInspector will use the number of stimTimes as my 'total trials/events' value denoted above with a red star. My data in this example is a 24 x 60214 matrix. There are 24 neurons and 60214 frames. The imaging/behavioral session was broken up into 253 trials, so the number of frames per trial were 60214/253, which comes to 238 and is calculated and entered for you in the 'frames per trial' entry box also pointed out with a red star. If it is wrong, feel free to enter the real value. You should now see something like this:



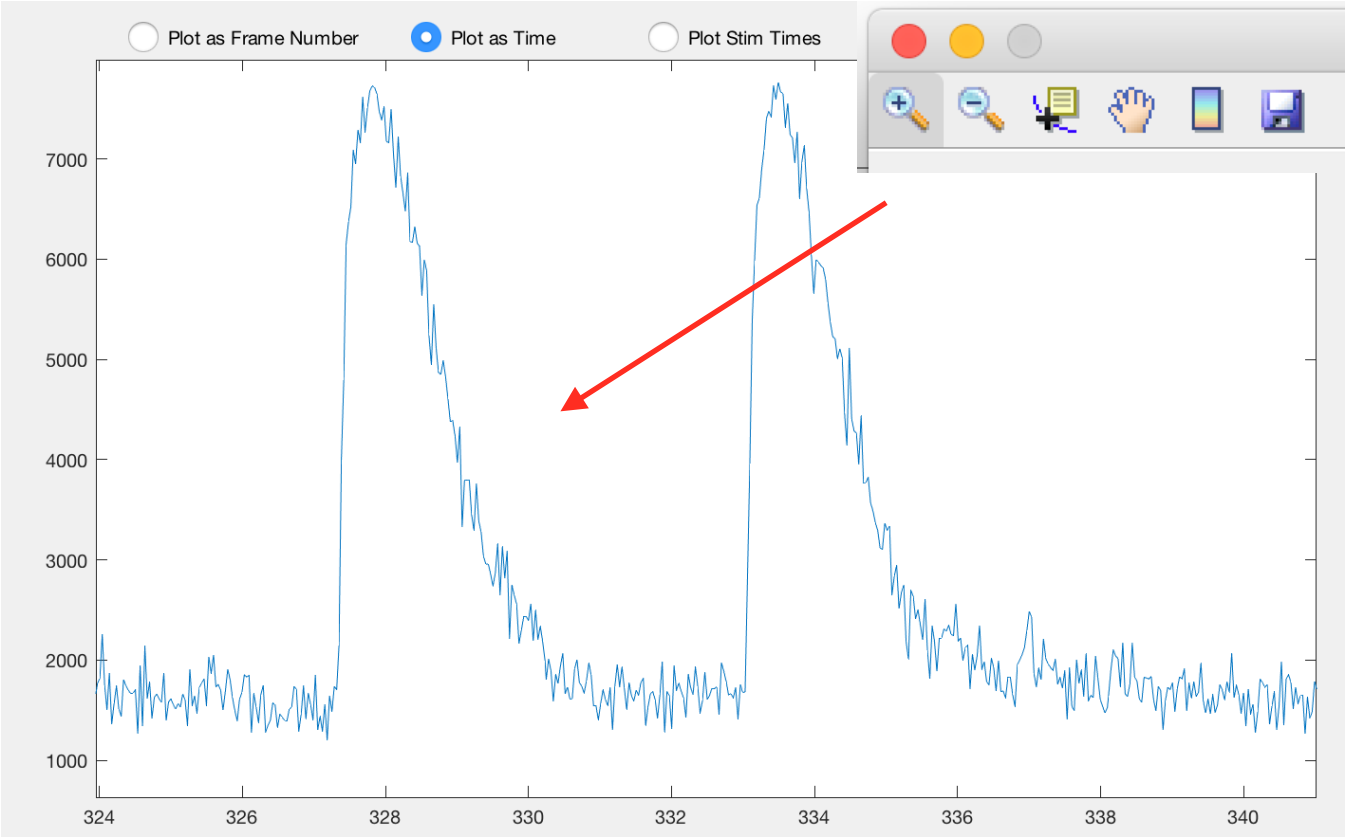
Now let's turn our attention to the trace and how it is plotted:

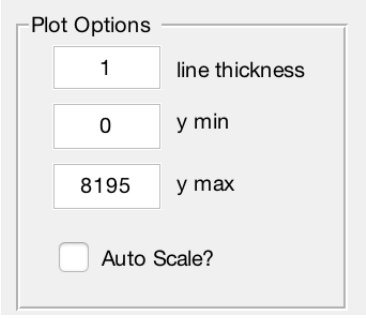


Here you will see that the initial plotting is done for Roi Number 1, with the Y values being the raw values I loaded (raw fluorescence values in time), and the X values are frame number. The frame interval is specified in the timing window, so if you click 'Plot as Time' (red arrow) you will now see your trace plotted in time (seconds):



In this example my data are an imaging frame's values for that ROI across 60214 frames collected at a sampling interval (frame interval) of 0.033 seconds, which I had specified in the timing window. It therefore made a time vector from the first point in time to 60214 * 0.033, which is 1987.062 seconds. Now the standard matlab plot tools are in eventInspectors toolbar so you can zoom in with those:

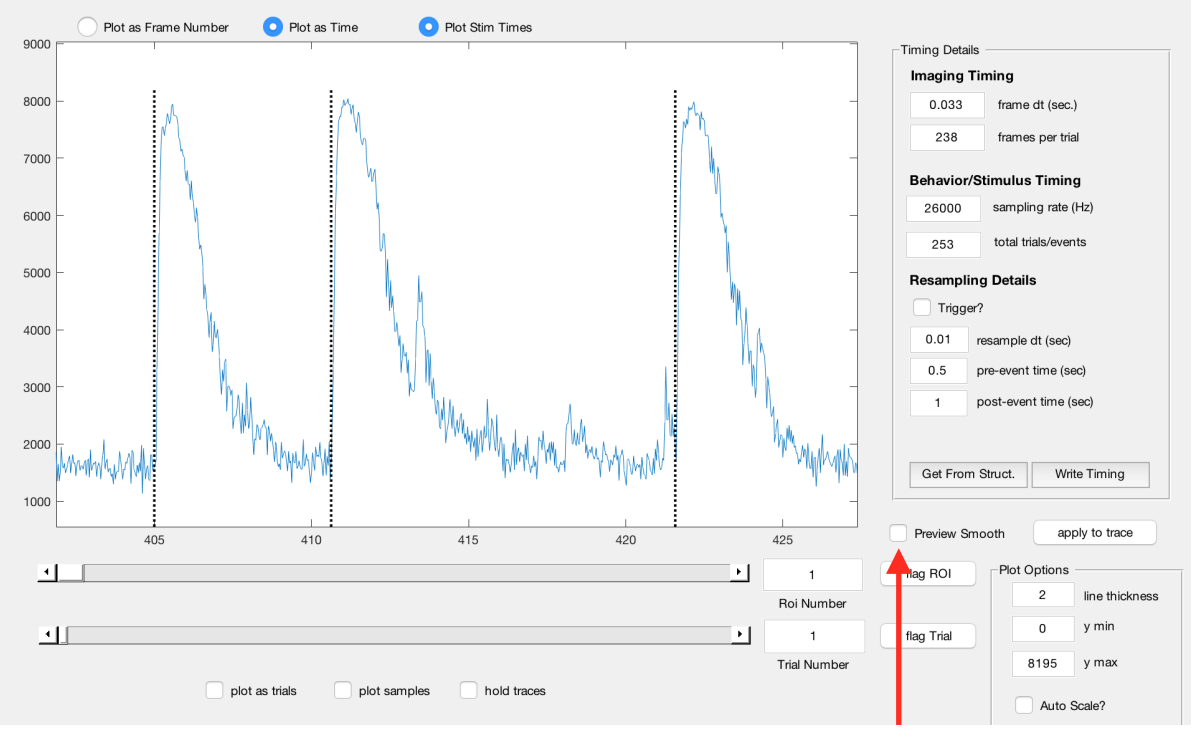




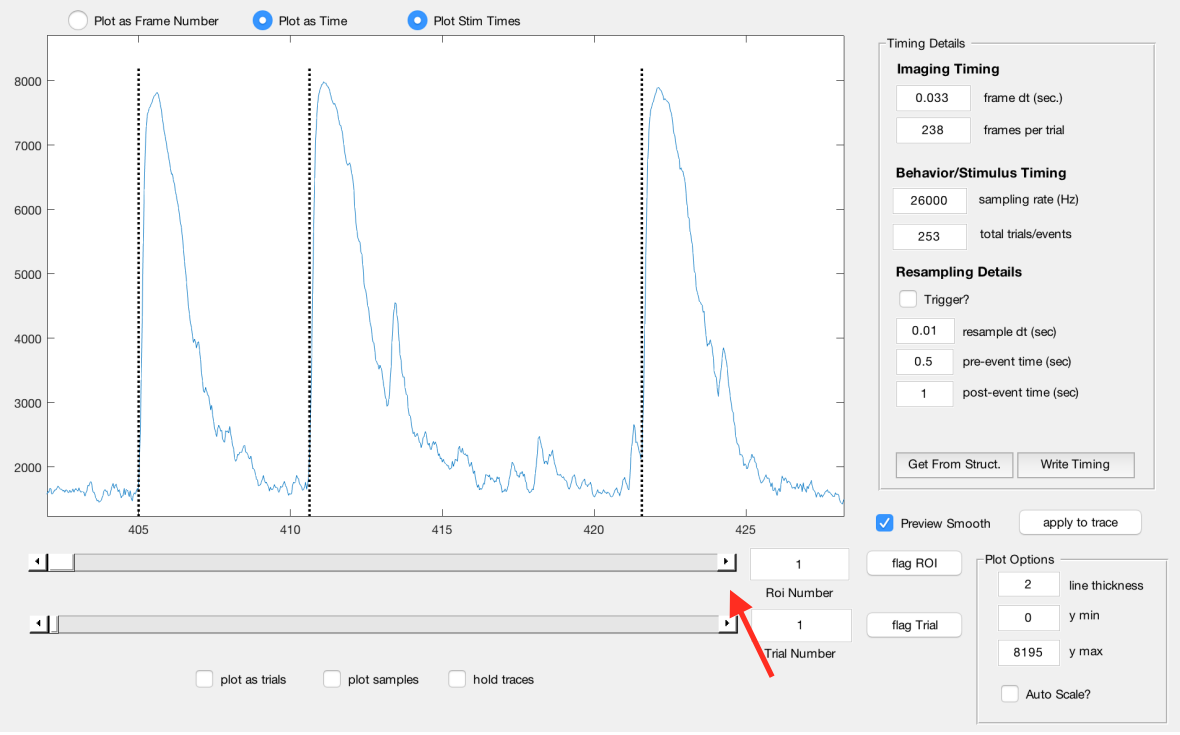
You can also change the y-axis values using the 'Plot Options' entries (look to the left). It will default to using the 'y min' and 'y max' values in these fields, which is the valid range for a 16-bit image (2^16; minus one for 0). But, you can also toggle 'Auto Scale?' and it will find the min and max of the ROI's data you have loaded and use those as its bounds.

The other thing to realize is that if you look at the trace above you will see to the right of 'Plot as Time' and option called 'Plot Stim Times' is available. It is grayed out and unavailable to you until you load some stim times. It is also unavailable to you if you are plotting data as frame number. If you toggle this it will display some dashed lines over where your stim times are, the lines will have a thickness of 1, by default, but this can be changed by changing 'line thickness' in the Plot Options you see to the left.

Once you do toggle 'Plot Stim Times' you will now see:

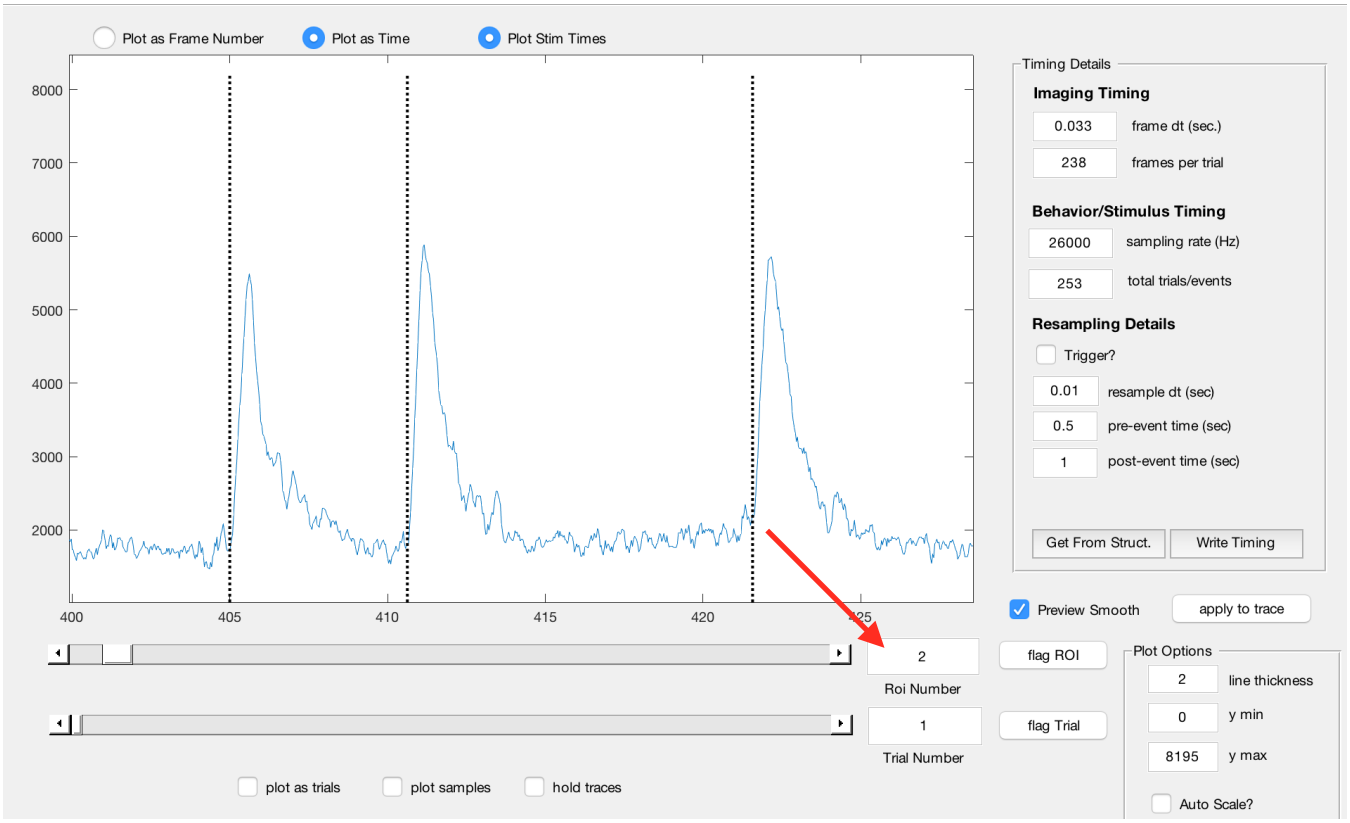


The dashed black lines are stimulus times, and the blue traces are still the data. Note that I changed the thickness of the stim lines to 2. Now I am going to toggle 'Preview Smooth,' as noted by the arrow:



Which applies a simple low-pass filter to the data. This can be left on, and you can save the results by clicking 'apply to trace,' which will save the filtered version to your matrix (more on that later).

If you toggle the 'Roi Number' (red arrow) you will get the next cell:



This is roi number 2, but similar time etc. to the one above. Now hitting the arrow, on the slider, changing the number and hitting enter, as well as clicking in the 'dead zone' of the slider itself will all change the roi by 1, do not drag the slider itself, as it does not 'scrub.' Instead, it will move in non-integer steps and throw and error. As far as I can tell, this is default behavior in matlab and I can't override it without subclassing that GUI element, which is not worth it.