[spikes\_struct] = detect\_events(templates,data(i).data,desired\_FP,pre\_spike,post\_spike,cutoff,fs,'Flowpass',20,'ratio\_thresh',3);

In the form [spikes\_struct] = detect\_spikes(template,data,desired\_FP,pre\_spike,post\_spike,cutoff,fs,varargin);,

template - A matrix where each row is a spike chosen to serve as a template.

template needs to be templates x time bins. Assumes templates were

centered at 1, otherwise it freaks out.

data - A matrix, where each column is an ROI, a trial etc. whatever you

are trying to do detection on. Data needs to be time bins x ROIs

desired\_FP - The desired false positive rate per sample. Ideally you choose a false positive rate such that in all of your

data you have a small chance of having one, say 1 in 20 (across all of your data),

as 0.05 is accepted as a standard. Inputting a string for desired\_FP results

in the threshold being the point where the signal and noise distributions meet.

pre\_spike - The number of sample points before the spike peak you want

to keep

post\_spike - The number after the spike peak you want to keep.

Cutoff - The amount of time in seconds you want shaved off of the

beginning and end of the filtered trace, so that filtering artifacts are

not included in the detection.

fs = The sampling rate the data was acquired at.

varargin:

'Fpass' - For filtering the data with a 2'nd order butterworth, it sets

the frequency cutoff for detection. The data itself is high pass filtered

at that frequency. If nothing is input, the default is 20.

'No\_Filt' - means the user already applied their own filtering, or otherwise

does not want the data being input to be further filtered.

The output: spikes\_struct is a structure with all relevant detection information

and spikes.

‘ratio\_thresh’ - The point in loglikelihood space (threshold for the probability vector)

HOW TO CALCULATE 'ratio\_thresh': ratio\_thresh is the cutoff in the probability vector, referred to as ratio in the code because it is the log likleihood ratio of the probability the

data is the template, above which a timepoint is considered to come from

the templates. Since it is the log of the probabilities, 0 means it is

equally likely to have come from either the templates or the noise,

because log(1) = 0, and if prob\_signal = prob\_noise, their ratio is 1.

For standard 1/20 biology significance levels, your threshold would be

log(20), or 2.9957. For imaging data this isn't realistic though, because

if you are sampling at 1000hz for example, for 11 seconds, then that is

11,000 chances for a false positive. In this example, you could

bonferroni correct and say you want a 1/20 chance per sample, so

log(20\*11000) = 12.3014 as the 'ratio\_thresh'. You would type 'ratio\_thresh',12.3014.

Bonferroni corrections are also known to err on the side of caution, so it will

be up to you to decide exactly where and how you want to set your

threshold, but this is how the code works. Where we set the cutoff determines our true and false positive detection rates. Note that we only know them because we are assuming our templates represent ground truth. Without that, we are out of luck.

Template is centered at the background, called base, at data point 1000. 100 was chosen arbitrarily. Base is currently assumed to be a vector of ones. The templates are taken in as a stack, where each column is a time bin. Each column (time bin) therefore has a mean and

a standard deviation. If there are less than 10 templates, the standard

deviation of each of the time bins is assigned to the standard deviation

of the noise, called Noise\_std. Noise\_std is the standard deviation of

the data after the background is subtracted. Since it is assumed that the

data is already normalized to 1, this means it is the standard deviation

of the data centered at 0. The number 10 for the template threshold was

chosen arbitrarily. length(template) data points are grabbed , and the

gaussian integral of each timebin is taken twice. Once for the noise

integral, and once for the template integral. The noise integral is the

integral from the data point in question to +inf on the noise

distribution, which is defined to have a mean of 1, and an std =

Noise\_std. The template (signal) integral is taken from -inf to the data

point in question on that particular time bin in the template

distribution, which will have a mean equal to the mean of the template at

that point, and an std equal to the std of that point in the template

stack. If there are less than 10 templates or if the template std at that

point is less than 1/100 of the Noise std, it is assigned the value of the

noise std. Noise std is assumed to be the

same as the data (Noise\_std), which should be a good approximation. If

the indicator is a downward going indicator, or if parts of the template

dip beneath the baseline, set to 1, then the integration limits are

flipped. That is, the template (signal) integral now goes from the data

point to +Inf.

if isempty(ratio\_thresh)==1 If the user wants to determine a threshold by estimating FP rates from the statistics of the data

EXAMPLE:

desired\_FP = 1/(11000\*20);

cutoff = 0.1;

fs = 1000;

pre\_spike = 100;

post\_spike = 50;

dbstop if error

[spikes\_struct] = detect\_events(templates,ASAP3\_T3,desired\_FP,pre\_spike,post\_spike,cutoff,fs,'Fpass',20,'ratio\_thresh',log(20\*(11000/size(templates,2))));

**How the threshold is set in log likelihood space if you don’t manually set it**

The log likelihood ratio is calculated, but instead of feeding it data the individual traces from the template matrix are fed in as data. Since we assumed these were all real spikes, we now know what the log likelihood ratio should look like when we hit a spike. Specifically, we have a mean and a standard deviation that we can define a signal distribution for in log likelihood space.

We do that again, but this time feeding it the average of the sgolay filtered baseline trace, which we know is noise. This gives us a number, which is the mean of the noise distribution in log likelihood space. Importantly, I assume the standard deviation of this distribution is the same as the signal one. Some of the rationale for why this had to be done is in the comments in the code, but it means that the templates should come from data that was recorded under similar conditions as the data it is being compared to. For example, you wouldn't want to use templates from 100mW of power on a sample where 25mW of power was used, as it would greatly underestimate the noise.

Now that we have a known signal and a known noise mean, we can compute d' on them. We can also set a threshold in the log likelihood ratio that will give us known detection rates, given that we know where the signal and the noise distributions are.

Please let me know if any of that is unclear, I am more than happy to do a better job explaining it. I will have to for the paper. Also let me know if you think something better could be done. The basic idea is that in the absence of ground truth (e.g. an electrode), how do we establish ground truth in noisy data? I am trying to do it basically using a p-value that takes into account what we are looking for and the noise of the data.