

## Summary

Modern techniques in optophysiology have allowed neuroscientists unprecedented access to neuronal activity *in vivo*. The time series datasets generated from these experiments are becoming increasing larger as new technologies allow for faster acquisition rates of raw data. These fluorescent recordings are being made with an ever expanding library of indicators for calcium, voltage, neurotransmitter and neuromodulator activity. These signals generated from these fluorescent bioindicators contain the information of the underlying neuronal activity but all have unique molecular kinetics and inherit signal-noise ratios which must be taken into account during signal processing. The development of pyNeuroTrace, an open-source Python library, was made to aid in the processing of these neuronal signals which must be filtered with these unique aspects in mind before analysis can be completed.

## Statement of need

Many neuroscience labs that use optophysiological methods, such as two-photon microscopy or fibrephotometry, frequently must rewrite and maintain common functions and filters needed to analysis the raw recordings. Furthermore, many technique and algorithms for signal processing are scattered throughout the literature and are frequently implemented programming languages other than Python. **pyNeuroTrace** meets the need of a time series analysis written purely in Python for neuronal activity. Our package is a collection of filters and algorithms implemented in a generalizable manner for time series data in either 1D-arrays or a collection of recording in a 2D-arrays. Additionally, with the increase in acquisition rate of new imaging techniques, we have implementations of these algorithms using GPU compatible code to increase the speed in which the techniques can be applied to larger datasets collected at kilohertz rates.

## Signal Processing

### DeltaF/F

There are several methods for calculating the DeltaF/F of a fluorescent trace. We implemented the method described by Jia *et al* which includes several smoothing steps to help with shot noise[@Jia2010]. In short, F0 is calculated by finding a the minimum signal in a window of the rolling average of the raw signal. Then DeltaF is calculated by the difference in the raw signal and F0 divided by F0 (F-F0)/F0. This DeltaF/F signal is optionally smoothed using an exponentially weighted moving average (ewma) to further remove shot noise.

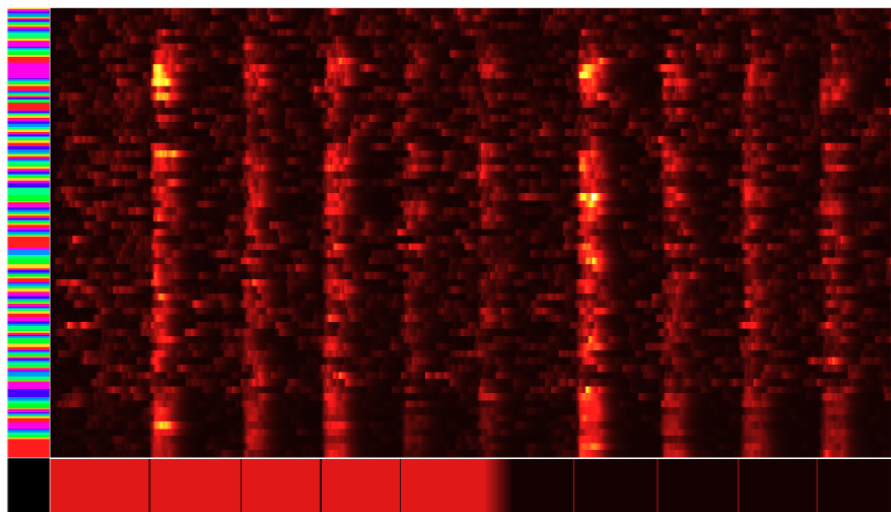
## Okada Filter

We implement the Okada Filter in Python[@Okada2016]. This filter is designed to filter shot-noise from traces in low-signal to noise paradigms, which is common for calcium imaging with two-photon imaging where the collected photon count is low and noise from PMT can be nontrivial.

## Nonnegative Deconvolution

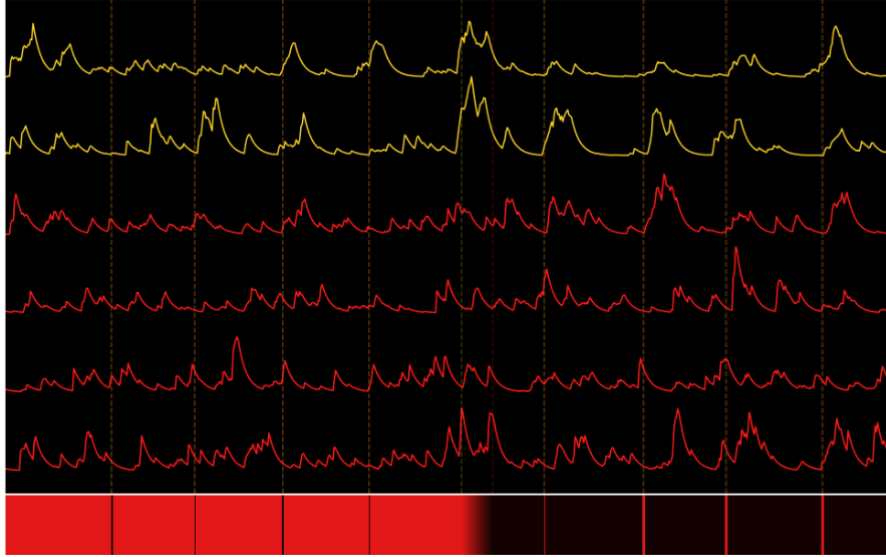
## Event Detection

## Visualization

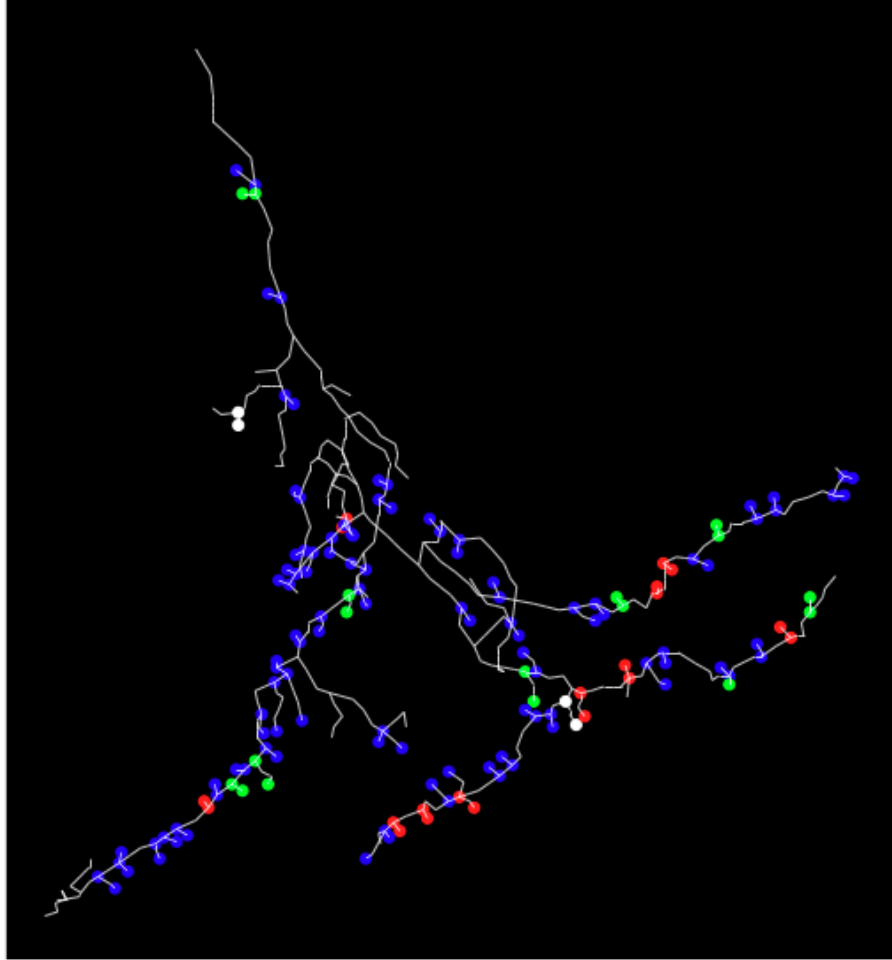


Figures can be included like this:

and referenced from text using ?? Figures can be included like this:



and referenced from text using ??.



Figures can be included like this:

## GPU Acceleration

Several of the filters in `pyNeuroTrace` have been rewritten to be almost entirely vectorized in their calculations. The benefit being a noticeable difference in the performance for larger time series. These vectorized versions gain further speed by being executed on a GPU using the Cupy python library [cupy\_\_learningsys2017]. To execute these versions the filters can be imported from the module, `pyneruopyneurotrace.gpu.filters`, and a CUDA compatible graphics card is needed. This functionality is becoming increasingly important as acquisition rates increase for kilohertz imaging of activity can generate arrays hundreds of thousands datapoints in length in just a few minutes. Figure 1 shows the difference in calculating arrays of various sizes using either the CPU or vectorized GPU based approach of the dF/F function. The CPU

used in these calculations was a Intel i5-9600K with six 4.600GHz cores, the GPU was a NVIDIA GeForce RTX 4070 with CUDA Version 12.3.

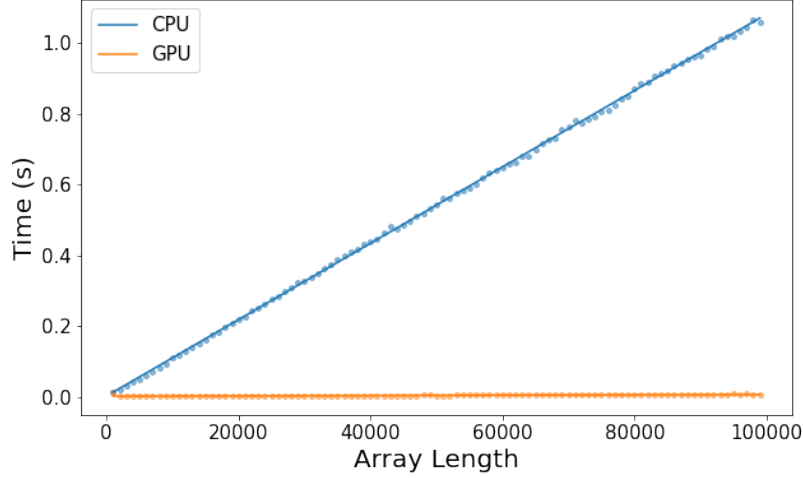


Figure 1: Comparison between dF/F with EWMA calculations for different array sizes.

To vectorize the functions several where modified. For example the EMWA used to smooth the dF/F signal as described by Jia *et al* was changed to an approximation using convolution with an exponential function. The kernel used to perform is defined as:

$$w[i] = \begin{cases} \alpha \cdot (1 - \alpha)^i & \text{for } i = 0, 1, 2, \dots, N - 1 \\ 0 & \text{otherwise} \end{cases}$$

Where  $\alpha$  is defined as:

$$\alpha = 1 - e^{-\frac{1}{\tau}}$$

$\tau$  where is a user selected time constant which is translated into number of samples.  $N$  is a window parameter for the kernel calculated using  $\alpha$ :

$$N = \left\lfloor -\frac{\log(10^{-10})}{\alpha} \right\rfloor$$

This allows a filter for smaller values that have a miniscule influence on the weighted average. The kernel needs to be normalized to produce smoothing with the same value of non-vectorized implementation:

$$w[i] \leftarrow \frac{w[i]}{\sum_{j=0}^{N-1} w[j]}$$

The normalized kernel is then convolved with the dF/F signal,  $d$ :

$$c[k] = \sum_{i=0}^{N-1} w[i] \cdot d[k-i]$$

This convolved signal,  $c$  is then normalized to the cumulative sum of the exponential kernel:

$$n[j] = \sum_{i=0}^j w[i]$$

$$emwa = c[i]/n[i]$$

## Acknowledgements

The development of this software was supported by funds from the Canadian Institutes of Health Research (CIHR) Foundation Award (FDN-148468).

## References