PyNeuroTrace - Python code for neural timeseries

05 May 2024

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 increasingly larger as new technologies allow for faster acquisition rates of raw data. This raw data is sourced from an ever-expanding library of fluorescent indicators encoding calcium, voltage, neurotransmitter, and neuromodulator activity. These diverse signals generated from these fluorescent bioindicators contain information on the underlying neuronal activity. Still, processing these signals is non-trivial, as each indicator has unique molecular kinetics and inherent signal-to-noise ratios. The signals are also acquired with different instruments, which differ in sensitivity and acquisition rate, all of which must be considered during signal processing. The development of pyNeuroTrace, an open-source Python library, was made to aid in processing these neuronal signals, which must be filtered with these unique aspects in mind before analysis of the underlying neuronal activity can be completed.

Statement of need

Many neuroscience labs that use optophysiological methods, such as two-photon microscopy or fiber photometry, frequently must rewrite and maintain standard functions and filters needed to analyze the raw recordings from experiments. Furthermore, many techniques and algorithms for signal processing are scattered throughout the literature and are frequently implemented in programming languages other than Python. `pyNeuroTrace` meets the need of a time series analysis package 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 recordings in 2D arrays. Additionally, with the increase in acquisition rates of new imaging techniques, we have implemented a subset of these algorithms using GPU-compatible code to increase the speed at which the methods can process larger datasets collected at kilohertz rates.

Signal Processing

DeltaF/F

There are several methods for calculating the change of intensity of a fluorescent trace (Grienberger et al. 2022). We implemented the method described by Jia *et al* for the calculation of $\Delta F/F$, which normalizes the signal to a baseline, helping with bleaching or other changes that occur over time, influencing the detection or magnitude of events in the raw signal(Jia et al. 2010). This implementation includes several smoothing steps to help with shot noise(Jia et al. 2010). In short, F_θ is calculated by finding the minimum signal in a window of the rolling average of the raw signal. Then ΔF is calculated by the difference in the raw signal and F_θ, which is then divided by F_θ to get the trace for $\Delta F/F_\theta$. This $\Delta F/F_\theta$ signal can be optionally smoothed using an exponentially weighted moving average (EWMA) to remove shot noise. Jia *et al* defined their rolling average with the following equation:

$$\bar{F} = \left(\frac{1}{\tau_1}\right) \int_{x-\tau_1/2}^{x+\tau_1/2} F(\tau)\, d\tau$$

The variable F_θ is defined using a second time constant, τ_2, that defines a rolling window to search for the minimum smoothed signal value to be used as a baseline:

$$F_\theta(t) = min(\bar{F}(x))|t - \tau_2 < x < t$$

Thus $\Delta F/F$ is where F is the original raw signal:

$$\Delta F/F = \frac{F(t) - F_\theta}{F_\theta}$$

The two time constants, τ_1 and τ_2, can be selected by users. Modifying these parameters will have a dramatic influence on the output signal.

Okada Filter

We implement the Okada Filter in Python(Okada, Ishikawa, and Ikegaya 2016). 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. This filter is defined by Okada *et al* as:

$$x_t \leftarrow x_t + \frac{x_{t-1} + x_{t+1} - 2x_t}{2(1 + e^{-\alpha(x_t - x_{t-1})(x_t - x_{t+1})})}$$

In this equation x_t is the value in the neural activity trace at time t. The value for α, which is a coefficient, should be selected so that the product of $x_t - x_t$

and $x_t - x_{t+1}$ causes a sufficiently steep sigmoid curve which functions a binary filter in the equation. This function is equivalent to the following conditional states from Okada *et al*:

$$\text{If } (x_t - x_{t-1})(x_t - x_{t+1}) \leq 0$$
$$x_t \leftarrow x_t$$
$$\text{If } (x_t - x_{t-1})(x_t - x_{t+1}) > 0$$
$$x_t \leftarrow \frac{x_{t-1} + x_{t+1}}{2}$$

Essentially, the Okada filter replaces the point, x_t, in a trace with the average of adjacent values when the product of the differences in adjacent values is greater than one. One useful characteristic of this smoothing algorithm is that it does not move the start position of events like other algorithms do (Okada, Ishikawa, and Ikegaya 2016)

Nonnegative Deconvolution

`pyNeuroTrace` also has an implementation of nonnegative deconvolution (NND) to be applied to photocurrents to reduce noise in raw time series recordings (Podgorski and Haas 2012). Another application of this algorithm is its use in event detection in neuronal activity traces, which, due to biosensor kinetics, follow similar decays as photocurrents from detectors (Podgorski and Haas 2012). This can be particularly useful for finding smaller magnitude events in fluorescent imaging that are often obfuscated by machine noise Podgorski and Haas (2012).

Event Detection

The event detection module uses several strategies to identify neuronal activity in time series datasets. These methodologies have been previously discussed and compared by Sakaki *et al* (Sakaki et al. 2018). These include two generalizable methods and one that requires prior knowledge of recorded event shape. The generalizable methods include filtering the signal through an exponentially weighted moving average (ewma) or cumulative sum of movement above the mean (cusum). The final filter is a matched filter that finds the probability of the trace matching a previously defined shape, such as one described by an exponential rise and decay of calcium signal generated by a GECI.

Visualization

`pyNeuroTrace` has several built-in visualization tools depending on the format of the data. 2D arrays of neuronal timeseries can be displayed as heat maps Figure 1 or as individual traces Figure 2. The heatmap is a useful visualization

tool for inspecting many traces at once; additionally, at the bottom of the plot, the stimuli timing is displayed if provided Figure 1. This functionality allows for quick visual inspection of activity from a population of neurons or signals sampled across a neuronal structure, such as a dendritic arbor.

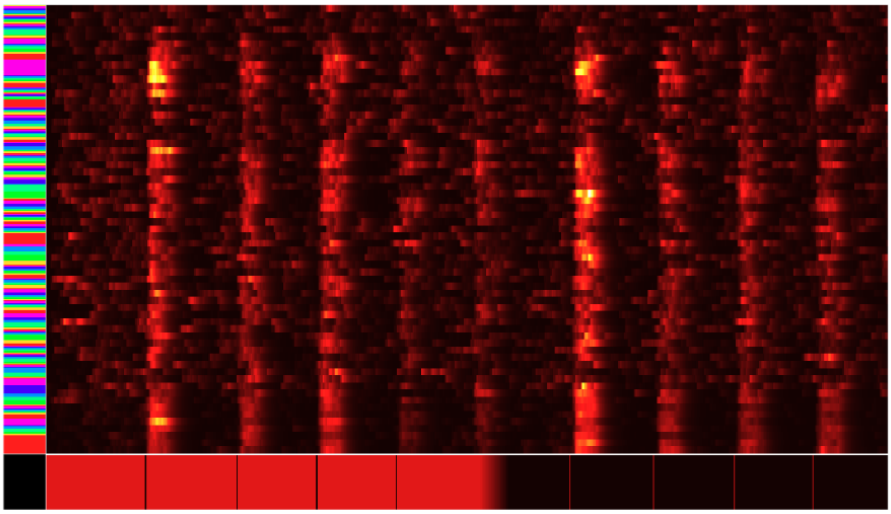

Figure 1: An example of a heatmap generated by pyNeuroTrace

For individual activity traces or small numbers of traces, 'pyNeuroTrace' has a line plot feature Figure 2. This is an ideal option for inspecting the shape of events, which may be difficult to appreciate from the colormaps in the heatmap visualization. Dotted lines are plotted vertically across the traces of neural activity to indicate when stimulus presentation occurred during an experiment.

One of these in-built visualizations is specific to the data structure generated by a custom acousto-optic random access microscope (Sakaki et al. 2020) Figure 3. This microscope uses acousto-optic deflectors (AODs) to perform inertia-free scanning between preselected points of interest, allowing for extremely fast acquisition rates for sampling neuronal activity. The scan engine of the microscope allows for random access imaging that is used to image the activity across the morphology of a single neuron (Sakaki et al. 2020). This type of imaging does not generate a traditional image. The microscope instead links acquired neuronal traces to points of interest organized into a hierarchical tree structure representing the neuronal morphology in a complex data file.

GPU Acceleration

Several of the filters in `pyNeuroTrace` have been rewritten to be almost entirely vectorized in their calculations. The benefit is more noticeable when

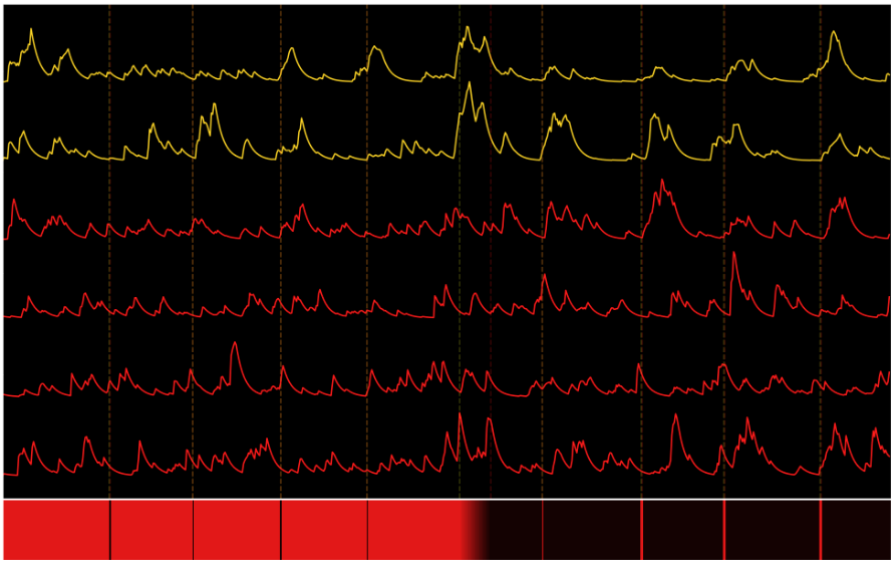

Figure 2: Six GCaMP7f calcium traces plotted with stimulus times below

comparing the difference in performance while using a longer time series or one acquired with faster acquisition rates. These vectorized implementations gain further speed by being executed on a GPU using the Cupy Python library (Okuta et al. 2017). The GPU-accelerated filters can be imported from the `pyneurotrace.gpu.filters` module, and a CUDA-compatible graphics card is required for their execution. This functionality is becoming increasingly crucial as acquisition rates increase for kilohertz imaging of activity, which can generate arrays of hundreds of thousands of data points in length in just a few minutes. Figure 4 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 an Intel i5-9600K with six 4.600GHz cores; the GPU was an NVIDIA GeForce RTX 4070 with CUDA Version 12.3.

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 exponetional function. The kernel used to perform this is defined as:

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

Where α is defined as:

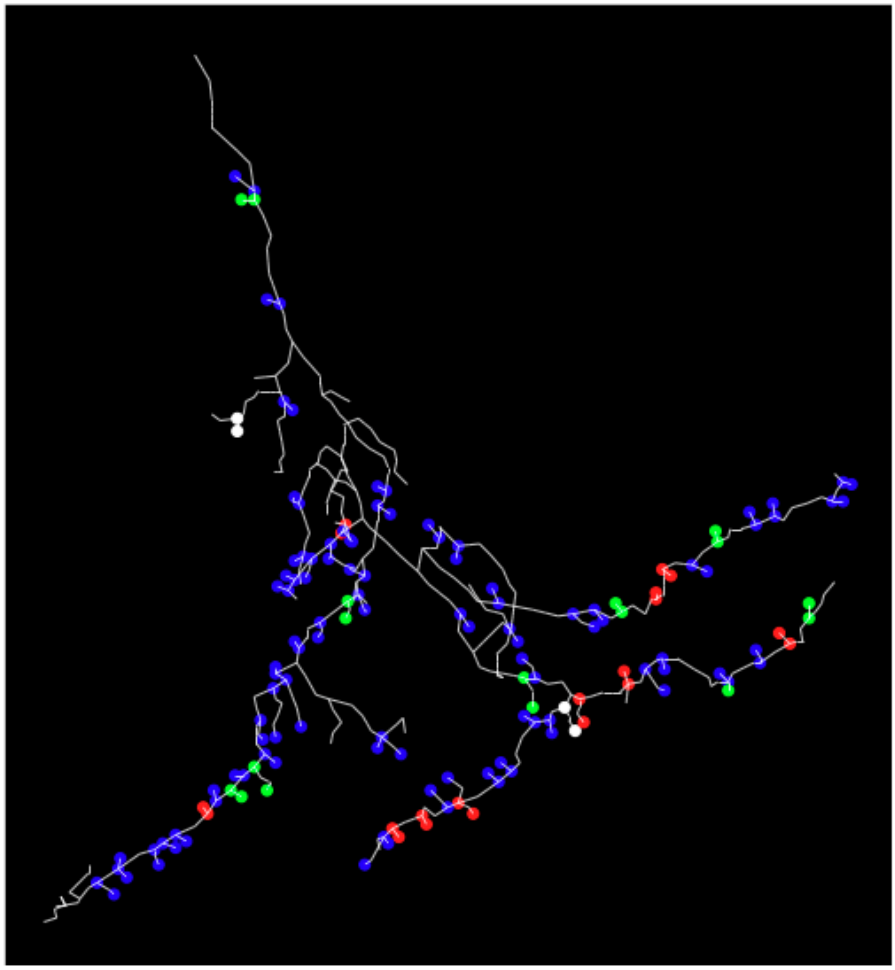

Figure 3: An example of a lab-specific visualisation of points of interest across a dendritic arbor imaged with an AOD microscope

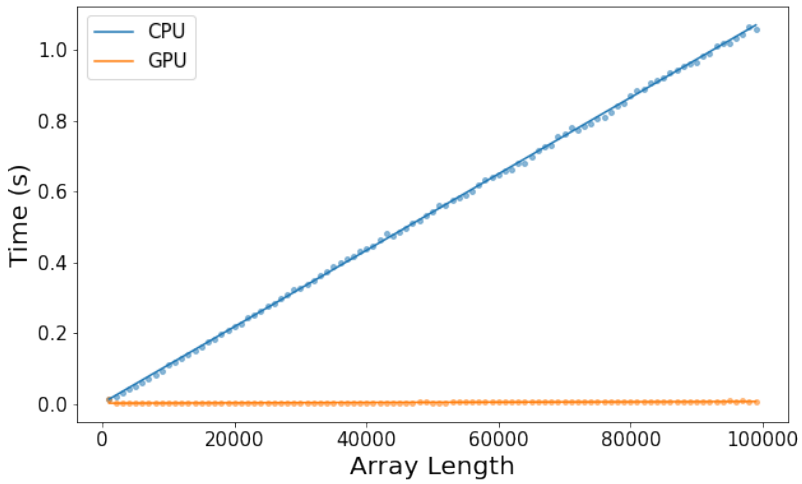

Figure 4: Comparison between $\Delta F/F$ with EWMA calculations for different array sizes using either the CPU (blue) or GPU (orange).

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

τ is a user-selected time constant in seconds, which is translated into the number of samples using the acquisition rate used to acquire the data. N is a window parameter for the kernal calcuated using α:

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

This filters for smaller values that have a minuscule influence on the weighted average. The kernel needs to be normalized to produce smoothing with the same output value as the non-vectorized impementation:

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

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:

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

$$emwa = \frac{c[i]}{n[i]}$$

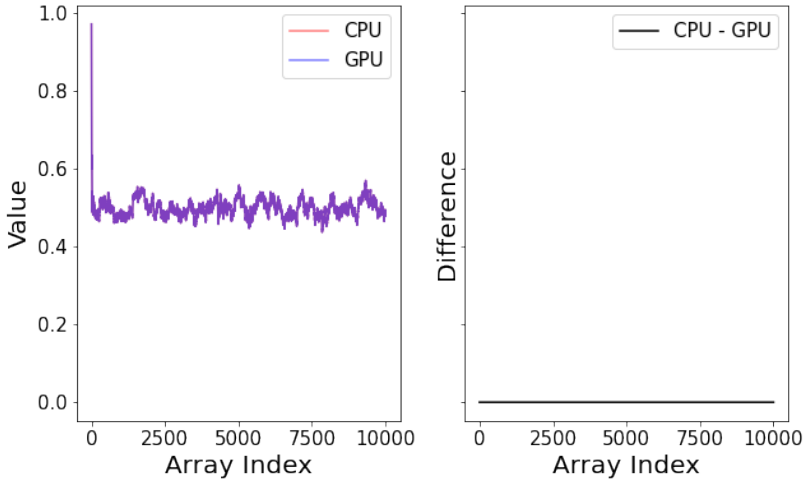

Figure 5: Overlay of the EWMA calculations using the CPU implementation and GPU approximation in red and blue. The difference in values from the output is also plotted.

To demonstrate the differences between the CPU and GPU implementations of the EWMA calculations were performed on an array of random values Figure 5. These were generated from the same array using the respective decays for either implementation using the time constant of 50 milliseconds and a sampling rate of 2kHz. Depending on user parameters, the difference between the two outputs typically ranges in magnitude from 1e-16 to 1e-12. These discrepancies can also be attributed to differences in floating-point number accuracy between CPU and GPU calculations.

Acknowledgements

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

References

Grienberger, Christine, Andrea Giovannucci, William Zeiger, and Carlos Portera-Cailliau. 2022. "Two-Photon Calcium Imaging of Neuronal Activity." *Nature Reviews Methods Primers* 2 (1). https://doi.org/10.1038/s43586-022-00147-1.

Jia, Hongbo, Nathalie L Rochefort, Xiaowei Chen, and Arthur Konnerth. 2010. "In Vivo Two-Photon Imaging of Sensory-Evoked Dendritic Calcium Signals in Cortical Neurons." *Nature Protocols* 6 (1): 28–35. https://doi.org/10.1038/nprot.2010.169.

Okada, Mami, Tomoe Ishikawa, and Yuji Ikegaya. 2016. "A Computationally Efficient Filter for Reducing Shot Noise in Low s/n Data." *PLOS ONE* 11 (June): e0157595. https://doi.org/10.1371/journal.pone.0157595.

Okuta, Ryosuke, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. 2017. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations." In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in the Thirty-First Annual Conference on Neural Information Processing Systems (NIPS)*. http://learningsys.org/nips17/assets/papers/paper_16.pdf.

Podgorski, Kaspar, and Kurt Haas. 2012. "Fast Non-negative Temporal Deconvolution for Laser Scanning Microscopy." *Journal of Biophotonics* 6 (2): 153–62. https://doi.org/10.1002/jbio.201100133.

Sakaki, Kelly D. R., Patrick Coleman, Tristan Dellazizzo Toth, Claire Guerrier, and Kurt Haas. 2018. "Automating Event-Detection of Brain Neuron Synaptic Activity and Action Potential Firing in Vivo Using a Random-Access Multiphoton Laser Scanning Microscope for Real-Time Analysis." In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE. https://doi.org/10.1109/embc.2018.8512983.

Sakaki, Kelly D. R., Kaspar Podgorski, Tristan A. Dellazizzo Toth, Patrick Coleman, and Kurt Haas. 2020. "Comprehensive Imaging of Sensory-Evoked Activity of Entire Neurons Within the Awake Developing Brain Using Ultrafast AOD-Based Random-Access Two-Photon Microscopy." *Frontiers in Neural Circuits* 14 (June). https://doi.org/10.3389/fncir.2020.00033.