PySpike—A Python library for analyzing spike train synchrony

Mario Mulansky[*], Thomas Kreuz

Institute for Complex Systems, CNR, Via Madonna del Piano 10 – 50019 Sesto Fiorentino, Italy

Abstract

Understanding how the brain functions is one of the biggest challenges of our time. The analysis of experimentally recorded neural firing patterns (spike trains) plays a crucial role in addressing this problem. Here, the PySpike library is introduced, a Python package for spike train analysis providing parameter-free and time-scale independent measures of spike train synchrony. It allows to compute similarity and dissimilarity profiles, averaged values and distance matrices. Although mainly focusing on neuroscience, PySpike can also be applied in other contexts like climate research or social sciences. The package is available as Open Source on Github and PyPI.

© 2016 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Code metadata

Current code version	v0.5.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00032
Legal Code License	BSD License
Code versioning system used	git
Software code languages, tools, and services used	Python, Cython
Compilation requirements, operating environments & dependencies	`numpy`, `cython`, `matplotlib`, `nosetests`
If available Link to developer documentation/manual	http://www.pyspike.de
Support email for questions	mario.mulansky@gmx.net

1. Introduction

Gaining insight into the inner workings of the brain remains a largely unsolved challenge that requires combined efforts of biophysics, medicine, experimental as well as computational neuroscience [1]. The basis for scientific advancement in this field are experimental recordings of neural activity usually represented in terms of spike trains, i.e. lists of spike times for each recorded neuron. With sophisticated modern recording techniques, it is now possible to perform highly parallel measurements of neural activity, typically resulting in very large sets of spike trains [2,3]. This generates an increased demand for powerful and high quality data analysis tools that are capable of processing large datasets as produced by parallel recordings.

There exist numerous methods to analyze spike train data, e.g. based on spike count distributions, interspike intervals or exact spike times. One very important approach is to quantify the synchrony between spike trains. In the past decades several synchrony measures have been proposed [4–6] which have already been used, among others, to quantify the reliability of neuronal responses [7], to analyze the role of spike synchronization in feature binding [8], and to distinguish different stimuli in the context of neuronal coding [1].

The PySpike library[1] introduced here (logo shown in Fig. 1) is a Python package that allows one to compute two

* Corresponding author.
  E-mail addresses: mario.mulansky@isc.cnr.it (M. Mulansky), thomas.kreuz@cnr.it (T. Kreuz).

1 www.pyspike.de.

Fig. 1. Logo of the PySpike library.

different dissimilarity measures, the ISI-distance [9], the SPIKE-distance [10] and additionally the similarity measure SPIKE-Synchronization [11,12]. Each of these three methods is *time-resolved*, *parameter-free* and *time-scale independent* and therefore highly versatile. Being time-resolved, for example, means these measure can detect changes in synchrony over time, while being parameter-free makes them readily applicable with unambiguous results, as no parameter optimization is required. These measures have already been applied in many experimental studies in the past, for example [13–16].

Although developed with a neuroscientific context in mind, the synchrony measures discussed here can be applied to any form of discrete time series consisting of event sequences of any kind. In fact, such measures have already been utilized in several other research areas, such as climate research [17] or social sciences [18,19].

PySpike is a library aimed to perform automated data analysis with Python scripts. It is therefore a complementary approach to the SPIKY software package,[2] a Matlab framework for spike train analysis providing a similar functionality but additionally offering a sophisticated GUI [11,20]. Several other software packages for spike train analysis have been developed in the recent past, notably SyncPy[3] [21], a Python based GUI for quantifying synchrony in time series. However, it currently does not include the synchrony measures implemented in PySpike. A C++ implementation of the spike train distance measures[4] was presented in [22], but it is based on sampled data and therefore of substantially inferior performance [11]. Finally, a comprehensive collection of scientific software for spike train analysis is also provided as part of [23], aiming specifically at multivariate recordings.[5]

2. Spike train distances

Here, discrete time series are represented by *spike trains*, sequences of time points denoting the occurrence of an event (spike) at those time points: $s = \{t_1, t_2, t_3, \ldots\}$. Generally, a time-resolved distance measure maps a pair of spike trains s_1, s_2 onto a profile $\{s_1, s_2\} \rightarrow S(t)$ with $0 \leq S(t) \leq 1$. The overall distance value can easily be obtained by integration: $D_S = \int S(t) \, dt$.

PySpike provides three such distance measures: ISI-distance, SPIKE-distance and SPIKE-Synchronization. These methods are sensitive to different aspects of spike train

synchrony (interspike intervals, exact spike timings, spike matching, respectively). Hence, the choice of method should be informed by assumptions on how information is encoded in the spike trains. In the following, we give a brief introduction to the measures provided in the PySpike library. For a detailed discussion of the methods and their properties see Appendix A and [12].

The **ISI-distance** profile $I(t)$, introduced in [9], quantifies dissimilarity in terms of the relative differences of the concurrent interspike intervals of the two spike trains. Essentially, it measures the relative differences of the instantaneous rates of the two spike trains, but it is not sensitive to exact spike timings. The ISI-distance profile is a bivariate piecewise constant function.

The **SPIKE-distance** profile $S(t)$, first introduced in [24] and refined in [10], represents a dissimilarity profile based on exact spike timings. Thus, the SPIKE-distance quantifies spike train dissimilarity in terms of deviations from exact coincidences of spikes in the two spike trains. This results in a bivariate piecewise linear profile.

While the fundamental definition of both the ISI- and the SPIKE-distance profile is bivariate (distance profile of two spike trains), the generalization to a multivariate context is a straightforward average over all spike train pairs [25].

SPIKE-Synchronization [11,12] is a straight-forward, normalized coincidence counter with an adaptive coincidence window. It quantifies similarity in terms of the fraction of coincidences between two spike trains and hence is a very intuitive measure. The generalization of SPIKE-Synchronization for many spike trains can be defined based on all spike train pairs leading to a consistent multivariate framework with similarity again quantified as the overall fraction of coincidences in all spike trains [11,12].

3. Package structure

3.1. The spike train

The central data structure of the PySpike library is the `SpikeTrain`, a Python class representing an individual spike train. This class contains the (sorted) spike times as a `numpy.array` as well as the start and end time of the spike train. Such `SpikeTrain` objects can either be created directly by providing the spike times, generated randomly from a Poisson process using the `generate_poisson_spikes` function, imported from text files via `load_spike_trains_from_txt` or imported from time series via `import_spike_trains_from_time_series`. These objects then serve as input to calculate the distance measures, cf. Fig. 2.

3.2. Computing profiles

Being time-resolved is a main advantage of the three spike train synchrony measures discussed here. Hence, PySpike contains functionality to compute synchrony profiles: `isi_profile` computes the piecewise constant ISI-profile $I(t)$, `spike_profile` returns the piecewise linear SPIKE-profile $S(t)$ and `spike_sync_profile` yields the discrete

2 http://www.fi.isc.cnr.it/users/thomas.kreuz/Source-Code/SPIKY.html.

3 https://github.com/syncpy/SyncPy.

4 https://github.com/modulus-metric.
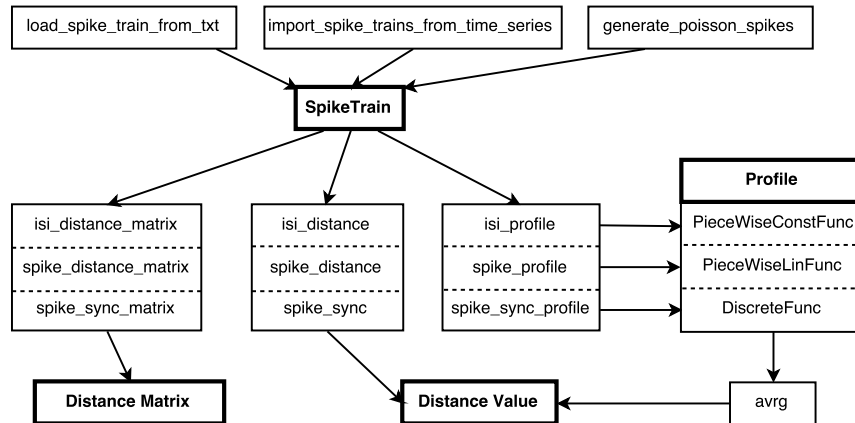
5 http://spiketrain-analysis.org/software.

Fig. 2. Structure of the PySpike package. The `SpikeTrain` is the central class for which several functions are provided to compute the dissimilarity or similarity-profiles, as well as distance/similarity values and matrices.

Listing 1: Using profile and distance functions.

```
# generate three arbitrary spike trains
# with start/end time of 0 and 4
st1 = SpikeTrain([1.0,2.0,3.0], edges=[0,4])
st2 = SpikeTrain([0.5,3.0,3.5], edges=[0,4])
st3 = SpikeTrain([2.5,3.8], edges=[0,4])

# bivariate profile of spike trains 1,2:
isi_prof = isi_profile(st1, st2)

# distance value as average over profile
isi_dist = isi_prof.avrg()

# faster: directly from the spike trains
isi_dist = isi_distance(st1, st2)

# profile of 3 spike trains:
spike_prof = spike_profile([st1, st2, st3])

# computing pairwise spike sync matrix
# of the whole list of spike trains:
spike_trains = [st1, st2, st3]

M = spike_sync_matrix(spike_trains)
# M is a 3x3 numpy array
```

SPIKE-Synchronization profile C_k. As the three profiles have different mathematical properties, they are represented by different Python classes in PySpike: `PieceWiseConstFunc`, `PieceWiseLinFunc` and `DiscreteFunc`, as seen in Fig. 2. By introducing specific data structures for each of these objects, PySpike can utilize their mathematical properties and use a highly efficient implementation while still providing a convenient user interface. All three function classes provide an `avrg` member function that returns the total (time-averaged) distance of the respective measure. In Listing 1 several profiles are computed and averaged. Furthermore, the profiles offer a `get_plottable_data` member function for easy visualization of the profile, shown exemplarily in Listing 2.

3.3. Computing distances and distance matrices

Besides calculating the time-resolved profiles, PySpike also provides functionality to directly compute the overall distance values for the ISI-distance and the SPIKE-distance as well as the overall SPIKE-Synchronization value. The respective functions are `isi_distance`, `spike_distance` and `spike_sync`, see Listing 1 for an example on how to compute profiles and Fig. 3 for example profile plots. Although the overall distances can also be computed from the profiles by using the `avrg()` member function, the specific distance functions allow for a more efficient implementation and hence a significantly better performance, as will be discussed in Section 4.2.

When analyzing sets of spike trains, one might not only be interested in the multivariate distance of the whole set, but also in the pairwise distances between all spike trains. For this purpose, PySpike offers the `isi_distance_matrix`, `spike_distance_matrix` and the `spike_sync_matrix` functions. Listing 1 shows an example on how to compute the SPIKE-Synchronization matrix. As all distance measures in PySpike are symmetric, the distance matrices are also always symmetric.

Finally, note that the distance and distance matrix functions can also be used to compute selective averages, that is averages over specified intervals of the profiles. This is accomplished by providing the (optional) named parameter `interval` to the distance functions, and similarly for the `avrg` member function of the profiles. An example for computing and plotting such a selective distance matrix is given in Listing 2. This allows for a very detailed analysis of the profiles, for example by comparing the average distances before and after repeated stimuli or triggers. Fig. 4 shows an example of a changing synchrony pattern revealed by distance matrices with different averaging intervals.

4. Implementation and performance

4.1. Code base

PySpike is written following modern coding standards and best practices in scientific computing [26], specifically adhering to the official Style Guide for Python Code (PEP8).[6] It is

---

[6] https://www.python.org/dev/peps/pep-0008/.

Listing 2: Load spike trains and plot profile and distance matrix.

```
# load spike train data
spike_trains = load_spike_trains_from_txt(
                 "spike_trains.txt",
                 edges=(0, 4000))

# compute and plot ISI profile
isi_prof = isi_profile(spike_trains)
x, y = isi_prof.get_plottable_data()
plot(x, y, '-k')

# plot SPIKE distance matrix
# with selective averaging t=0...1000
spike_mat = spike_distance_matrix(
                 spike_trains,
                 interval=[0, 1000])
imshow(spike_mat)
```
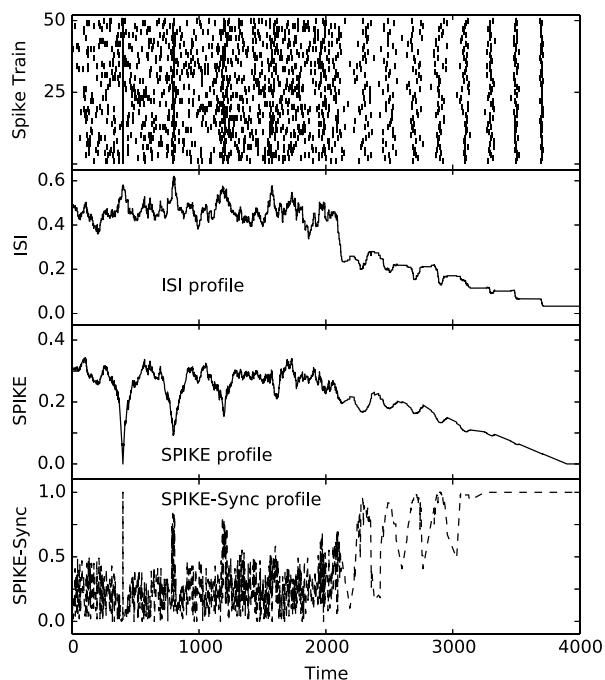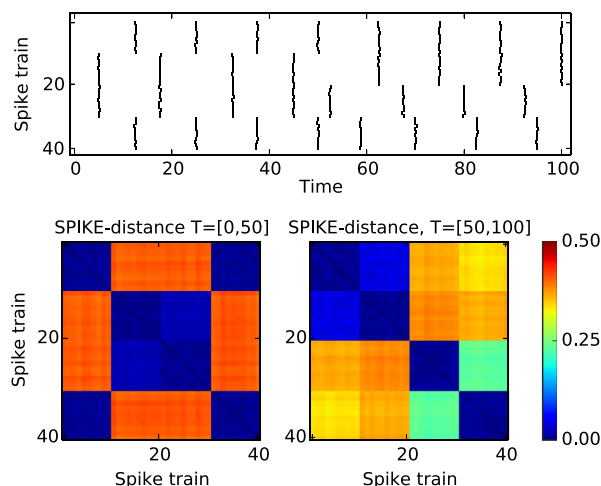

Fig. 4. Spike-distance matrices for a set of $M = 40$ artificial spike trains (top panel) with changing synchronous firing. The spike trains are divided into four groups, which fire in two different synchrony patterns in the first half and second half of the observation interval. This change of synchrony structure is clearly captured by using selective averaging for the SPIKE-distance matrices ($T = [0, 50]$ on the left and $T = [50, 100]$ on the right).

The PySpike library is developed on Github[7] with periodic releases on PyPI.[8] It is fully unit tested using the `nosetest` framework,[9] including an extensive test matrix on a continuous integration server,[10] where all tests are performed on each commit using Python versions 2.6, 2.7, 3.3, 3.4, 3.5, and for both the Python and the Cython backend.

4.2. Performance

With the increasing experimental abilities in neuroscience it is now possible to obtain parallel recordings of thousands and more spike trains in both cultured neurons [30] and *in-vivo* recordings [2]. This requires an efficient implementation of spike train analysis tools that allow one to process such huge datasets within acceptable times. However, Python is known for its very poor performance compared to low-level languages such as C/C++. We therefore designed the PySpike library to consist of two parts: (1) A front-end implemented fully in Python representing the interface of the PySpike functionality to the library users; and (2) A backend providing the numeric implementation of the distance measures. For the backend, PySpike provides two versions, a pure Python implementation and a much faster C implementation based on Cython. The Cython backend is the default choice, and only if Cython is not available PySpike falls back to the Python backend.

Utilizing Cython (and therefore the speed of C), we were able to gain performance improvements of more than a factor of 200. Fig. 5 shows the runtime of a multivariate analysis of $M = 1000$ Poissonian spike trains each containing $N \approx 500$ spikes, representative of a realistic experimental situation.

Fig. 3. Multivariate ISI, SPIKE and SPIKE-Synchronization profiles for $M = 50$ spike trains (shown in top panel). In this example, artificially generated spike trains are used where the first half consists of noisy spiking superimposed with a few synchronous events with more and more jitter. The second half contains increasingly synchronized events without the noisy background.
Source: Adapted from [12].

compatible with both the Python2 and Python3 runtime and available as open source software distributed under the BSD License. The frontend is entirely implemented in Python, whereas for the crucial computations a Cython [27] version is provided for maximal performance, as shown in Section 4.2 below. As most scientific Python libraries, PySpike requires NumPy as `numpy` arrays [28] are the underlying data structures. Furthermore, it is designed to interact with `matplotlib` for the plotting of profiles and distance matrices and it can also be easily combined with other scientific libraries such as `scikit-learn` for cluster analysis [29].

7 https://github.com/mariomulansky/PySpike/.

8 https://pypi.python.org/pypi/pyspike/.

9 https://nose.readthedocs.org/en/latest.
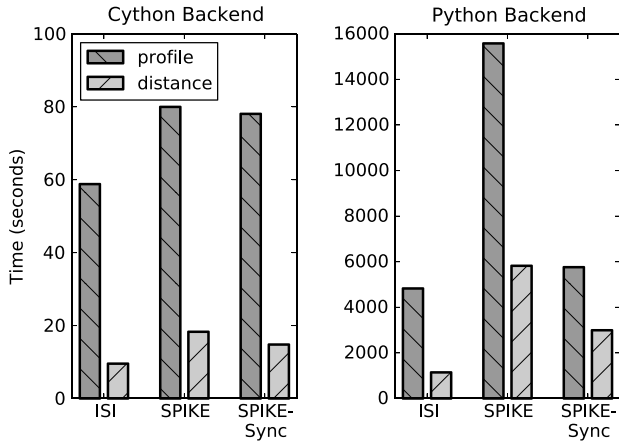
10 https://travis-ci.org/mariomulansky/PySpike.

Fig. 5. Runtime for computing the multivariate ISI-distance, SPIKE-distance and SPIKE-Synchronization for a set of $M = 1000$ spike trains each containing $N = 500$ spikes on average. Dark gray bars represent the runtime for first computing profiles and then averaging, while light gray bars indicate the direct computation of the distance values. The left panel shows the runtime (in seconds) for the Cython backend, while the right panel presents the runtime for the Python backend.

For both backends, Cython (left) and Python (right), we show the runtime of a full multivariate computation of the ISI-distance, SPIKE-distance and SPIKE-Synchronization for first computing the profile and then averaging, as well as for the immediate distance functions. All performance measurements were done on an Intel Core i5-3210M CPU @ 2.50 GHz. First, note that with the Python backend, the computation time reaches several hours, which is clearly beyond any tolerable runtime for a data analysis procedure. Using the Cython backend, however, the computations require only 10–20 s (distance) or around one minute (profiles), which we believe are acceptable runtimes for a data analysis of this size. Second, as mentioned earlier, the direct computations of the overall distance values (light gray in Fig. 5) are significantly faster than first computing the profile and then averaging (dark gray), which is the reason why PySpike provides this functionality.

While Fig. 5 only shows the results for one size of dataset, namely $M = 1000$ spike trains with each on average containing $N \approx 500$ spikes, it is known that the runtime scales linearly with the number of spikes N and quadratically with the number of spike trains M (as the number of pairs grows $\sim M^2$). Hence, the expected runtime for larger (or smaller) datasets can easily be estimated from the results in Fig. 5.

5. Conclusions and outlook

We presented PySpike, a Python library for measuring synchrony in experimental and simulated spike train data. PySpike provides three parameter-free, time-scale independent and multivariate synchrony measures as well as facilities for plotting, Poisson spike train generation and selective averaging. It is implemented in a clean, fully documented code base that follows official Python code style guidelines. Furthermore, PySpike's development utilizes advanced software engineering techniques such as version control, unit testing and continuous integration. By virtue of a Cython backend, it also shows excellent performance and hence is suitable for the analysis of large datasets.

Besides regular maintenance and bug fixing, future work on PySpike will include the addition of new synchrony measures specifically designed for spike trains with bursts [31]; support for event detection (transformation from continuous to discrete data); and extended functionality of spike train generation. Current work is focused on the development of a spike order indicator that allows to analyze propagation patterns by quantifying temporal leader/follower properties within sets of spike trains [32]. A preliminary Python implementation already exists and will become part of an official PySpike release in the near future.

In conclusion, we are convinced that the PySpike library is a helpful software package for spike train analysis. It provides a clean, well documented and consistent Python interface to compute measures of spike train synchrony, while offering excellent performance.

Acknowledgments

Appendix. Mathematical definitions

A.1. ISI-distance

With $\{t_i^{(1)}\}$ being the spike times of the first spike train, its interspike intervals are given as $v_i^{(1)} = t_{i+1}^{(1)} - t_i^{(1)}$ and similarly for the second spike train (cf. Fig. A.6) [9]. The ISI-profile is then computed as the normalized absolute difference of these interspike intervals:

$$I(t) = \frac{|v^{(1)}(t) - v^{(2)}(t)|}{\max\{v^{(1)}(t), v^{(2)}(t)\}}, \quad t \in [T_0, T_1], \tag{A.1}$$

where $T_0 \leq t_i^{(1,2)} \leq T_1$ are the edges of the spike trains.

A.2. SPIKE-distance

The computation of $S(t)$ is based on the four corner spikes surrounding the current time t: the preceding spikes $t_P^{(1),(2)}(t)$ and the following spikes $t_F^{(1),(2)}(t)$ of each spike train (cf. Fig. A.6) [10]. For each of the corner spikes, the distance to the closest spike of the *other spike train* is computed, e.g.:

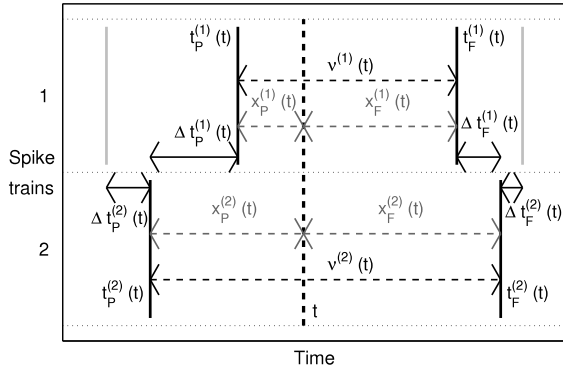$$\Delta t_P^{(1)}(t) = \min_i\{|t_P^{(1)} - t_i^{(2)}|\}, \tag{A.2}$$

Fig. A.6. Local definitions of interspike intervals and time differences required for the calculation of the ISI- and the SPIKE-profile.

and similarly for $\Delta t_P^{(2)}$ and $\Delta t_F^{(1),(2)}$. These distances are then combined by a linear interpolation according to the current time point t resulting in the following quantity:

$$S_1(t) = \frac{\Delta t_P^{(1)}(t) x_F^{(1)}(t) + \Delta t_F^{(1)}(t) x_P^{(1)}}{\nu^{(1)}(t)}, \qquad (A.3)$$

and similarly $S_2(t)$ is defined for the second spike train. Based on these quantities, we finally arrive at the definition of the SPIKE-profile using the following normalization:

$$S(t) = \frac{S_1(t)\nu^{(2)}(t) + S_2(t)\nu^{(1)}(t)}{\frac{1}{2}(\nu^{(1)}(t) + \nu^{(2)}(t))^2}, \qquad t \in [0, T]. \qquad (A.4)$$

Note, that this is a piecewise linear function as $S_{1,2}$ are piecewise linear while the other terms are piecewise constant.

A.3. SPIKE-Synchronization

For SPIKE-Synchronization, a coincidence indicator $C_i^{(1),(2)}$ is defined for every spike of the two spike trains $s^{(1),(2)}$, where $C_i = 1$ if the spike at t_i is part of a coincidence, and $C_i = 0$ if not [11]. A coincidence is defined in terms of an *adaptive* coincidence window τ according to the local firing rate:

$$\tau_{ij}^{(1,2)} = \frac{1}{2} \min\{\nu_i^{(1)}, \nu_{i-1}^{(1)}, \nu_j^{(2)}, \nu_{j-1}^{(2)}\}, \qquad (A.5)$$

with $\nu^{(1),(2)}$ being the interspike intervals as above. The value of the coincidence indicator is then given by:

$$C_i^{(1)} = \begin{cases} 1 & \text{if } \min_j(|t_i^{(1)} - t_j^{(2)}|) < \tau_{ij}^{(1,2)} \\ 0 & \text{otherwise.} \end{cases} \qquad (A.6)$$

In the same way, the coincidence indicator for the second spike train $C_i^{(2)}$ is computed.

The SPIKE-Synchronization profile is then given by the discrete function defined from the pooled coincidence indicators $\{C_k\} = \{C_i^{(1)}\} \cup \{C_i^{(2)}\}$ and spike times: $\{t_k'\} = \{t_i^{(1)}\} \cup \{t_i^{(2)}\}$. Since integrals are not defined for discrete functions, the overall SPIKE-Synchronization value SYNC is obtained via summation:

$$\text{SYNC} = \frac{1}{M} \sum_{k=1}^{M} C_k = \frac{C}{M}, \qquad (A.7)$$

where M is the total number of spikes in the pooled spike train $\{t_k'\}$ and C denotes the total number of coincident spikes. In contrast to the other two measures, SPIKE-Synchronization quantifies similarity rather than distance.

References

[1] Quian Quiroga R, Panzeri S. Principles of neural coding. Boca Raton, FL, USA: CRC Taylor and Francis; 2013.
[2] Alivisatos AP, et al. Nanotools for neuroscience and brain activity mapping. ACS Nano 2013;7(3):1850.
[3] Marblestone AH. et al. Physical principles for scalable neural recording, Front Comput Neurosci 7 (137).
[4] Victor JD, Purpura KP. Nature and precision of temporal coding in visual cortex: A metric-space analysis. J Neurophysiol 1996;76:1310.
[5] van Rossum MCW. A novel spike distance. Neural Comput 2001;13:751.
[6] Quian Quiroga R, Kreuz T, Grassberger P. Event synchronization: A simple and fast method to measure synchronicity and time delay patterns. Phys. Rev. E 2002;66:041904.
[7] Tiesinga PHE, Fellous JM, Sejnowski TJ. Regulation of spike timing in visual cortical circuits. Nat Rev Neurosci 2008;9:97.
[8] Singer W. Distributed processing and temporal codes in neuronal networks. Cogn Neurodyn 2009;3:189.
[9] Kreuz T, Mormann F, Andrzejak RG, Kraskov A, Lehnertz K, Grassberger P. Measuring synchronization in coupled model systems: A comparison of different approaches. Physica D 2007;225:29.
[10] Kreuz T, Chicharro D, Houghton C, Andrzejak RG, Mormann F. Monitoring spike train synchrony. J Neurophysiol 2013;109:1457.
[11] Kreuz T, Mulansky M, Bozanic N. SPIKY: A graphical user interface for monitoring spike train synchrony. J Neurophysiol 2015;113:3432.
[12] Mulansky M, Bozanic N, Sburlea A, Kreuz T. A guide to time-resolved and parameter-free measures of spike train synchrony. In: Proceedings of the international conference on event-based control, communication, and signal processing (EBCCSP). IEEE; 2015. p. 1.
[13] Dodla R, Wilson CJ. Asynchronous response of coupled pacemaker neurons. Phys Rev Lett 2009;102:068102.
[14] Ibarz JM, Foffani G, Cid E, Inostroza M, de la Prida LM. Emergent dynamics of fast ripples in the epileptic hippocampus. J Neurosci 2010; 30:16249.
[15] Wildie M, Shanahan M. Establishing communication between neuronal populations through competitive entrainment. Front Comput Neurosci 2012;5:62.
[16] Di Poppa M, Gutkin BS. Correlations in background activity control persistent state stability and allow execution of working memory tasks. Front Comput Neurosci 2013;7:139.
[17] Malik N, Marwan N, Kurths J. Spatial structures and directionalities in monsoonal precipitation over South Asia. Nonlinear Process Geophys 2010;17:371.
[18] Williams MJ, Whitaker RM, Allen SM. Measuring individual regularity in human visiting patterns. In: Proceedings of the ASE international conference on social computing; 2012. p. 117.
[19] Rabinowitch TC, Knafo-Noam A. Synchronous rhythmic interaction enhances children's perceived similarity and closeness towards each other. PLoS ONE 2015;10:e0120878.
[20] Bozanic N, Mulansky M, Kreuz T. SPIKY. Scholarpedia 2014;9(12): 32344. revision #152246.
[21] Varni G, Avril M, Usta A, Chetouani M. SyncPy: a unified open-source analytic library for synchrony. In: Proceedings of the 1st workshop on modeling interpersonal synchrony and influence. ACM; 2015. p. 41.
[22] Rusu CV, Florian RV. A new class of metrics for spike trains. Neural Comput 2014;26(2):306.
[23] Grün S, Rotter S. Analysis of parallel spike trains. Springer; 2010.
[24] Kreuz T, Chicharro D, Greschner M, Andrzejak RG. Time-resolved and time-scale adaptive measures of spike train synchrony. J Neurosci Methods 2011;195:92.
[25] Kreuz T, Chicharro D, Andrzejak RG, Haas JS, Abarbanel HDI. Measuring multiple spike train synchrony. J Neurosci Methods 2009;183: 287.
[26] Wilson G, et al. Best practices for scientific computing. PLoS Biol 2014; 12(1):1.

[27] Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn D, Smith K. Cython: The best of both worlds. Comput Sci Eng 2011;13(2):31.

[28] van der Walt S, Colbert SC, Varoquaux G. The numpy array: A structure for efficient numerical computation. Comput Sci Eng 2011;13(2):22.

[29] Pedregosa F, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825.

[30] Berdondini L, Imfeld K, Maccione A, Tedesco M, Neukom S, Koudelka-Hep M, Martinoia S. Active pixel sensor array for high spatio-temporal resolution electrophysiological recordings from single cell to large scale neuronal networks. Lab Chip 2009;9:2644.

[31] Satuvuori E, Mulansky M, Bozanic N, Kreuz T. in preparation; 2016.

[32] Kreuz T, Satuvuori E, Pofahl M, Mulansky M. in preparation; 2016.