pymuvr Documentation

Release 1.3.0

Eugenio Piasini

January 06, 2015

Contents

Author Eugenio Piasini

Contact e.piasini@ucl.ac.uk

Documentation http://pymuvr.readthedocs.org

Source code https://github.com/epiasini/pymuvr

PyPI entry https://pypi.python.org/pypi/pymuvr

A Python package for the fast calculation of Multi-unit Van Rossum neural spike train metrics, with the kernel-based algorithm described in Houghton and Kreuz, *On the efficient calculation of Van Rossum distances* (Network: Computation in Neural Systems, 2012, 23, 48-58). This package started out as a Python wrapping of the original C++ implementation given by the authors of the paper, and evolved from there with bugfixes and improvements.

Contents

1.1 Installation

1.1.1 Requirements

- CPython 2.7 or 3.x.
- NumPy>=1.7.
- C++ development tools and Standard Library (package *build-essential* on Debian).
- Python development tools (package *python-dev* on Debian).

1.1.2 Installing via *pip*

To install the latest release, run:

```
pip install pymuvr
```

1.1.3 Testing

From the root directory of the source distribution, run:

```
python setup.py test
```

(requires *setuptools*). **Alternatively**, if pymuvr is already installed on your system, look for the copy of the `test_pymuvr.py` script installed alongside the rest of the pymuvr files and execute it. For example:

```
python /usr/lib/pythonX.Y/site-packages/pymuvr/test/test_pymuvr.py
```

1.2 Mathematical methods

We define a compact formalism for multiunit spike-train metrics by opportunely characterising the space of multiunit feature vectors as a tensor product. Previous results from Houghton and Kreuz (2012, *On the efficient calculation of van Rossum distances*. Network: Computation in Neural Systems, 2012, 23, 48-58) on a clever formula for multiunit Van Rossum metrics are then re-derived within this framework, also fixing some errors in the original calculations.

1.2.1 A compact formalism for kernel-based multiunit spike train metrics

Consider a network with C cells. Let

$$\mathcal{U} = \left\{\boldsymbol{u}^1, \boldsymbol{u}^2, \ldots, \boldsymbol{u}^C\right\}$$

be an *observation of network activity*, where

$$\boldsymbol{u}^i = \left\{u_1^i, u_2^i, \ldots, u_{N_{\boldsymbol{u}^i}}^i\right\}$$

is the (ordered) set of times of the spikes emitted by cell i. Let $\mathcal{V} = \left\{\boldsymbol{v}^1, \boldsymbol{v}^2, \ldots, \boldsymbol{v}^C\right\}$ be another observation, different in general from $\mathcal{U}$.

To compute a kernel based multiunit distance between $\mathcal{U}$ and $\mathcal{V}$, we map them to the tensor product space $\mathcal{S} \doteq \mathbb{R}^C \bigotimes L_2(\mathbb{R} \to \mathbb{R})$ by defining

$$|\mathcal{U}\rangle = \sum_{i=1}^{C} |i\rangle \otimes |\boldsymbol{u}^i\rangle$$

where we consider $\mathbb{R}^C$ and $L_2(\mathbb{R} \to \mathbb{R})$ to be equipped with the usual euclidean distances, consequently inducing an euclidean metric structure on $\mathcal{S}$ too.

Conceptually, the set of vectors $\{|i\rangle\}_{i=1}^C \subset \mathbb{R}^C$ represents the different cells, while each $|\boldsymbol{u}^i\rangle \in L_2(\mathbb{R} \to \mathbb{R})$ represents the convolution of a spike train of cell i with a real-valued feature function $\phi : \mathbb{R} \to \mathbb{R}$,

$$\langle t|\boldsymbol{u}\rangle = \sum_{n=1}^{N_{\boldsymbol{u}}} \phi(t - u_n)$$

In practice, we will never use the feature functions directly, but we will be only interested in the inner products of the $|i\rangle$ and $|\boldsymbol{u}\rangle$ vectors. We call $c_{ij} \doteq \langle i|j\rangle = \langle i|j\rangle_{\mathbb{R}^C} = c_{ji}$ the *multiunit mixing coefficient* for cells i and j, and $\langle\boldsymbol{u}|\boldsymbol{v}\rangle = \langle\boldsymbol{u}|\boldsymbol{v}\rangle_{L_2}$ the *single-unit inner product*,

$$\begin{aligned}
\langle\boldsymbol{u}|\boldsymbol{v}\rangle &= \langle\{u_1, u_2, \ldots, u_N\} \,|\, \{v_1, v_2, \ldots, v_M\}\rangle = \\
&= \int \mathrm{d}t \langle\boldsymbol{u}|t\rangle\langle t|\boldsymbol{v}\rangle = \int \mathrm{d}t \sum_{n=1}^{N}\sum_{m=1}^{M} \phi\left(t - u_n\right) \phi\left(t - v_m\right) \\
&\doteq \sum_{n=1}^{N}\sum_{m=1}^{M} \mathcal{K}(u_n, v_m)
\end{aligned}$$

where $\mathcal{K}(t_1, t_2) \doteq \int \mathrm{d}t \left[\phi\left(t - t_1\right) \phi\left(t - t_2\right)\right]$ is the *single-unit metric kernel*, and where we have used the fact that the feature function ϕ is real-valued. It follows immediately from the definition above that $\langle\boldsymbol{u}|\boldsymbol{v}\rangle = \langle\boldsymbol{v}|\boldsymbol{u}\rangle$.

Note that, given a cell pair (i, j) or a spike train pair $(\boldsymbol{u}, \boldsymbol{v})$, c_{ij} does not depend on spike times and $\langle\boldsymbol{u}|\boldsymbol{v}\rangle$ does not depend on cell labeling.

With this notation, we can define the *multi-unit spike train distance* as

$$\left\||\mathcal{U}\rangle - |\mathcal{V}\rangle\right\|^2 = \langle\mathcal{U}|\mathcal{U}\rangle + \langle\mathcal{V}|\mathcal{V}\rangle - 2\langle\mathcal{U}|\mathcal{V}\rangle$$

where the *multi-unit spike train inner product* $\langle\mathcal{V}|\mathcal{U}\rangle$ between $\mathcal{U}$ and $\mathcal{V}$ is just the natural bilinear operation induced on $\mathcal{S}$ by the tensor product structure:

$$\begin{aligned}
\langle\mathcal{V}|\mathcal{U}\rangle &= \sum_{i,j=1}^{C} \langle i|j\rangle\langle\boldsymbol{v}^i|\boldsymbol{u}^j\rangle = \sum_{i,j=1}^{C} c_{ij}\langle\boldsymbol{v}^i|\boldsymbol{u}^i\rangle \\
&= \sum_{i=1}^{C} \left[c_{ii}\langle\boldsymbol{v}^i|\boldsymbol{u}^i\rangle + c_{ij}\left(\sum_{j<i}\langle\boldsymbol{v}^i|\boldsymbol{u}^j\rangle + \sum_{j>i}\langle\boldsymbol{v}^i|\boldsymbol{u}^j\rangle\right)\right]
\end{aligned}$$

But $c_{ij} = c_{ji}$ and $\langle \boldsymbol{v}|\boldsymbol{u}\rangle = \langle \boldsymbol{u}|\boldsymbol{v}\rangle$, so

$$\begin{aligned}\sum_{i=1}^{C}\sum_{j<i} c_{ij}\langle \boldsymbol{v}^i|\boldsymbol{u}^j\rangle &= \sum_{j=1}^{C}\sum_{i<j} c_{ji}\langle \boldsymbol{v}^j|\boldsymbol{u}^i\rangle = \sum_{i=1}^{C}\sum_{j>i} c_{ji}\langle \boldsymbol{v}^j|\boldsymbol{u}^i\rangle = \sum_{i=1}^{C}\sum_{j>i} c_{ij}\langle \boldsymbol{v}^j|\boldsymbol{u}^i\rangle \\ &= \sum_{i=1}^{C}\sum_{j>i} c_{ij}\langle \boldsymbol{u}^i|\boldsymbol{v}^j\rangle\end{aligned}$$

and

$$\langle \mathcal{V}|\mathcal{U}\rangle = \sum_{i=1}^{C}\left[c_{ii}\langle \boldsymbol{v}^i|\boldsymbol{u}^i\rangle + c_{ij}\sum_{j>i}\left(\langle \boldsymbol{v}^i|\boldsymbol{u}^j\rangle + \langle \boldsymbol{u}^i|\boldsymbol{v}^j\rangle\right)\right]$$

Now, normally we are interested in the particular case where c_{ij} is the same for all pair of distinct cells:

$$c_{ij} = \begin{cases} 1 & \text{if } i = j \\ c & \text{if } i \neq j \end{cases}$$

and under this assumption we can write

$$\langle \mathcal{V}|\mathcal{U}\rangle = \sum_{i=1}^{C}\left[\langle \boldsymbol{v}^i|\boldsymbol{u}^i\rangle + c\sum_{j>i}\left(\langle \boldsymbol{v}^i|\boldsymbol{u}^j\rangle + \langle \boldsymbol{u}^i|\boldsymbol{v}^j\rangle\right)\right]$$

and

$$\begin{aligned}\||\mathcal{U}\rangle - |\mathcal{V}\rangle\|^2 = \sum_{i=1}^{C}\Bigg\{&\langle \boldsymbol{u}^i|\boldsymbol{u}^i\rangle + c\sum_{j>i}\left(\langle \boldsymbol{u}^i|\boldsymbol{u}^j\rangle + \langle \boldsymbol{u}^i|\boldsymbol{u}^j\rangle\right) + \\ &+\langle \boldsymbol{v}^i|\boldsymbol{v}^i\rangle + c\sum_{j>i}\left(\langle \boldsymbol{v}^i|\boldsymbol{v}^j\rangle + \langle \boldsymbol{v}^i|\boldsymbol{v}^j\rangle\right) + \\ &-2\left[\langle \boldsymbol{v}^i|\boldsymbol{u}^i\rangle + c\sum_{j>i}\left(\langle \boldsymbol{v}^i|\boldsymbol{u}^j\rangle + \langle \boldsymbol{u}^i|\boldsymbol{v}^j\rangle\right)\right]\Bigg\}\end{aligned}$$

Rearranging the terms

$$\begin{aligned}\||\mathcal{U}\rangle - |\mathcal{V}\rangle\|^2 = \sum_{i=1}^{C}\Bigg[&\langle \boldsymbol{u}^i|\boldsymbol{u}^i\rangle + \langle \boldsymbol{v}^i|\boldsymbol{v}^i\rangle - 2\langle \boldsymbol{v}^i|\boldsymbol{u}^i\rangle + \\ &+2c\sum_{j>i}\left(\langle \boldsymbol{u}^i|\boldsymbol{u}^j\rangle + \langle \boldsymbol{v}^i|\boldsymbol{v}^j\rangle - \langle \boldsymbol{v}^i|\boldsymbol{u}^j\rangle - \langle \boldsymbol{v}^j|\boldsymbol{u}^i\rangle\right)\Bigg]\end{aligned}$$

1.2.2 Van Rossum-like metrics

In Van Rossum-like metrics, the feature function and the single-unit kernel are, for $\tau \neq 0$,

$$\begin{aligned}\phi_\tau^{\mathrm{VR}}(t) &= \sqrt{\frac{2}{\tau}} \cdot e^{-t/\tau}\theta(t) \\ \mathcal{K}_\tau^{\mathrm{VR}}(t_1, t_2) &= \begin{cases} 1 & \text{if } t_1 = t_2 \\ e^{-|t_1 - t_2|/\tau} & \text{if } t_1 \neq t_2 \end{cases}\end{aligned}$$

where θ is the Heaviside step function (with $\theta(0) = 1$), and we have chosen to normalise ϕ_τ^{VR} so that

$$\left\|\phi_\tau^{\mathrm{VR}}\right\|_2 = \sqrt{\int \mathrm{d}t \,[\phi_\tau^{\mathrm{VR}}(t)]^2} = 1 \quad .$$

In the $\tau \to 0$ limit,

$$\phi_0^{\mathrm{VR}}(t) = \delta(t)$$
$$\mathcal{K}_0^{\mathrm{VR}}(t_1, t_2) = \begin{cases} 1 & \text{if } t_1 = t_2 \\ 0 & \text{if } t_1 \neq t_2 \end{cases}$$

In particular, the single-unit inner product now becomes

$$\langle \boldsymbol{u}|\boldsymbol{v}\rangle = \sum_{n=1}^{N}\sum_{m=1}^{M} \mathcal{K}^{\mathrm{VR}}(u_n, v_m) = \sum_{n=1}^{N}\sum_{m=1}^{M} e^{-|u_n - v_m|/\tau}$$

Markage formulas

For a spike train $\boldsymbol{u}$ of length N and a time t we define the index $\tilde{N}\left(\boldsymbol{u}, t\right)$

$$\tilde{N}\left(\boldsymbol{u}, t\right) \doteq \max\{n | u_n < t\}$$

which we can use to re-write $\langle \boldsymbol{u}|\boldsymbol{u}\rangle$ without the absolute values:

$$\begin{aligned}
\langle \boldsymbol{u}|\boldsymbol{u}\rangle &= \sum_{n=1}^{N}\left(\sum_{m|v_m<u_n} e^{-(u_n-v_m)/\tau} + \sum_{m|v_m>u_n} e^{-(v_m-u_n)/\tau} + \sum_{m=1}^{M}\delta\left(u_n, v_m\right)\right) \\
&= \sum_{n=1}^{N}\left(\sum_{m|v_m<u_n} e^{-(u_n-v_m)/\tau} + \sum_{m|u_m<v_n} e^{-(v_n-u_m)/\tau} + \sum_{m=1}^{M}\delta\left(u_n, v_m\right)\right) \\
&= \sum_{n=1}^{N}\left[\sum_{m=1}^{\tilde{N}(\boldsymbol{v},u_n)} e^{-(u_n-v_m)/\tau} + \sum_{m=1}^{\tilde{N}(\boldsymbol{u},v_n)} e^{-(v_n-u_m)/\tau} + \delta\left(u_n, v_{\tilde{N}(\boldsymbol{v},u_n)+1}\right)\right] \\
&= \sum_{n=1}^{N}\left[e^{-(u_n-v_{\tilde{N}(\boldsymbol{v},u_n)})/\tau}\sum_{m=1}^{\tilde{N}(\boldsymbol{v},u_n)} e^{-(v_{\tilde{N}(\boldsymbol{v},u_n)}-v_m)/\tau} + \right. \\
&\qquad + e^{-(v_n-u_{\tilde{N}(\boldsymbol{u},v_n)})/\tau}\sum_{m=1}^{\tilde{N}(\boldsymbol{u},v_n)} e^{-(u_{\tilde{N}(\boldsymbol{u},v_n)}-u_m)/\tau} + \\
&\qquad \left. + \,\delta\left(u_n, v_{\tilde{N}(\boldsymbol{v},u_n)+1}\right)\right]
\end{aligned}$$

For a spike train $\boldsymbol{u}$ of length N, we also define the the *markage vector* $\boldsymbol{m}$, with the same length as $\boldsymbol{u}$, through the following recursive assignment:

$$\begin{aligned}
m_1(\boldsymbol{u}) &\doteq 0 \\
m_n(\boldsymbol{u}) &\doteq (m_{n-1} + 1)\, e^{-(u_n - u_{n-1})/\tau} \quad \forall n \in \{2, \ldots, N\}
\end{aligned}$$

It is easy to see that

$$\begin{aligned}
m_n(\boldsymbol{u}) &= \sum_{k=1}^{n-1} e^{-(u_n-u_k)/\tau} = \left(\sum_{k=1}^{n} e^{-(u_n-u_k)/\tau}\right) - e^{-(u_n-u_n)/\tau} \\
&= \sum_{k=1}^{n} e^{-(u_n-u_k)/\tau} - 1
\end{aligned}$$

and in particular

$$\sum_{n=1}^{\tilde{N}(\boldsymbol{u},t)} e^{-(u_{\tilde{N}(\boldsymbol{u},t)}-u_n)/\tau} = 1 + m_{\tilde{N}(\boldsymbol{u},t)}(\boldsymbol{u})$$

With this definition, we get

$$\begin{aligned}\langle \boldsymbol{u}|\boldsymbol{v}\rangle = \sum_{n=1}^{N} \Bigg[& e^{-(u_n - v_{\tilde{N}(\boldsymbol{v},u_n)})/\tau}\left(1 + m_{\tilde{N}(\boldsymbol{v},u_n)}(\boldsymbol{v})\right) + \\ & + e^{-(v_n - u_{\tilde{N}(\boldsymbol{u},v_n)})/\tau}\left(1 + m_{\tilde{N}(\boldsymbol{u},v_n)}(\boldsymbol{u})\right) + \\ & + \delta\left(u_n, v_{\tilde{N}(\boldsymbol{v},u_n)+1}\right) \Bigg]\end{aligned}$$

Finally, note that because of the definition of the markage vector

$$e^{-(u_n - u_{\tilde{N}(\boldsymbol{u},u_n)})/\tau}\left(1 + m_{\tilde{N}(\boldsymbol{u},u_n)}(\boldsymbol{u})\right) = e^{-(u_n-u_{n-1})/\tau}\left(1 + m(\boldsymbol{u})\right) = m_n(\boldsymbol{u})$$

so that in particular

$$\langle \boldsymbol{u}|\boldsymbol{u}\rangle = \sum_{n=1}^{N} \left(1 + 2m_n(\boldsymbol{u})\right)$$

A formula for the efficient computation of multiunit Van Rossum spike train metrics, used by `pymuvr`, can then be obtained by opportunely substituting these expressions for the single-unit scalar products in the definition of the multiunit distance.

1.3 Usage

1.3.1 Examples

```
>>> import pymuvr
>>> # define two sets of observations for two cells
>>> observations_1 = [[[1.0, 2.3], # 1st observation, 1st cell
...                    [0.2, 2.5, 2.7]],            # 2nd cell
...                   [[1.1, 1.2, 3.0], # 2nd observation
...                    []],
...                   [[5.0, 7.8],
...                    [4.2, 6.0]]]
>>> observations_2 = [[[0.9],
...                    [0.7, 0.9, 3.3]],
...                   [[0.3, 1.5, 2.4],
...                    [2.5, 3.7]]]
>>> # set parameters for the metric
>>> cos = 0.1
>>> tau = 1.0
>>> # compute distances between all observations in set 1
>>> # and those in set 2
>>> pymuvr.dissimilarity_matrix(observations_1,
...                             observations_2,
...                             cos,
```

```
...                             tau,
...                             'distance')
array([[ 2.40281585,  1.92780957],
       [ 2.76008964,  2.31230263],
       [ 3.1322069 ,  3.17216524]])
>>> # compute inner products
>>> pymuvr.dissimilarity_matrix(observations_1,
...                             observations_2,
...                             cos,
...                             tau,
...                             'inner product')
array([[ 4.30817654,  5.97348384],
       [ 2.08532468,  3.85777053],
       [ 0.59639918,  1.10721323]])
>>> # compute all distances between observations in set 1
>>> pymuvr.square_dissimilarity_matrix(observations_1,
...                                    cos,
...                                    tau,
...                                    'distance')
array([[ 0.        ,  2.6221159 ,  3.38230952],
       [ 2.6221159 ,  0.        ,  3.10221811],
       [ 3.38230952,  3.10221811,  0.        ]])
>>> # compute inner products
>>> pymuvr.square_dissimilarity_matrix(observations_1,
...                                    cos,
...                                    tau,
...                                    'inner product')
array([[ 8.04054275,  3.3022304 ,  0.62735459],
       [ 3.3022304 ,  5.43940985,  0.23491838],
       [ 0.62735459,  0.23491838,  4.6541841 ]])
```

See the `examples` and `test` directories in the source distribution for more detailed examples of usage. These should also have been installed alongside the rest of the pymuvr files.

The script `examples/benchmark_versus_spykeutils.py` compares the performance of pymuvr with the pure Python/NumPy implementation of the multiunit Van Rossum distance in spykeutils.

1.3.2 Reference

`pymuvr.`**`dissimilarity_matrix`**(*observations1*, *observations2*, *cos*, *tau*, *mode*)

Return the *bipartite* (rectangular) dissimilarity matrix between the observations in the first and the second list.

Parameters

- **observations1,observations2** (*list*) – Two lists of multi-unit spike trains to compare. Each *observations* parameter must be a thrice-nested list of spike times, with *observations[i][j][k]* representing the time of the kth spike of the jth cell of the ith observation.
- **cos** (*float*) – mixing parameter controlling the interpolation between *labelled-line* mode (cos=0) and *summed-population* mode (cos=1). It corresponds to the cosine of the angle between the vectors used for the euclidean embedding of the multiunit spike trains.
- **tau** (*float*) – time scale for the exponential kernel, controlling the interpolation between pure *coincidence detection* (tau=0) and *spike count* mode (very large tau). Note that setting tau=0 is always allowed, but there is a range (0, epsilon) of forbidden values that tau is not allowed to assume. The upper bound of this range is proportional to the absolute value of the largest spike time in *observations*, with the proportionality constant being system-dependent. As a rule of thumb tau and the spike times should be within 4 orders of magnitude of each

other; for example, if the largest spike time is 10s a value of tau>1ms will be expected. An exception will be raised if tau falls in the forbidden range.

- **mode** (*string*) – type of dissimilarity measure to be computed. Must be either 'distance' or 'inner product'.

Returns A len(observations1) x len(observations2) numpy array containing the dissimilarity (distance or inner product) between each pair of observations that can be formed by taking one observation from *observations1* and one from *observations2*.

Return type *numpy.ndarray*

Raises

- **IndexError** – if the observations in *observations1* and *observations2* don't have all the same number of cells.
- **OverflowError** – if *tau* falls in the forbidden interval.

`pymuvr.`**`square_dissimilarity_matrix`**(*observations*, *cos*, *tau*, *mode*)

Return the *all-to-all* (square) dissimilarity matrix for the given list of observations.

Parameters

- **observations** (*list*) – A list of multi-unit spike trains to compare.
- **cos** (*float*) – mixing parameter controlling the interpolation between *labelled-line* mode (cos=0) and *summed-population* mode (cos=1).
- **tau** (*float*) – time scale for the exponential kernel, controlling the interpolation between pure *coincidence detection* (tau=0) and *spike count* mode (very large tau).
- **mode** (*string*) – type of dissimilarity measure to be computed. Must be either 'distance' or 'inner product'.

Returns A len(observations) x len(observations) numpy array containing the dissimilarity (distance or inner product) between all possible pairs of observations.

Return type *numpy.ndarray*

Raises

- **IndexError** – if the observations in *observations* don't have all the same number of cells.
- **OverflowError** – if *tau* falls in the forbidden interval.

Effectively equivalent to `dissimilarity_matrix(observations, observations, cos, tau)`, but optimised for speed. See `pymuvr.dissimilarity_matrix()` for details.

`pymuvr.`**`distance_matrix`**(*trains1*, *trains2*, *cos*, *tau*)

Return the *bipartite* (rectangular) distance matrix between the observations in the first and the second list.

Convenience function; equivalent to `dissimilarity_matrix(trains1, trains2, cos, tau, "distance")`. Refer to `pymuvr.dissimilarity_matrix()` for full documentation.

`pymuvr.`**`square_distance_matrix`**(*trains*, *cos*, *tau*)

Return the *all-to-all* (square) distance matrix for the given list of observations.

Convenience function; equivalent to `square_dissimilarity_matrix(trains, cos, tau, "distance")`. Refer to `pymuvr.square_dissimilarity_matrix()` for full documentation.

1.4 C++ API reference

class **ConvolvedSpikeTrain**

A class representing a single-unit spike train for which several helper vectors relating to a Van Rossum-like exponential convolution have been calculated.

Public Functions

ConvolvedSpikeTrain()

Default constructor.

ConvolvedSpikeTrain(std::vector< double > *spikes*, double *tau*)

Constructor accepting a spike train.

Parameters

- `spikes` - A *vector* of spike times.
- `tau` - Time scale for the exponential kernel. There is a limit to how small *tau* can be compared to the absolute value of the spike times. An exception will be raised if this limit is exceeded; its value is system-dependent, but as a rule of thumb *tau* and the spike times should be within 4 orders of magnitude of each other.

void **distance**(std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains*, double *c*, double ** *d_matrix*)

Compute the square (all-to-all) distance matrix for a set of multi-unit spike trains.

Parameters

- `trains` - The set of multi unit spike trains.
- `c` - Cosine of the multi-unit mixing angle. *c=0* corresponds to labelled-line code, *c=1* to summed-population code.
- `d_matrix` - The *trains.size()* x *trains.size()* matrix where the results will be written.

void **distance**(std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains1*, std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains2*, double *c*, double ** *d_matrix*)

Compute the rectangular (bipartite) distance matrix between two sets of multi-unit spike trains.

Parameters

- `trains1trains2` - The sets of multi unit spike trains.
- `c` - Cosine of the multi-unit mixing angle. *c=0* corresponds to labelled-line code, *c=1* to summed-population code.
- `d_matrix` - The *trains1.size()* x *trains2.size()* matrix where the results will be written.

void **inner_product**(std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains*, double *c*, double ** *g_matrix*)

Compute the square (all-to-all) inner product matrix for a set of multi-unit spike trains.

Parameters

- `trains` - The set of multi unit spike trains.
- `c` - Cosine of the multi-unit mixing angle. *c=0* corresponds to labelled-line code, *c=1* to summed-population code.
- `g_matrix` - The *trains.size()* x *trains.size()* matrix where the results will be written.

void **inner_product** (std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains1*, std::vector< std::vector< *ConvolvedSpikeTrain* > > & *trains2*, double *c*, double ** *g_matrix*)

Compute the rectangular (bipartite) inner product matrix between two sets set of multi-unit spike trains.

Parameters

- `trains1trains2` - The sets of multi unit spike trains.
- `c` - Cosine of the multi-unit mixing angle. *c=0* corresponds to labelled-line code, *c=1* to summed-population code.
- `g_matrix` - The *trains1.size()* x *trains2.size()* matrix where the results will be written.

1.5 License

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

1.6 Acknowledgements

This software has been developed at the Silver Lab, University College London, with support from the EU Marie Curie Initial Training Network CEREBNET (FP7-ITN-PEOPLE-2008; 238686).

Thanks to Robert Pröpper (author of spykeutils) for testing under Windows and providing bugfixes.

D

S