Reconstruction of the functional connectivity

April 7, 2020

Abstract

To identify the functional connectivity between neurons, previous works have used Hawkes processes to model conditional intensities of spike trains and have reconstructed functional connectivity by penalized least square method. We propose here a new method to construct the matrices in the resulting Lasso problem and the choice of another solver, which seems to be more efficient, to improve computational times.

Contents

1 Presentation of the problem

The goal is to study the functional connectivity between neurons, which is an important issue in Neuroscience. The study is based on the simultaneous recordings of firing times of action potentials, also called spikes, of M neurons. Like what was introduced in [4], we consider M simultaneous spike trains modelled as multivariate Hawkes processes. A spike train is a set of spikes and we denote by N^i the spike train of neuron i with contains all the spikes of neuron i, for all $i \in [\![1, M]\!]$. The intensity of the i-th spike train N^i has the following form, where dN_u^j denotes the counting measure of the spike train N^j with respect to the variable u,

$$\lambda_i(t) = \left(\nu_i + \sum_{j=1}^{M} \int_{-\infty}^{t^-} h_{j\to i}(t-u)\, dN_u^j \right)_+ , \quad \forall\, t, \quad \forall i \in [\![1, M]\!],$$

which rewrites

$$\lambda_i(t) = \left(\nu_i + \sum_{j=1}^{M} \sum_{T \in N^j, T<t} h_{j\to i}(t-T) \right)_+ , \quad \forall\, t, \quad \forall i \in [\![1, M]\!].$$

Coefficient ν_i is the spontaneous firing rate of the i-th spike train, which gives the averaged frequency of apparition of a new spike, and function $h_{j\to i}$, which depends on time, models the interaction from the j-th train to the i-th one. We assume that $h_{j\to i}$ functions are piecewise constant on a partition of K bins of length δ, such that

$$h_{j\to i} = \sum_{k=1}^{K} a_{j\to i}^k \varphi_k \quad \text{where } \varphi_k = \mathbb{1}_{((k-1)\delta, k\delta]}.$$

Coefficients (ν_i) and $(a_{j\to i}^k)$ need to be estimated, for all $(i, j) \in [\![1, M]\!]^2$ and $k \in [\![1, K]\!]$.

neuro-stat code introduced in [4] allows to find a sparse estimation of coefficients (ν_i) and $(a_{j\to i}^k)$ and therefore to obtain the functional connectivity graph in which the existence of a connection between neurons j and i corresponds to the non nullity of $h_{j\to i}$. Moreover, the code allows to estimate different data sets depending on the segmentations of the recording and different graphs (typically we can associate a graph with an animal behaviour).

We focus here on the improvement of the algorithm on a fixed time range $(T_{\min}, T^{\max}]$.

We know that $\lambda_i\, dt$ is an approximation of dN^i, for all $i \in [\![1, M]\!]$. We want to minimize the distance between both

Generally, we focus on the following least square criterion that will be penalized later

$$\int_{T_{\min}}^{T^{\max}} \overline{\lambda_i}^2(t)\,dt - 2\int_{T_{\min}^+}^{T^{\max}} \overline{\lambda_i}(t)\,dN_t^i$$

where $\overline{\lambda}_i$ is an intensity candidate of the form

$$\begin{aligned}\overline{\lambda_i}(t) &= \overline{\nu_i} + \sum_{j=1}^{M} \sum_{T\in N^j, T<t} \overline{h}_{j\to i}(t-T), \quad \forall\, t, \quad \forall i \in [\![1, M]\!] \\ &\text{with } \overline{h}_{j\to i} = \sum_{k=1}^{K} \overline{a}_{j\to i}^k\, \varphi_k\end{aligned}$$

It is straightforward to see that $\overline{\lambda}_i$ writes on the dictionary of previsible functions in the following form

Introducing $\mathbf{b}$, $\mathbf{G}$, we see that the least square criterion

We penalize by a l_1 norm with a weight (see [3])

We introduce several matrices and vectors intervening in the least square criterion and used in the following:

- G is a squared symmetric matrix of order $(1 + MK)$,
- $\mathbf{b}$, $\mathbf{d}$ and $\mu_{\mathbf{2}}$ are matrices of dimension $(1 + MK)$-by-M
- $\mu_{\mathbf{A}}$ is a vector of size $(1 + MK)$

In the sequel, we resume the definitions of [4] and we set $\alpha(l, k) = 1+(l-1)\,K+k$ as an index in $[\![2, 1+M\,K]\!]$, for $k \in [\![1, K]\!]$ and $l \in [\![1, M]\!]$:

$$\begin{aligned}\psi_t^l(\varphi_k) &= \int_{-\infty}^{t^-} \varphi_k(t-u)\,dN_u^l = \sum_{T<t, T\in N^l} \mathbb{1}_{((k-1)\delta, k\delta]}(t-T) \\ \mathbf{b}^i_{\alpha(l,k)} &= \int_{T_{\min}^+}^{T^{\max}} \psi_t^l(\varphi_k)\,dN_t^i,\ \mathbf{b}_1^i = \#\left\{T \in N^i,\ \ T \in (T_{\min}, T^{\max}]\right\}, \\ \mathbf{G}_{\alpha(l_1,k_1),\alpha(l_2,k_2)} &= \int_{T_{\min}}^{T^{\max}} \psi_t^{l_1}(\varphi_{k_1})\,\psi_t^{l_2}(\varphi_{k_2})dt,\ \mathbf{G}_{\alpha(l,k),1} = \int_{T_{\min}}^{T^{\max}} \psi_t^l(\varphi_k)\,dt,\ \mathbf{G}_{1,1} = T^{\max} - T_{\min}\end{aligned}$$

We use the same definition as [4] for $\mathbf{d}$, in which we choose $\gamma = 3$.

$$\begin{aligned}\mathbf{d}^i &= \sqrt{2\gamma\, c_{log}\, \mu_{\mathbf{2}i}} + \frac{\gamma}{3} c_{log}\, \mu_{\mathbf{A}},\ \forall i \in [\![1, 1+MK]\!],\ \ c_{log} = \log((1+MK)M) \\ (\mu_{\mathbf{A}})_{\alpha(l,k)} &= \sup_{t\in(T_{\min}, T^{\max}]} |\psi_t^l(\varphi_k)|,\ \ (\mu_{\mathbf{A}})_1 = 1, \\ (\mu_{\mathbf{2}})^i_{\alpha(l,k)} &= \int_{T_{\min}^+}^{T^{\max}} \left(\psi_t^l(\varphi_k)\right)^2\, dN_t^i,\ \ (\mu_{\mathbf{2}})_1^i = \#\left\{T \in N^i,\ \ T \in (T_{\min}, T^{\max}]\right\}.\end{aligned}$$

We follow the same procedure as [4] to stabilize the method and weshould solve M Lasso problems of dimension $(1+M\,K)$, to estimate coefficients $a^k_{j\to i}$

$$\begin{aligned}\mathbf{a}^i_{BL} &= \arg\min_{\beta\in\mathbb{R}^{(1+M\,K)}} -2\,{}^T\mathbf{b}^i\,\beta + {}^T\beta\,\mathbf{G}\,\beta + 2\,{}^T\mathbf{d}^i\,|\beta| \\ &\quad \text{with } \mathbf{a}^i_{BL} = (\nu_i, a^1_{1\to i}, a^2_{1\to i}, \cdots, a^K_{1\to i}, a^1_{2\to i}, \cdots, a^K_{M\to i})\end{aligned}$$

2 Notation and definitions

2.1 Interval and function definitions

- N^j denotes the j-th spike train or set of spikes of the neuron j
- M is the total number of neurons.
- K is the number of bins or time windows.
- δ is the size of a bin (or a time window).
- $Ntot$ is the total number of spikes.
- A is the scope verifying $A = K\,\delta$. If the distance between two spikes is strictly greater than A, they don't influence each other.
- $T_{\min}$ is the beginning of the study time interval.
- $T^{\max}$ is the end of the study time interval.
- We set $\mathbb{I}_k = ((k-1)\,\delta, k\,\delta]$
- We set $\varphi_k(t) = \mathbb{1}_{\mathbb{I}_k}(t)$, $\forall\ 1 \leqslant k \leqslant K, \forall\, t$.
- We set

$$\psi^l_t(\varphi_k) = \int_{-\infty}^{t^-} \mathbb{1}_{\mathbb{I}_k}(t-u)\,dN^l_u = \sum_{T<t, T\in N^l} \mathbb{1}_{\mathbb{I}_k}(t-T)$$

2.2 Numbering

For each vector or matrix, we start from the spontaneous part then, for a given neuron, we start from the first bin of the first neuron, then the second bin, the third one, up to the last bin, then we switch to the first bin of the next neuron and so on...

2.3 Matrix definitions

2.3.1 Matrix b (or μ_1)

b, also denoted by $\mu_{\mathbf{1}}$, is a $(1+MK)$-by-M matrix (a rectangular matrix) with positive integer coefficients (b or $\mu_1 \in \mathcal{M}^{(1+MK)\times M}(\mathbb{N})$).

- Spontaneous part of neuron r

$$b^r_{spont} = \int_{T^+_{\min}}^{T^{\max}} dN^r_t = \#\{T : T_{\min} < T \leqslant T^{\max}, T \in N^r\}$$

- Contribution of neuron l to neuron r, for bin k

$$b^r_{(l,k)} = \int_{T^+_{\min}}^{T^{\max}} \psi^l_t(\varphi_k)\, dN^r_t$$

- *get_k* is a function with three arguments which returns an unsigned integer such that:

  $get\_k(x, \delta, \varepsilon) = k$ such that $x \in ((k-1)\,\delta, k\,\delta]$, ε is for numerical precision, $x > 0$,

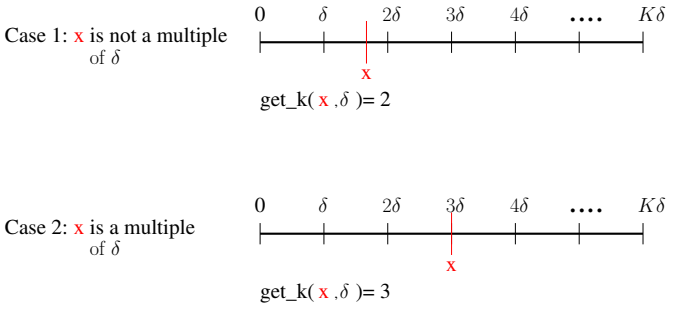

Figure 1: Description of *get_k* function

The algorithm 1 describes how to compute b.
Complexity for b seems to be $O(Ntot\,\overline{\nu}\,K\,\delta)$ where $Ntot$ is the total number of spikes, independently of the neurons and $\overline{\nu}$ is the average firing rate between neurons.

2.3.2 Matrix G

G is a symmetric $(1+MK)$-by-$(1+MK)$ matrix with real coefficients ($G \in \mathcal{M}^{(1+MK)}(\mathbb{R})$).

- Spontaneous part

$$G_{spont,spont} = \int_{T_{\min}}^{T^{\max}} dt = T^{\max} - T_{\min}$$

- First row or first column

$$G_{spont,(l,k)} = \int_{T_{\min}}^{T^{\max}} \psi_t^l(\varphi_k)\, dt$$

- Contribution of neuron l_1 and bin k_1 for neuron l_2 and bin k_2

$$G_{(l_1,k_1),(l_2,k_2)} = \int_{T_{\min}}^{T^{\max}} \psi_t^{l_1}(\varphi_{k_1})\, \psi_t^{l_2}(\varphi_{k_2}) dt$$

-

$$\begin{aligned} G_{spont,(l,k)} &\simeq \sum_{\substack{\theta \in N^l, \\ (T_{\min}-A) \leqslant \theta < T^{\max}}} (\min(T^{\max}, k\,\delta + \theta) - \max(T_{\min}, (k-1)\,\delta + \theta)) \\ G_{(l_1,k_1),(l_2,k_2)} &\simeq \sum_{\substack{\tau \in N^{l_1}, \theta \in N^{l_2}, \\ (T_{\min}-A) \leqslant \tau < T^{\max}, \\ (T_{\min}-A) \leqslant \theta < T^{\max}}} \int_{T_{\min}}^{T^{\max}} \mathbb{1}_{(\mathbb{I}_{k_1}+\tau) \cap (\mathbb{I}_{k_2}+\theta) \cap]T_{\min}, T^{\max}]}(t)\, dt \end{aligned}$$

- low: at index i, it returns the lowest index j in the spike array T such that $|T_j - T_i| \leqslant K\,\delta$
- restrict_2(T, $T_{\min}$, $T^{\max}$) → restriction of T to $(T_{\min}, T^{\max}]$
- *get_low_index* is a function described in Figure 2, with two arguments which returns an unsigned integer such that

$$get\_low\_index(x, T) = i \;\text{ such that } T_i > x \;\text{ and } 0 \leqslant i \leqslant T.size()$$

The algorithm 2 describes how to compute G.
Complexity for G seems to be $O(Ntot\, K\, \delta\, \overline{\nu}\, K^2)$ where $Ntot$ is the total number of spikes, independently of the neurons.

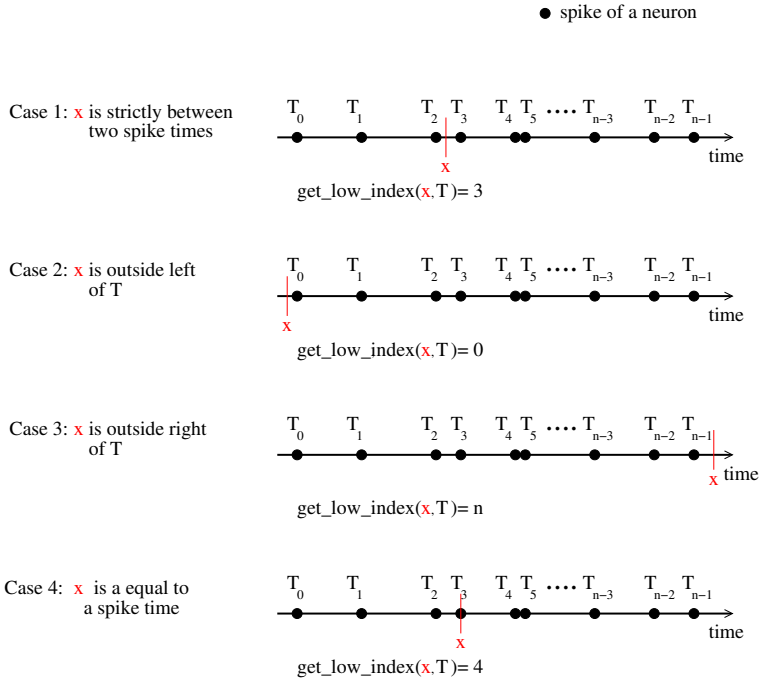

Figure 2: Description of *get_low_index* function

2.3.3 Vector $\mu_{\mathbf{A}}$

$\mu_{\mathbf{A}}$ is a vector of size $(1+MK)$ with positive integer coefficients ($\mu_A \in \mathbb{N}^{(1+MK)}$).

- μ_A is defined such that

$$\begin{aligned} (\mu_A)_{spont} &= 1, \\ (\mu_A)_{(l,k)} &= \sup_{t\in(T_{\min},T^{\max}]} |\psi_t^l(\varphi_k)| \end{aligned}$$

Let us recall that

$$\psi_t^l(\varphi_k) = \sum_{\substack{T<t,\\ T\in N^l}} \mathbb{1}_{\mathbb{I}_k+T}(t) \quad \text{with } \mathbb{I}_k =](k-1)\,\delta, k\,\delta]$$

Assume that, we have m intervals I_i satisfying, for all i such that $1 \leqslant i \leqslant m$,

$$I_i =]a_i, b_i] \quad \text{such that } b_i > a_i$$

We know that

$$\max \sum_{i=1}^{m} \mathbb{1}_{I_i}(t) = \text{biggest j such that } \bigcap_{p=1}^{j} I_{i_p} \neq \varnothing \quad \text{with } 1 \leqslant i_p \leqslant m \text{ and } i_1 < i_2 < \cdots < i_j$$

In our case, we have $b_i = a_i+\delta$ which implies $I_i =]a_i, a_i+\delta]$. Consequently, for example, for two intervals, for any $(l, q) \in [\![1, m]\!]^2$

$$\begin{aligned} I_l \bigcap I_q &=]\max(a_l, a_q), \min(a_q, a_l)+\delta] \\ I_l \bigcap I_q &=]a_q, a_l+\delta] \quad \text{if } q > l \text{ (we assume } a_q > a_l) \end{aligned}$$

If l is known and q unknown, we look for all the indices $q > l$ such that $(a_l + \delta) > a_q$.

In our case, for m intervals, we have

$$I_{i_1} \bigcap I_{i_2} \bigcap \cdots \bigcap I_{i_j} =]a_{i_j}, a_{i_1}+\delta]$$

If i_1 is known and i_j unknown, we search and count all the indices i_p such that $(a_{i_1} + \delta) > a_{i_p}$. We set $i_j = \max_p i_p$

The algorithm 5 describes how to compute μ_A.
Complexity for μ_A seems to be $O(M\,K\,Ntot)$ where $Ntot$ is the total number of spikes, independently of the neurons (A VERIFIER)

2.3.4 Matrix μ_2

- μ_2 is a $(1+MK)$-by-M matrix (a rectangular matrix) with positive integer coefficients, it has the same dimensions as b (or μ_1) ($\mu_2 \in \mathcal{M}^{(1+MK)\times M}(\mathbb{N})$).

  For neuron r, $1 \leqslant r \leqslant M$, with $1 \leqslant l \leqslant M$, $1 \leqslant k \leqslant K$, we have

$$
\begin{aligned}
{\mu_2}^{(r)}_{(l,k)} &= \int_{T^+_{\min}}^{T^{\max}} (\psi^l_t(\varphi_k))^2 \, dN^r_t \\
(\psi^l_t(\varphi_k))^2 &= \left(\sum_{\substack{T<t,\\ T\in N^l}} \mathbb{1}_{\mathbb{I}_k}(t-T) \right)^2 \\
&= \sum_{\substack{T<t,\\ T\in N^l}} \mathbb{1}_{\mathbb{I}_k}(t-T) + 2 \sum_{\substack{T_1<t, T_2<T_1\\ T_1,T_2\in N^l}} \mathbb{1}_{(\mathbb{I}_k+T_1)\cap(\mathbb{I}_k+T_2)}(t) \\
&= \psi^l_t(\varphi_k) + 2 \sum_{\substack{T_1<t, T_2<T_1\\ T_1,T_2\in N^l\\ (T_1-T_2)<\delta}} \mathbb{1}_{]T_1+(k-1)\delta, T_2+k\delta]}(t) \\
{\mu_2}^{(r)}_{(l,k)} &= {\mu_1}^{(r)}_{(l,k)} + 2\,\zeta \quad \text{where} \quad \zeta = \sum_{\substack{\tau\in N^r, T_{\min}<\tau\leqslant T^{\max}\\ T_1<\tau, T_2<T_1\\ T_1,T_2\in N^l, (T_1-T_2)<\delta}} \mathbb{1}_{]T_1+(k-1)\delta, T_2+k\delta]}(\tau)
\end{aligned}
$$

  If μ_1 is already known, computing ζ is sufficient to get μ_2.

  Complexity for μ_2 seems to be $O(M\, nmax^2)$ where $nmax$ is the maximum number of spikes per neuron. (A VERIFIER)

- *compute_k*(τ, T, δ) is a function used to compute μ_2 such that:

$$
compute\_k(\tau,T,\delta) = \begin{cases} 1 & \text{if } \dfrac{\tau-\delta-T}{\delta} < 0 \\ \lfloor \dfrac{\tau-\delta-T}{\delta} \rfloor + 2 & \text{otherwise} \end{cases}
$$

The algorithm 4.4 describes how to compute μ_2.

3 Input data

Input data are spike times of each neuron. There are several possibilities: *DataNeur* or *DataSpike* are two possible choices.

3.1 DataSpike

- A DataSpike corresponds to a matrix of dimension 2-by-$Ntot$, where $Ntot$ is the total number of spikes, whatever the number of neurons is.

- The first row contains spike times in the increasing order, whatever the neuron.
- The second row contains the neuron number corresponding to the spike of the first row at the same column.
- For example: for 3 neurons, each neuron has $Ns = [2, 1, 3]$ spikes respectively, the obtained DataSpike is a matrix 2-by-6 such that (numbering starts at 0)

$$\begin{pmatrix} 0.05 & 0.21 & 0.4 & 0.46 & 0.6 & 0.62 \\ 2 & 2 & 0 & 2 & 0 & 1 \end{pmatrix}$$

3.2 DataNeur

- A DataNeur is a matrix of dimension M-by-(1+max(Ns)), where M is the number of neurons and Ns is the number of spikes per neuron.
- Each row of the matrix contains the spikes of the corresponding neuron, sorted in the increasing order.
- When there is no spike for a neuron, we fill the matrix with "0".
- For example, for the same test as before: 3 neurons, $Ns = [2, 1, 3]$, the obtained DataNeur is a matrix 3-by-4 such that

$$\begin{pmatrix} 2 & 0.4 & 0.6 & 0 \\ 1 & 0.62 & 0 & 0 \\ 3 & 0.05 & 0.21 & 0.46 \end{pmatrix}$$

4 Algorithms

4.1 Computation of b (or μ_1)

Algorithm 1 Computation of b

```
1: T ← DS[0,]                                   # first row of DS: all spike values
2: neur ← DS[1,]                                # second row of DS: neuron numbers
3: Ntot ← length(T)                             # total number of spikes
4: A ← K ∗ δ                                    # scope, K bins of size δ
5: b ← matrix(0, nrow=(1+M*K), ncol= M)         # init of b
6: low ← vector(mode='integer', length=Ntot)    # init of low - for the State
7: spont ← vector(mode='integer', length=M)     # init of spont - spontaneous part
8: ilow ← 0
9: eps ← 1.e-12                                 # for numerical precision
10: for (i=0:(Ntot-1)) do                       # loop over spikes
11:     t ← T[i]; r ← neur[i]
12:     while (|T[ilow] − t| > A) do
13:         ilow ← ilow+1
14:     low[i] ← ilow
15:     if (T_min < t ⩽ T^max) then
16:         spont[r] ← spont[r]+1
17:         for (j=ilow:(i-1)) do
18:             if (|t − T[j]| ⩽ eps) then
19:                 break
20:             k ← get_k(t-T[j], delta)
21:             l ← neur[j]
22:             b[(l-1)*K+k, r] += 1
23: b[0,] ← spont                               # 1st line of b
24: return b, low, spont
```

4.2 Computation of G

Algorithm 2 Computation of G

G ← matrix(0, nrow=(1+M*K), ncol= (1+M*K)) # init of G
G[0,0] ← $T^{\max} - T_{\min}$
A ← $K * \delta$ # scope
α ← get_low_index(($T_{\min} - A$), T)
β ← get_low_index($T^{\max}$, T) - 1
for (i in α:β) **do**
    ti ← T[i]; l_1 ← neur[i]
    for (k in (1:K)) **do** # 1st row and 1st column of G
        $x_1 \leftarrow \min(T^{\max}, ti + k\,\delta)$
        $x_2 \leftarrow \max(T_{\min}, ti + (k-1)\,\delta)$
        $dx = x_1 - x_2$
        if ($dx > 0$) **then**
            G[0,l_1*K+k] += dx
            G[l_1*K+k,0] += dx
    for (j in (low[i]:(i-1))) **do** # inner part of G, $l_1 \neq l_2$
        tj ← T[j]; l_2 ← neur[j]
        for (k_1 in (1:K)) **do**
            for (k_2 in (1:K)) **do**
                $x_1 \leftarrow \min(T^{\max}, ti + k_1\,\delta, tj + k_2\,\delta)$
                $x_2 \leftarrow \max(T_{\min}, ti + (k_1-1)\,\delta), (tj + (k_2-1)\,\delta)$
                $dx = x_1 - x_2$
                if ($dx > 0$) **then**
                    G[l_1*K+k_1,l_2*K+k_2] += dx
                    G[l_2*K+k_2,l_1*K+k_1] += dx
    for (k_1 in 1:K) **do** # $l_1 == l_2$
        $x_1 \leftarrow \min(T^{\max}, ti + k_1\,\delta)$
        $x_2 \leftarrow \max(T_{\min}, ti + (k_1-1)\,\delta)$
        $dx = x_1 - x_2$
        if ($dx > 0$) **then** # diagonal part
            G[l_1*K+k_1,l_1*K+k_1] += dx
        for (k_2 in (k_1+1):K) **do** # extradiagonal part
            $x_2 \leftarrow \max(T_{\min}, ti + (k_2-1)\,\delta)$
            $dx = x_1 - x_2$
            if ($dx > 0$) **then**
                G[l_1*K+k_1,l_1*K+k_2] += dx
                G[l_1*K+k_2,l_1*K+k_1] += dx
return G

4.3 Computation of μ_A

Algorithm 5 Computation of μ_A

Usage: make_muA(S_0, delta, K, Tmin, $T^{\max}$)
Description: Computation for a given neuron
Input: S_0; set of ordered spikes of a given neuron
Input: delta; double, size of a bin
Input: Tmin; first value of the time interval
Input: $T^{\max}$; second value of the time interval
Output: v - integer vector of size K

```
v ← zeros(K)
for (k in 0:(K-1)) do
    SP ← restrict_2(S_0, T_min−(k + 1) δ, T^max)        # restriction
    for i in 0:(length(SP)-1) do
        imax ← which(SP<(SP[i]+delta))
        if (length(imax)) then
            v[k] ← max(v[k], 1+ imax[length(imax)-1]-i)
return v
```

Usage: compute_muA(param, data)
Description: Computation of μ_A
Input: param; instance of Parameter class
Input: data; instance of *data* class
Output: μ_A - vector of size (1+M*K)

```
from param, we get K, M, δ, Tmin, T^max
DN ← data.DataNeur()
μ_A allocated to size (1+MK)
μ_A[0] ← 1
for (r in 0:(M-1)) do
    μ_A[rK+1:(r+1)K]= make_muA(DN[r,], delta, K, Tmin, T^max)
return μ_A
```

4.4 Computation of μ_2

Algorithm 6 computation of couples of spikes (make_couple)

Usage: make_couple(M, data, delta)
Description: Computation of couples
Input: M; number of neurons
Input: data; data containing the DataNeur
Input: delta; double, size of a bin
Output: Mcouple; array of two-column matrices containing couples of spikes

```
Mcouple ← array(list(), dim=M)
for (r in 0:(M-1)) do
    SP ← list of spikes of neuron r
    N ← length of SP
    for (i in 0:(N-1)) do
        imax ← all indices > i of SP st SP < SP(i) + δ
        for (ip in imax) do
            adding the row (SP[i], SP[ip]) to Mcouple[r]
return Mcouple
```

Algorithm 7 Computation of μ_2

Usage: compute_mu2(param, data)
Description: Computation of μ_2
Input: param; instance of Parameter class
Input: data; instance of *data* class
Output: μ_2 - matrix of dimension (1+M*K)-by-M

from param, we get K, M, δ, $T_{\min}$, $T^{\max}$
DNrest ← data.restrictDN(M, $T_{\min}$, $T^{\max}$) # restriction of the DataNeur to $(T_{\min}, T^{\max}]$
$\zeta \leftarrow 0$ # matrix of dimension (1+M*K)-by-M
Mcouple ← make_couple(M, data, delta) # array of size M of 2-column matrices
for (r_1 in 0:(M-1)) **do**
    for (ic in 0:(length(Mcouple[r_1][[1]])-1)) **do**
        $T_1 \leftarrow$ Mcouple[r_1][ic,1]; $T_2 \leftarrow$ Mcouple[r_1][ic,2]
        for (r in 0:(M-1)) **do**
            $i_1 \leftarrow$ get_low_index(T_1, DNrest[r])
            $i_2 \leftarrow$ get_low_index(T_2+A, DNrest[r])
            for (ip in i_1:i_2) **do**
                τ= DNrest[r,ip+1]
                k←compute_k(τ, T_2, delta) # $\lfloor \frac{\tau - \delta - T_2}{\delta} \rfloor + 2$ or 1
                if (($k \geqslant 1$) and ($k \leqslant K$)) **then**
                    incrementing $\zeta[r_1 * K + k, r]$
$\mu_2 \leftarrow 2\,\zeta + \mu_1$
return μ_2

5 Lasso

5.1 Description of the problem

We follow the same procedure as [4] to stabilize the method and we should solve M Lasso problems of dimension $(1+M\,K)$, to estimate coefficients $(a^k_{j\to i})_{1\leqslant i,j\leqslant M, 1\leqslant k\leqslant K}$ and $(\nu_i)_{1\leqslant i\leqslant M}$.

$$\begin{aligned} \mathbf{a}^i_{BL} &= \arg\min_{\beta\in\mathbb{R}^{(1+MK)}} -2\,{}^T\mathbf{b}^i\,\beta + {}^T\beta\,\mathbf{G}\,\beta + 2\,{}^T\mathbf{d}^i\,|\beta| \\ &\quad \text{with } \mathbf{a}^i_{BL} = (\nu_i, a^1_{1\to i}, a^2_{1\to i}, \cdots, a^K_{1\to i}, a^1_{2\to i}, \cdots, a^K_{M\to i}) \end{aligned}$$

5.2 Verification of KKT conditions

We set $p = 1+MK$, we introduce $\mathbf{b} = \mathbf{b}^i$ and $\mathbf{d} = \mathbf{d}^i$ for the sake of simplicity to focus on a given Lasso problem (we have M similar Lasso problems) and we define the functional J from $\mathbb{R}^p$ to $\mathbb{R}$ such that

$$J(x) = \frac{1}{2}\,{}^T x\,\mathbf{G}\,x - {}^T\mathbf{b}\,x + {}^T\mathbf{d}\,|x|$$

The subdifferential of J is as follows for all $x \in \mathbb{R}^p$, where we set $*$ as the pointwise product such that $(d*v)_i = d_i\, v_i,\ \forall\, i \in [\![1,p]\!]$ and we use the fact that G is symmetric

$$\begin{aligned}
\partial J(x) &= \left\{ \frac{1}{2}({}^T\mathbf{G} + \mathbf{G})\, x - \mathbf{b} \right\} + \mathbf{d} * v = \{\mathbf{G}\, x - \mathbf{b}\} + \mathbf{d} * v \\
v_i &= \begin{cases} \{\mathrm{sgn}(x_i)\} & \text{if } x_i \neq 0 \\ [-1,1] & \text{if } x_i \equiv 0 \end{cases},\ \forall i \in [\![1,p]\!] \\
\text{where } (d*v)_i &= \begin{cases} \{d_i\, \mathrm{sgn}(x_i)\} & \text{if } x_i \neq 0 \\ [-d_i, d_i] & \text{if } x_i \equiv 0 \end{cases},\ \forall i \in [\![1,p]\!]
\end{aligned}$$

We know that $\overline{x}$ is the arg min of J in $\mathbb{R}^p$ if and only if

$$\begin{cases} \displaystyle\sum_{j=1}^{p} G_{ij}\, \overline{x}_j & -b_i + d_i\, \mathrm{sgn}(\overline{x}_i) = 0, \quad \forall i \text{ such that } \overline{x}_i \neq 0 \\ \displaystyle\sum_{j=1}^{p} G_{ij}\, \overline{x}_j & -b_i \in [-d_i, d_i] \qquad\qquad \forall i \text{ such that } \overline{x}_i \equiv 0 \end{cases}$$

Note that the sequel is a necessary condition (not sufficient) for $\overline{x}$ to be the arg min of J

$$\overline{x}_i \left(\sum_{j=1}^{p} G_{ij}\, \overline{x}_j - b_i + d_i\, \mathrm{sgn}(\overline{x}_i) \right) = 0\ \forall i \in [\![1,p]\!]$$

which rewrites, summing over the indices i

$$\overline{x}^T\, \mathbf{G}\, \overline{x} - \mathbf{b}^T\, \overline{x} + \mathbf{d}^T\, |\overline{x}| = 0$$

5.3 Another form of the Lasso problem

We set $p = (1 + M\, K)$.

For any $a \in \mathbb{R}^p$, we denote by $|a|$ a vector $\in \mathbb{R}^p$ such that

$$|a|_i = |a_i| \quad \forall i \in [\![1,p]\!].$$

and we set

$$\|a\|_1 = \sum_{i=1}^{p} |a|_i$$

We introduce the matrix $\Delta \in \mathcal{M}_p(\mathbb{R})$ whose diagonal is equal to d.

$$\Delta = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & d_p \end{pmatrix}$$

We set $y \in \mathbb{R}^p$ such that $y = \Delta\, a$. We assume that $d_i > 0$, $\forall\, i \in [\![1, p]\!]$, so $d^t\,|a| = \sum_{i=1}^{p} d_i |a_i| = \sum_{i=1}^{p} |d_i\, a_i| = \sum_{i=1}^{p} |(\Delta\, a)_i| = \|\Delta\, a\|_1 = \|y\|_1$.
We have

$$\begin{aligned} \frac{1}{2} a^t G a - b^t a + d^t |a| &= \frac{1}{2} a^t G a - b^t a + \|\Delta\, a\|_1 \\ &= \frac{1}{2} (\Delta^{-1} y)^t\, G\, (\Delta^{-1} y) - (\Delta^{-1} b)^t\, y + \|y\|_1 \end{aligned} \tag{1}$$

Most of the computational libraries solving Lasso problems solve them in the following form (where $D \in \mathcal{M}_{n,p}(\mathbb{R})$ (known), $\xi \in \mathbb{R}^n$ (known), $\lambda \geqslant 0$ (known), $\alpha \in \mathbb{R}^p$ (unknown))

$$\text{Find } \underset{\alpha \in \mathbb{R}^p}{\arg\min}\ \frac{1}{2} \|\xi - D\alpha\|_2^2 + \lambda\, \|\alpha\|_1 \tag{2}$$

We have

$$\frac{1}{2} \|\xi - D\alpha\|_2^2 + \lambda\, \|\alpha\|_1 = \frac{1}{2} (\alpha^t D^t D \alpha - 2 \xi^t D \alpha + \xi^t \xi) + \lambda\, \|\alpha\|_1 \tag{3}$$

So minimizing (3) is equivalent to minimize

$$\frac{1}{2} \alpha^t D^t D \alpha - \xi^t D \alpha + \lambda\, \|\alpha\|_1 \tag{4}$$

If G is symmetric positive definite, we can compute its Cholesky factorization, where $U \in \mathcal{M}_p(\mathbb{R})$ is an upper triangular matrix

$$G = U^t\, U$$

If we set $x \in \mathbb{R}^p$ such as $U^t\, x = b$ and $D = U\, \Delta^{-1}$, we have using (1)

$$\begin{aligned} \frac{1}{2} a^t G a - b^t a + d^t |a| &= \frac{1}{2} (\Delta^{-1} y)^t\, G\, (\Delta^{-1} y) - (\Delta^{-1} b)^t\, y + \|y\|_1 \\ &= \frac{1}{2} y^t (U\, \Delta^{-1})^t\, (U\, \Delta^{-1})\, y - (\Delta^{-1}\, U^t\, x)^t\, y + \|y\|_1 \\ &= \frac{1}{2} y^t (U\, \Delta^{-1})^t\, (U\, \Delta^{-1})\, y - x^t\, (U\, \Delta^{-1})^t\, y + \|y\|_1 \end{aligned}$$

So if G is symmetric positive definite, problem $(\mathcal{P})$ is equivalent to the following one

$$\begin{aligned} &\text{Find } \underset{y \in \mathbb{R}^p}{\arg\min}\ \frac{1}{2} \|x - D\, y\|_2^2 + \|y\|_1 \\ &\text{where } D = U\, \Delta^{-1} \text{ and } x \text{ such that } U^t\, x = b \end{aligned}$$

So to use standard libraries, by comparison with (4), we have different steps to solve the problem

- compute a Cholesky factorization of G giving the triangular upper matrix U (p-by-p matrix) such that $G = U^t\, U$
- find x (p-by-1 vector) as the solution of the linear system such that $U^t\, x = b$
- computing $\Delta^{inv} = \Delta^{-1}$ as a diagonal matrix (p-by-p matrix) whose coefficients are $\Delta^{inv}_{ii} = 1/d_i, \forall i \in [\![1, p]\!]$,
- set $D = U\, \Delta^{inv}$ (p-by-p matrix)
- set $\lambda = 1$
- once the Lasso problem (2) is solved returning y, the solution of our problem is given by $a = \Delta^{inv}\, y$

5.4 Active set method

We consider a sub-problem such that $J \subset [\![1, p]\!]$.

$$\mathbf{a}_J \quad = \quad \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{2}{}^T\beta\, \mathbf{G}\, \beta - {}^T\mathbf{b}\, \beta + {}^T\mathbf{d}\, |\beta|$$

Algorithm 8 Active Set

Input: : **b**, **G**, **d**, ε (for numerical precision)

J$\leftarrow \varnothing, a \leftarrow 0$
g $\leftarrow$ G a - b
repeat
  $j_0 \leftarrow \arg\max_{l \in J^c} |g_l|$
  J $\leftarrow J \bigcup j_0$
  $a_J \leftarrow$ solve Lasso for sub-problem
  g $\leftarrow$ G a - b
until $(g_{j_0} < d_{j_0} + \varepsilon)$
return a, J

6 Errors

6.1 Graph errors

We denote by $\widehat{a}^k_{j \to i}$ the estimates computed by Lasso, the exact value is denoted by $a^k_{j \to i}$. We denote by $|E|$ the number of elements of the set E (which is a set of indices or couples of indices).

We introduce the following graph errors $err_0 \in \mathbb{R}^2$, $err_1 \in \mathcal{M}^{2\times M}(\mathbb{R})$, $err_2 \in \mathcal{M}^{2\times K}(\mathbb{R})$, $err_3 \in \mathbb{R}^{2\times M\times K}$ (3D array) such that

$$\begin{aligned}
err_0 &= \sum_{i=1}^{M} err_1^i, \\
err_1^i &= (|E_1^i - \widehat{E}_1^i|, |\widehat{E}_1^i - E_1^i|) \quad \forall\, i \in [\![1, M]\!], \\
err_2^k &= (|E_2^k - \widehat{E}_2^k|, |\widehat{E}_2^k - E_2^k|), \quad \forall\, k \in [\![1, K]\!], \\
err_3^{i,k} &= (|E_3^{i,k} - \widehat{E}_3^{i,k}|, |\widehat{E}_3^{i,k} - E_3^{i,k}|), \quad \forall\, (i,k) \in [\![1, M]\!] \times [\![1, K]\!]
\end{aligned}$$

where

$$\begin{aligned}
\widehat{E}_1^i &= \{j \in [\![1, M]\!], \exists\, k \in [\![1, K]\!] \; s.t. \; |\widehat{a}_{j\to i}^k| > 0\}, \\
E_1^i &= \{j \in [\![1, M]\!], \exists\, k \in [\![1, K]\!] \; s.t. \; |a_{j\to i}^k| > 0\}, \\
\widehat{E}_2^k &= \left\{(i,j) \in [\![1, M]\!]^2 \; s.t. \; |\widehat{a}_{j\to i}^k| > 0\right\}, \qquad E_2^k = \left\{(i,j) \in [\![1, M]\!]^2 \; s.t. \; |a_{j\to i}^k| > 0\right\} \\
\widehat{E}_3^{i,k} &= \left\{j \in [\![1, M]\!] \; s.t. \; |\widehat{a}_{j\to i}^k| > 0\right\}, \qquad E_3^{i,k} = \left\{j \in [\![1, M]\!] \; s.t. \; |a_{j\to i}^k| > 0\right\}
\end{aligned}$$

6.2 Matrix errors

Estimate coefficients are stored in a matrix of dimension $(1+MK)-by-M$ denoted here by $BLest$. In some cases, we know the matrix of exact values, denoted here by $BLex$.

We recall the following definitions of matrix norms for a matrix M of dimension n-by-p.

$$\begin{aligned}
\|M\|_\infty &= \max_{i=0,\cdots,n-1} \left(\sum_{j=0}^{p-1} |m_{ij}| \right), \quad \|M\|_1 = \max_{j=0,\cdots,p-1} \left(\sum_{i=0}^{n-1} |m_{ij}| \right) \\
\|M\|_{fro} &= \sqrt{\sum_{i=0}^{n-1} \sum_{j=0}^{p-1} m_{ij}^2}, \qquad \|M\|_2 = \text{spectral radius}
\end{aligned}$$

For a vector v of dimension n,

$$\|v\|_\infty = \max_{i=0,\cdots,n-1} (|v_i|), \quad \|v\|_1 = \sum_{i=0}^{n-1} |v_i|, \quad \|v\|_2 = \sqrt{\sum_{i=0}^{n-1} v_i^2}$$

We define several matrix errors (notation $M_{1:,:}$ means we consider the entire matrix except its first row which contains spontaneous values, $M_{0,:}$ represents the first row of M)

$$\begin{aligned}
M &= BLest - BLex \\
M_\infty &= (\|M_{0,:}\|_\infty, \|M_{1:,:}\|_\infty) \\
M_1 &= (\|M_{0,:}\|_1, \|M_{1:,:}\|_1) \\
M_{fro} &= (\|M_{0,:}\|_{fro}, \|M_{1:,:}\|_{fro}) \\
M_2 &= (\|M_{0,:}\|_2, \|M_{1:,:}\|_2)
\end{aligned}$$

References

[1] Laurent Dragoni, Rémi Flamari, Karim Lounici, and Patricia Reynaud-Bouret. Large scale Lasso with windowed active set for convolutional spike sorting. arXiv:1906.12077, 2019.

[2] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[3] Niels Richard Hansen, Patricia Reynaud-Bouret, and Vincent and Rivoirard. Lasso and probabilistic inequalities for multivariate point processes. *Bernoulli*, 21(1):83–143, 2015.

[4] Régis C. Lambert, Christine Tuleau-Malot, Thomas Bessaih, Vincent Rivoirard, Yann Bouret, Nathalie Leresche, and Patricia Reynaud-Bouret. Reconstructing the functional connectivity of multiple spike trains using Hawkes models. *Journal of Neuroscience Methods*, 297:9–21, 2018.

[5] Cyrille Mascart, Alexandre Muzy, and Patricia Reynaud-Bouret. Discrete event simulation of point processes: Computational complexity analysis on sparse graphs. submitted to ACM Trans. Algor, 2020.

[6] Minh-Toan Nguyen. Information flow in plastic neural network. Training report, 2019.