

- Détection de dépendance entre M neurones
- On recherche, pour chaque neurone r , son intensité fonctionnelle $\lambda^r(t)$: c'est la probabilité d'un spike au temps t pour le neurone.
- Les intensités fonctionnelles sont modélisées par un processus de Hawkes
- N^r est l'ensemble des spikes du neurone r (train de spikes)
-

$$\begin{aligned}\lambda^r(t) &= \nu^r(t) + \sum_{l=1}^M \int_{-\infty}^{t^-} h_{l \rightarrow r}(t-u) dN_u^l \\ &= \nu^r(t) + \sum_{l=1}^M \sum_{T \in N^l, T < t} h_{l \rightarrow r}(t-T), \quad \forall t, \forall r\end{aligned}$$

- Pour $\lambda^r(t)$, on approche la partie spontanée ν^r et le noyau $h_{l \rightarrow r}$ par des fonctions constantes par morceaux
- On suppose que ν^r est une fonction constante.

Présentation du problème

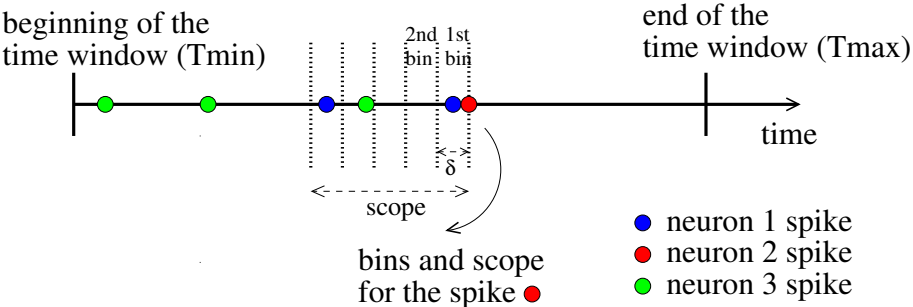


Figure: Exemple pour $M=3$ neurones

- On considère M neurones sur la fenêtre de temps (T_{min}, T_{max}].
- L'influence de chaque spike sur les autres n'excède pas une portée A , divisée en K boîtes (ou K bins).

Données d'entrée - DataNeur

Les données d'entrée sont les temps de spikes pour chaque neurone.

- Deux possibilités: *DataNeur* ou *DataSpike*
- Un DataNeur est une matrice de dimension M -by- $(1+\max(N_s))$, où M est le nombre de neurones et N_s le nombre de spikes par neurones
- Chaque ligne de la matrice contient les spikes d'un neurone correspondant à la ligne, triés dans l'ordre croissant
- On complète par des "0" s'il n'y a pas de spike.
- Exemple: 3 neurones, $N_s = [2, 1, 3]$, le DataNeur obtenu est la matrice 3-by-4 telle que:

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

- Un DataSpike est un tableau de dimension 2-by- N_{tot} , où N_{tot} est le nombre total de spikes, indépendamment des neurones.
- La première ligne contient les spikes par ordre croissant, quel que soit le neurone
- La deuxième ligne contient le numéro de neurone correspondant au spike de la 1ère ligne
- Exemple: 3 neurones, $N_s = [2, 1, 3]$, le DataSpike obtenu est la matrice 2-by-6 telle que (la numérotation commence à 1)

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

Classe DataSpike

- Attributs de la classe
 - `_T`: tableau des spikes (`double *`)
 - `_neur`: tableau des numéros de neurones (`unsigned int *`)
- Diagramme de la classe

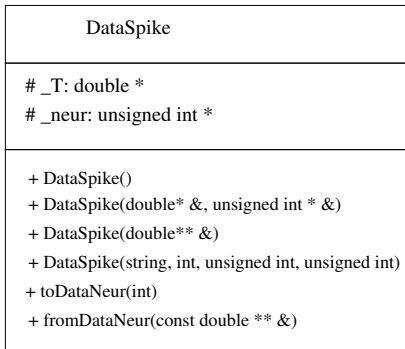


Figure: Diagramme de la classe *DataSpike* en C++

Notations

- K est le nombre de boîtes (bins).
- On note

$$\begin{aligned}\mathbb{I}_k &= ((k-1)\delta, k\delta], \quad \forall 1 \leq k \leq K \\ \varphi_k(t) &= \mathbb{1}_{\mathbb{I}_k}(t), \quad \forall 1 \leq k \leq K\end{aligned}$$

- On introduit l'opérateur ψ^I qui transforme une fonction v en une autre fonction, tel que, $\forall I$

$$\psi_t^I(v) = \int_{-\infty}^{t^-} v(t-u) dN_u^I = \sum_{T \in N^I, T < t} v(t-T), \quad \forall v, \forall t, \forall I$$

D'où, pour $v = \varphi_k$

$$\psi_t^I(\varphi_k) = \int_{u < t} \mathbb{1}_{\mathbb{I}_k}(t-u) dN_u^I = \sum_{T < t, T \in N^I} \mathbb{1}_{\mathbb{I}_k}(t-T)$$

Matrices du problème

- b et d sont de taille $(M \cdot K + 1)$ -by- M
- G est de taille $(M \cdot K + 1)$ -by- $(M \cdot K + 1)$
- On note par *spont* ce qui correspond à la partie spontanée (1ère ligne et/ou 1ère colonne des matrices)

Numérotation de b

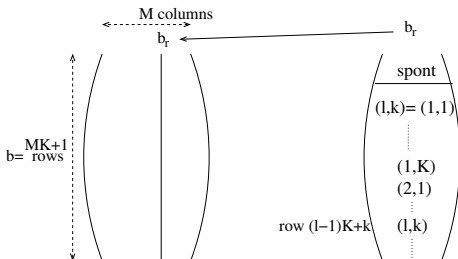
- Numérotation: pour le neurone r donné, pour chaque neurone $1 \leq l \leq M$ influençant r , à l fixé, on considère toutes les boîtes d'indice k ($1 \leq k \leq K$).

$$b_{spont}^r \leftrightarrow b[0, r] \quad 0 \leq r \leq M-1$$

...

$$b_{(l,k)}^r \leftrightarrow b[(l-1)K + k, r]$$

...



- Partie spontanée pour le neurone r

$$b_{spont}^r = \int_{T_{\min}}^{T^{\max}} dN_t^r = \# \{ T : T_{\min} < T \leq T^{\max}, T \in N^r \}$$

- Contribution du neurone l sur le neurone r , pour la boîte (bin) k

$$b_{(l,k)}^r = \int_{T_{\min}}^{T^{\max}} \psi_t^l(\varphi_k) dN_t^r$$

Calcul de la matrice $b - 0$

- Dans ce qui suit, cnt est un vecteur de taille M , contenant la partie spontanée de b

$$cnt^r = b_{spont}^r = \int_{T_{\min}}^{T_{\max}} dN_t^r$$

- low désigne un tableau d'indices tel que $low(i)$ est le plus petit indice dans le tableau des spikes tel que

$$j = low(i) \Rightarrow |T_j - T_i| \leq A = K \delta, \quad i, j \in \llbracket 1, N_{tot} \rrbracket$$

- $get_k(x, \delta)$ retourne l'entier $k \geq 1$ tel que $x \in \mathbb{I}_k = ((k-1)\delta, k\delta]$
- Dans les algos qui suivent, la numérotation des matrices et vecteurs commencent à 1 (comme en R).

Calcul de la matrice b - I

Algorithm 1 Computation of b

```
1:  $T \leftarrow DS[1,]$  # first row of DS: all spike values
2:  $neur \leftarrow DS[2,]$  # second row of DS: neuron numbers
3:  $Ntot \leftarrow \text{length}(T)$  # total number of spikes
4:  $A \leftarrow K * \delta$  # scope, K bins of size  $\delta$ 
5:  $b \leftarrow \text{matrix}(0, \text{nrow}=(1+M*K), \text{ncol}=M)$  # init of b
6:  $low \leftarrow \text{vector}(\text{mode}='integer', \text{length}=Ntot)$  # init of low - for the State
7:  $cnt \leftarrow \text{vector}(\text{mode}='integer', \text{length}=M)$  # init of cnt - for the algo
8:  $ilow \leftarrow 1$ 
9:  $eps \leftarrow 1.e-12$  # for numerical precision
10: for ( $i=1:Ntot$ ) do # loop over spikes
11:    $t \leftarrow T[i]; r \leftarrow neur[i]$ 
12:   while ( $|T[ilow] - t| > A$ ) do
13:      $ilow \leftarrow ilow+1$ 
14:   end while
```

Calcul de la matrice b - II

```
15:   low[i] ← ilow
16:   if ( $T_{\min} < t \leq T^{\max}$ ) then
17:       cnt[r] ← cnt[r]+1
18:       for (j=ilow:(i-1)) do
19:           if ( $|t - T[j]| < eps$ ) then
20:               break
21:           end if
22:           k ← get_k(t-T[j], delta)
23:           l ← neur[j]
24:           b[(l-1)*K+k+1, r] += 1
25:       end for
26:   end if
27: end for
28: b[1,] ← cnt                                     # 1st line of b
29: return b, low, cnt
```

- Partie spontanée

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

- Première ligne ou colonne

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

- Contribution du neurone l_1 , pour la boîte (bin) k_1 , du neurone l_2 pour la boîte (bin) k_2

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

Calcul de la matrice $G - 0$



$$G_{spont, (l, k)} \simeq \sum_{\substack{\theta \in N^l, \\ (T_{\min} - A) \leq \theta < T^{\max}}} (\min(T^{\max}, k\delta + \theta) - \max(T_{\min}, (k-1)\delta + \theta))$$

$$G_{(l_1, k_1), (l_2, k_2)} \simeq \sum_{\substack{\tau \in N^{l_1}, \theta \in N^{l_2}, \\ (T_{\min} - A) \leq \tau < T^{\max}, \\ (T_{\min} - A) \leq \theta < T^{\max}}} \int_{T_{\min}}^{T^{\max}} \mathbb{1}_{(\mathbb{I}_{k_1} + \tau) \cap (\mathbb{I}_{k_2} + \theta) \cap]T_{\min}, T^{\max}]}(t) dt$$

- $get_low_index(x, T)$ renvoie le plus petit entier i tel que

$$T_i > x \text{ et } 1 \leq i \leq (T.size() + 1)$$

Calcul de la matrice G - I

Algorithm 2 Computation of G

```
1:  $G \leftarrow \text{matrix}(0, \text{nrow}=(1+M*K), \text{ncol}=(1+M*K))$  # init of  $G$ 
2:  $G[1,1] \leftarrow T^{\max} - T_{\min}$ 
3:  $A \leftarrow K * \delta$  # scope
4:  $\text{depart} \leftarrow \text{get\_low\_index}((T_{\min} - A), T)$ 
5:  $\beta \leftarrow \text{get\_low\_index}(T^{\max}, T) - 1$ 
6: for ( $i$  in  $\text{depart}:\beta$ ) do
7:    $ti \leftarrow T[i]; l_1 \leftarrow \text{neur}[i]$ 
8:   for ( $k$  in  $(1:K)$ ) do # 1st row and 1st column of  $G$ 
9:      $x_1 \leftarrow \min(T^{\max}, ti + k \delta)$ 
10:     $x_2 \leftarrow \max(T_{\min}, ti + (k - 1) \delta)$ 
11:     $dx = x_1 - x_2$ 
12:    if ( $dx > 0$ ) then
13:       $G[1, (l_1-1)*K+k+1] += dx$ 
14:       $G[(l_1-1)*K+k+1, 1] += dx$ 
15:    end if
16: end for
```

Calcul de la matrice G - II

```
17:  for (j in (low[i]:(i-1))) do                                     # inner part of G,  $l_1 \neq l_2$ 
18:       $t_j \leftarrow T[j]$ ;  $l_2 \leftarrow \text{neur}[j]$ 
19:      for ( $k_1$  in (1:K)) do
20:          for ( $k_2$  in (1:K)) do
21:               $x_1 \leftarrow \min(T^{\max}, t_i + k_1 \delta, t_j + k_2 \delta)$ 
22:               $x_2 \leftarrow \max(T_{\min}, t_i + (k_1 - 1) \delta, (t_j + (k_2 - 1) \delta)$ 
23:               $dx = x_1 - x_2$ 
24:              if ( $dx > 0$ ) then
25:                   $G[(l_1-1)*K+k_1+1, (l_2-1)*K+k_2+1] += dx$ 
26:                   $G[(l_2-1)*K+k_2+1, (l_1-1)*K+k_1+1] += dx$ 
27:              end if
28:          end for
29:      end for
30:  end for
31:  for ( $k_1$  in 1:K) do                                             #  $l_1 == l_2$ 
32:       $x_1 \leftarrow \min(T^{\max}, t_i + k_1 \delta)$ 
33:       $x_2 \leftarrow \max(T_{\min}, t_i + (k_1 - 1) \delta)$ 
34:       $dx = x_1 - x_2$ 
35:      if ( $dx > 0$ ) then                                           # diagonal part
36:           $G[(l_1-1)*K+k_1+1, (l_1-1)*K+k_1+1] += dx$ 
37:      end if
```


Calcul de la matrice G - III

```
38:         for ( $k_2$  in ( $k_1+1$ ): $K$ ) do                                     # extradiagonal part
39:              $x_2 \leftarrow \max(T_{\min}, ti + (k_2 - 1) \delta)$ 
40:              $dx = x_1 - x_2$ 
41:             if ( $dx > 0$ ) then
42:                  $G[(l_1-1)*K+k_1+1, (l_1-1)*K+k_2+1] += dx$ 
43:                  $G[(l_1-1)*K+k_2+1, (l_1-1)*K+k_1+1] += dx$ 
44:             end if
45:         end for
46:     end for
47: end for
48: return  $G$ 
```

- Code *neuro-stat* existant mais manque de documentation et première version des algos pour calculer b , G et d
- Utilisation de la bibliothèque C++ *armadillo* dans la nouvelle version
- Intégration dans R a priori simple avec le package *RcppArmadillo*
- Performances assez semblables entre C++ *armadillo* seul et *RcppArmadillo*
- Une variante du calcul de G a été faite, où les indices (k_1, k_2) ont été calculés (pour $G_{(l_1, k_1), (l_2, k_2)}$)
- d pas encore implémenté dans la nouvelle version (plus compliqué)

Temps calcul des matrices pour un exemple

On considère l'intervalle de temps $(0, 600]$ et $\sum_{i=1}^M \nu_i = 15000$. On a les résultats suivants:

M	ν	$\sum_{i=1}^M \nu_i$	spike number per neuron on $(0, 600]$	Elapsed time for G (our method)	Elapsed time for b (our method)	neuro-stat time
100	150	15000	90000	1243.33	125.935	1188.961
200	75	15000	45000	1247.23	128.933	2302.629
500	30	15000	18000	1998.98	129.578	5521.465
1000	15	15000	9000	3049.78	190.101	10468.655

Table: Temps calcul pour $\sum_{i=1}^M \nu_i = 15000$

Temps calcul des matrices

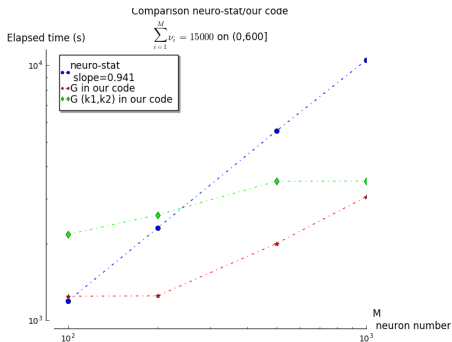


Figure: Comparaison des temps calcul pour G (loglog)

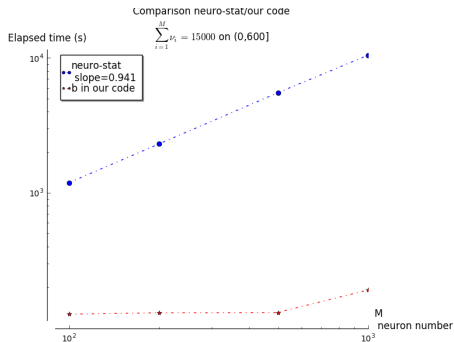


Figure: Comparaison des temps calcul pour b (loglog)

Calcul de d : matrice μ_2 et vecteur μ_A

- d et μ_2 sont des matrices de taille $(1+MK)$ -by- M (comme b)
- μ_A est un vecteur de taille $(1+MK)$
- On pose $\gamma = 3$ et $c_{\log} = \log((1 + MK)M)$
-

$$d_r = \sqrt{2\gamma c_{\log} \mu_{2r}} + \frac{\gamma}{3} c_{\log} \mu_A, \quad \forall r \in \llbracket 1, 1 + MK \rrbracket$$

- Partie spontanée pour le neurone r dans μ_2

$$(\mu_2)_{spont}^r = \int_{T_{\min}}^{T_{\max}} dN_t^r$$

Contribution du neurone l sur r , pour la boîte (bin) k

$$\begin{aligned} (\mu_2)_{(l,k)}^r &= \int_{T_{\min}}^{T_{\max}} \left(\psi_t^l(\varphi_k) \right)^2 dN_t^r \\ (\mu_A)_{spont} &= 1, \quad (\mu_A)_{(l,k)} = \sup_{t \in (T_{\min}, T_{\max}]} |\psi_t^l(\varphi_k)| \end{aligned}$$

- lassoshooting (R)
- LARS dans scikit-learn, mlpack
- Spams
- code inclus dans *neuro-stat*

Pour chaque neurone, il faut résoudre un problème du Lasso, à savoir trouver $a_r \in \mathbb{R}^{\text{dim}}$, où b_r et d_r représentent la r -ième colonne de b et d , respectivement,

$$\begin{aligned} \text{dim} &= M K + 1 \\ \text{trouver } \arg \min_{a_r \in \mathbb{R}^{\text{dim}}} & \frac{1}{2} a_r^T G a_r - b_r^T a_r + d_r^T |a_r| \end{aligned}$$

a_r est la r -ième colonne de la matrice a de dimension dim-by-M .

lasso_shooting algorithm

Algorithm 3 lasso_shooting

Usage: lasso_shooting(G, b, d, a_0 , nitmax=10000, eps=1e-8)

```
1: M ← ncol(b)
2: Dim ← nrow(b)
3: a ← matrix(nrow=Dim, ncol=M)           # array containing Lasso solutions
4: nit ← vector(length=M, mode='numeric') # array containing iteration
    number
5: for (r in 1:M) do
6:   a[,r] ← lasso_shooting1(G, b[,r], d[,r],  $a_0$ , nitmax, eps) # a[,r] is of
    dimension Dim
7: end for
8: return a, nit
```

lasso_shooting algorithm

lasso_shooting1 permet de résoudre un seul problème du Lasso dans \mathbb{R}^{dim} .

Algorithm 4 lasso_shooting1

Usage: lasso_shooting1(G, b, d, a₀, nitmax=10000, eps=1e-8)

```
1: Dim ← length(b); a ← a0; aold ← a # to define aold
2: J ← (function(a) 1/2*t(a)*G*a - t(b)*a + d*abs(a))
3: while (((|J(a) - J(aold)| > eps) & (nit ≤ nitmax) ) |(nit==0)) do
4:   aold ← a
5:   for (i in 1:Dim) do
6:     S ← G[i,] * a - G[i,i]*a[i] - b[i]
7:     if (G[i,i] ≤ 0) then
8:       warning ('unexpected value for G(i,i)')
9:     end if
10:    if ((-S - d[i]) > 0) then
11:      a[i] ←  $\frac{-S - d[i]}{G[i, i]}$ 
12:    end if
13:    if ((-S + d[i]) < 0) then
14:      a[i] ←  $\frac{-S + d[i]}{G[i, i]}$ 
15:    else
16:      a[i] ← 0
17:    end if
18:  end for
19:  nit ← nit+1
20: end while
21: return a, nit
```

- Mise en œuvre du calcul de d
- Algo de Lasso shooting à améliorer (si G non sdp)
- Initialisation du lasso shooting
- Implémenter active set method
- Exemples de validation complets