Contexte/ Objectifs

- Détection de dépendance entre 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
- $\mathcal{N}^r$ est l'ensemble des spikes du neurone r (processus ponctuel)
- $$\begin{aligned}\lambda^{(r)}(t) &= \nu^{(r)}(t) + \sum_{l=1}^{M} \int_{-\infty}^{t^-} h^{l,r}(t-u)\, dN_u^l \\ &= \nu^{(r)}(t) + \sum_{l=1}^{M} \sum_{T \in \mathcal{N}^l,\, T<t} h^{l,r}(t-T),\ \forall t,\ \forall r\end{aligned}$$
- Pour $\lambda^{(r)}(t)$, on approche la partie spontanée et le noyau par des fonctions constantes par morceaux
- On suppose que $\nu^{(r)}$ est une fonction constante.

Présentation du problème

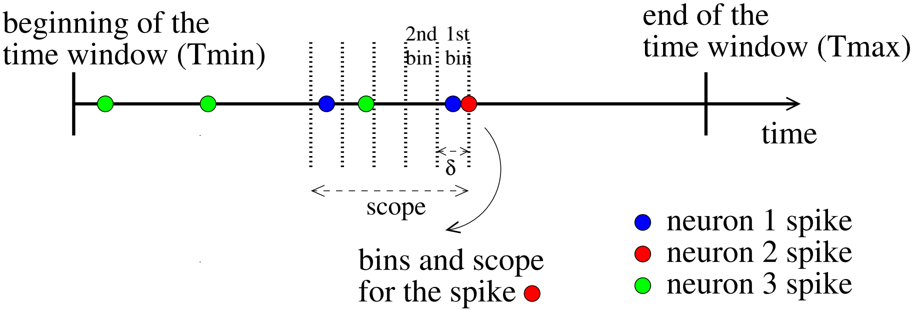


Figure: Exemple pour M=3 neurones

- 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(Ns)), où M est le nombre de neurones et Ns 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, $Ns = [2, 1, 3]$, le DataNeur obtenu est la matrice 3-by-4 telle que:

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

Données d'entrée - DataSpike

- Un DataSpike est un tableau de dimension 2-by-$Ntot$, où $Ntot$ 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, $Ns = [2, 1, 3]$, 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

DataSpike
_T: double * # _neur: unsigned int *
+ DataSpike() + DataSpike(double* &, unsigned int * &) + DataSpike(double** &) + DataSpike(string, int, unsigned int, unsigned int) + toDataNeur(int) + fromDataNeur(const double ** &)

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

Notations

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

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

- On introduit l'opérateur $\psi^{(l)}$ qui transforme une fonction v en une autre fonction, tel que, $\forall l$

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

D'où, pour $v = \varphi_k$

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

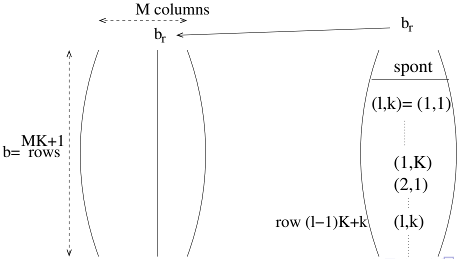
Matrices du problème

- b et d sont de taille (M*K+1)-by-M
- G est de taille (M*K+1)-by-(M*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 $1 \leqslant l \leqslant M$ influençant r, à l fixé, on considère toutes les boîtes d'indice k ($1 \leqslant k \leqslant K$).
-

$$
\begin{aligned}
b^r_{spont} &\leftrightarrow b[0, r] \quad 0 \leqslant r \leqslant M-1 \\
&\dots \\
b^r_{(l,k)} &\leftrightarrow b[(l-1)K + k, r] \\
&\dots
\end{aligned}
$$



Matrice b

- Partie spontanée pour le neurone r

$$b_{spont}^{(r)} = \int_{T_{\min}}^{T^{\max}} dN_t^r = \# \left\{ T : T_{\min} < T \leqslant T^{\max}, T \in \mathcal{N}^{(r)} \right\}$$

- 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 taillle M, contenant la partie spontanée de b

$$cnt^r = b^{(r)}_{spont} = \int_{T_{\min}}^{T^{\max}} dN^r_t$$

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

$$j = low(i) \Rightarrow |T_j - T_i| \leqslant A = K\,\delta, \quad i,j \in [\![1, Ntot]\!]$$

- $get\_k(x, \delta)$ retourne l'entier k$\geqslant$ 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

```
T ← DS[1,]                                          # first row of DS: all spike values
neur ← DS[2,]                                       # second row of DS: neuron numbers
Ntot ← length(T)                                    # total number of spikes
A ← K ∗ δ                                           # scope, K bins of size δ
b ← matrix(0, nrow=(1+M*K), ncol= M)                # init of b
low ← vector(mode='integer', length=Ntot)           # init of low - for the State
cnt ← vector(mode='integer', length=M)              # init of cnt - for the algo
ilow ← 1
eps ← 1.e-12                                        # for numerical precision
for (i=1:Ntot) do                                   # loop over spikes
    t ← T[i]; r ← neur[i]
    while (|T[ilow] − t| > A) do
        ilow ← ilow+1
    end while
```

Calcul de la matrice b - II

```
        low[i] ← ilow
        if (T_min < t ⩽ T^max) then
            cnt[r] ← cnt[r]+1
            for (j=ilow:(i-1)) do
                if (|t − T[j]| < eps) then
                    break
                end if
                k ← get_k(t-T[j], delta)
                l ← neur[j]
                b[(l-1)*K+k+1, r] += 1
            end for
        end if
    end for
    b[1,] ← cnt                                   # 1st line of b
    return b, low, cnt
```

Matrice G

- 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

- $$G_{spont,(l,k)} \simeq \sum_{\substack{\theta \in \mathcal{N}^{(l)}, \\ (T_{\min} - A) \leqslant \theta < T^{\max}}} \left(\min\left(T^{\max}, k\,\delta + \theta\right) - \max\left(T_{\min}, (k-1)\,\delta + \theta\right)\right)$$
$$G_{(l_1,k_1),(l_2,k_2)} \simeq \sum_{\substack{\tau \in \mathcal{N}^{(l_1)}, \theta \in \mathcal{N}^{(l_2)}, \\ (T_{\min} - A) \leqslant \tau < T^{\max}, \\ (T_{\min} - A) \leqslant \theta < T^{\max}}} \int_{T_{\min}}^{T^{\max}} \mathbb{1}_{(\mathbb{I}_{k_1} + \tau) \cap (\mathbb{I}_{k_2} + \theta) \cap]T_{\min}, T^{\max}]}(t)\, dt$$
- $get\_low\_index(x, T)$ renvoie le plus petit entier i tel que

$$T_i > x \ \text{ et } 0 < i < T.size()$$

Calcul de la matrice G - I

Algorithm 2 Computation of G

G $\leftarrow$ matrix(0, nrow=(1+M*K), ncol= (1+M*K)) # init of G
G[1,1] $\leftarrow T^{\max} - T_{\min}$
A $\leftarrow K * \delta$ # scope
depart $\leftarrow$ get_low_index(($T_{\min} - A$), T)
$\beta \leftarrow$ get_low_index($T^{\max}$, T) - 1
for (i in depart:β) **do**
  ti $\leftarrow$ T[i]; $l_1 \leftarrow$ neur[i]
  for (k in (1:K)) **do** # 1st row and 1st column of G
    $x_1 \leftarrow \min\left(T^{\max}, ti + k\,\delta\right)$
    $x_2 \leftarrow \max\left(T_{\min}, ti + (k-1)\,\delta\right)$
    $dx = x_1 - x_2$
    if $(dx > 0)$ **then**
      G[1,(l_1-1)*K+k+1] += dx
      G[(l_1-1)*K+k+1,1] += dx
    end if
  end for

Calcul de la matrice G - II

```
for (j in (low[i]:(i-1))) do                              # inner part of G, l_1 ≠ 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 ← min(T^max, ti + k_1 δ, tj + k_2 δ)
            x_2 ← max(T_min, ti + (k_1 − 1) δ), (tj + (k_2 − 1) δ)
            dx = x_1 − x_2
            if (dx > 0) then
                G[(l_1-1)*K+k_1+1,(l_2-1)*K+k_2+1] += dx
                G[(l_2-1)*K+k_2+1,(l_1-1)*K+k_1+1] += dx
            end if
        end for
    end for
end for
for (k_1 in 1:K) do                                       # l_1 == l_2
    x_1 ← min(T^max, ti + k_1 δ)
    x_2 ← max(T_min, ti + (k_1 − 1) δ)
    dx = x_1 − x_2
    if (dx > 0) then                                      # diagonal part
        G[(l_1-1)*K+k_1+1,(l_1-1)*K+k_1+1] += dx
    end if
```

Calcul de la matrice G - III

```
        for (k2 in (k1+1):K) do                                # extradiagonal part
            x2 ← max(T_min, ti + (k2 − 1) δ)
            dx = x1 − x2
            if (dx > 0) then
                G[(l1-1)*K+k1+1,(l1-1)*K+k2+1] += dx
                G[(l1-1)*K+k2+1,(l1-1)*K+k1+1] += dx
            end if
        end for
    end for
end for
return G
```

Mise en œuvre

- 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

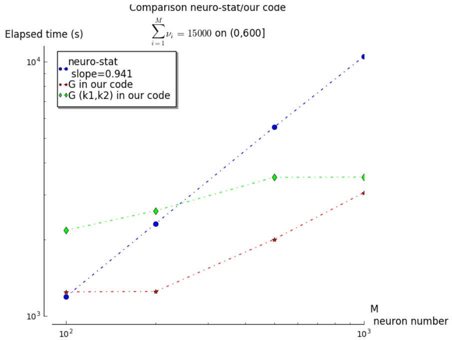


Figure: Comparaison des temps calcul pour G (loglog)

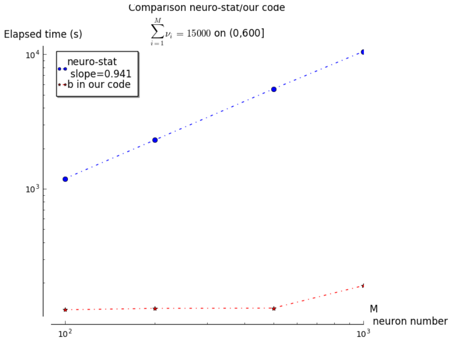


Figure: Comparaison des temps calcul pour b (loglog)

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 $c_{log} = \log((1+MK)M)$
- $$d_r = \sqrt{2\gamma\, c_{log}\, {\mu_2}_r} + \frac{\gamma}{3} c_{log}\, \mu_A, \ \forall r \in [\![1, 1+MK]\!]$$
- Partie spontanée pour le neurone r dans μ_2

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

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

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

Solveurs Lasso

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

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

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

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)

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

lasso_shooting algorithm

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

Algorithm 4 lasso_shooting1

Usage: lasso_shooting1(G, b, d, a_0, nitmax=10000, eps=1e-8)

Dim ← length(b); a ← a_0; aold ← a # to define aold
J ← (function(a) 1/2*t(a)*G*a - t(b)*a + d*abs(a))
while ((($|J(a) - J(aold)| > eps$) & ($nit \leqslant nitmax$)) |(nit==0)) **do**
    aold ← a
    for (i in 1:Dim) **do**
        S ← G[i,] * a - G[i,i]*a[i] - b[i]
        if (G[i,i]$\leqslant$ 0) **then**
            warning ('unexpected value for G(i,i)!')
        end if
        if $((-S - d[i]) > 0)$ **then**
            a[i] ← $\dfrac{-S - d[i]}{G[i,i]}$
        end if
        if $((-S + d[i]) < 0)$ **then**
            a[i] ← $\dfrac{-S + d[i]}{G[i,i]}$
        else
            a[i] ← 0
        end if
    end for
    nit ← nit+1
end while
return a, nit

Perspectives

- 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