Examen M1 C++ & Biomod, UE 4TBI806U

Date de l'épreuve : jeudi 1 avril 2021, 13h30

L'examen a lieu en binôme mais si certains préfèrent travailler seul c'est également possible à condition de m'en informer. Si vous êtes un nombre impair un trinôme pourra être accepté. La séance en distanciel du **01/04/2021** permettra de vous donner les bases et de vous aider à répondre au cahier des charges du projet. Celui-ci devra être renvoyé par email **avant le 15/04/2021 minuit**.

Chaque groupe devra renvoyer :

- un fichier C++ **commenté pour chaque commande demandée** dans le sujet
- les fichiers R, Python ou Gnuplot ou les méthodes vous ayant permis de visualiser vos résultats
- un **document pdf** de 4 ou 5 pages expliquant pour chaque fonction comment vous avez procédé ou comment vous auriez procédé si vous n'avez pas eu le temps de finir l'implémentation

Pas d'envoi sous forme de fichier compressé merci !

Vous pourrez ajouter des figures à ce document pdf. Chaque document renvoyé devra comporter en outre **les noms et prénoms des auteurs**.

J'aurai beaucoup de codes sources à traiter donc pensez bien à rappeler au début de chacun en commentaire vos **nom**, **prénom** et **nom de commande**. Exemple :

```
// Nom, Prénom
// commande extract
... votre code ...
```

Comme cet examen devrait se dérouler en conférence audio vous pouvez éventuellement me poser des questions d'ordre général durant la séance. Si vous préférez des échanges privés vous pouvez passer par les emails. Je serai aussi réactif que possible.

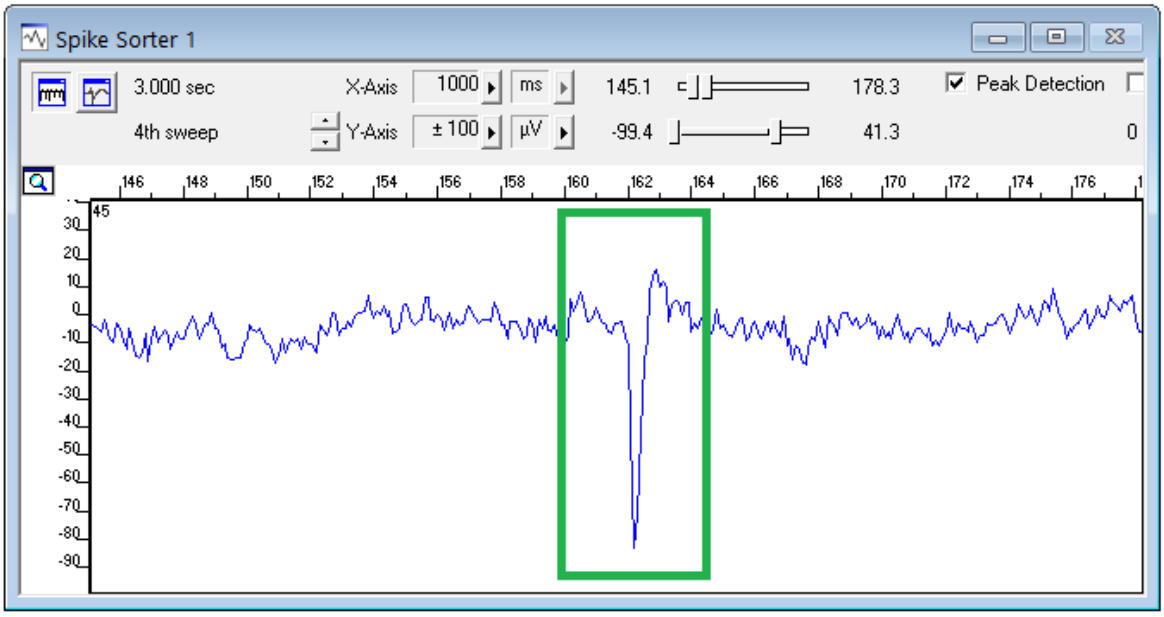
---

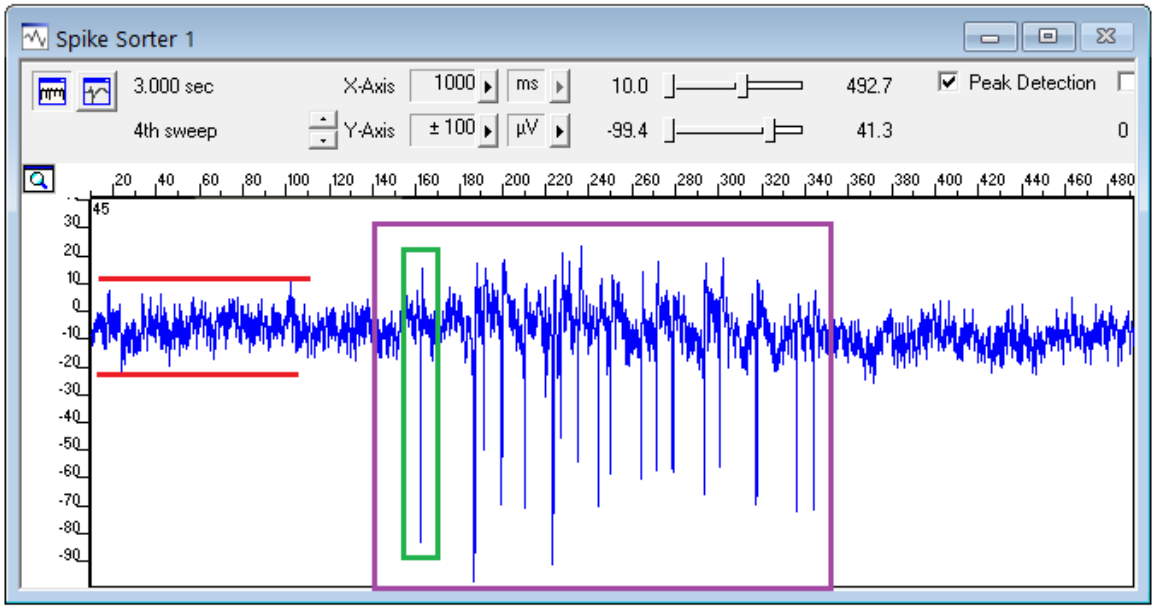
1 Téléchargement et description des données

Le fichier **data_mcs.txt** sur lequel travailler est téléchargeable ici :

https://filesender.renater.fr/?s=download&token=def8f394-9e71-4d17-8a4f-eba4bb886335

Ce fichier au format txt contient 60 secondes de l'enregistrement d'une culture de neurones réalisée en 3 points différents de sa surface. La colonne 1 contient l'instant de chaque enregistrement en ms. Les colonnes 2, 3 et 4 contiennent la valeur de potentiel enregistrée à chaque instant. Vous avez donc 3 séries temporelles exprimées en μV. Les neurones enregistrés émettent spontanément des potentiels d'action isolés (*spikes*) ou bien des séries de potentiels d'action rapprochés dans le temps (*bursts*) qui se traduisent par des brusques variations de la ligne de base. Les 2 figures ci-dessous illustrent respectivement un *spike* et un *burst*. L'enveloppe du bruit est représentée en rouge (les signaux extracellulaires sont souvent très bruités), des *spikes* sont encadrés en vert et un *burst* est entouré en violet. Le temps en abscisse est en ms et l'amplitude en ordonnée en μV :





L'objectif de votre travail est d'écrire plusieurs fonctions pouvant être appelées depuis la ligne de commande comme de simple commandes bash ou DOS et qui permettent de traiter et d'analyser ces signaux.

2 Commande *extract*

Cette commande doit permettre d'extraire depuis le fichier data_mcs.txt l'un des canaux qu'ils contient en l'appelant par son numéro et de créer un nouveau fichier ne contenant plus que 2 colonnes, 1 pour le temps et 1 pour le canal. La commande doit pouvoir s'exécuter comme suis depuis la ligne de commande :

```
extract data_mcs.txt channel_2.txt 2
```

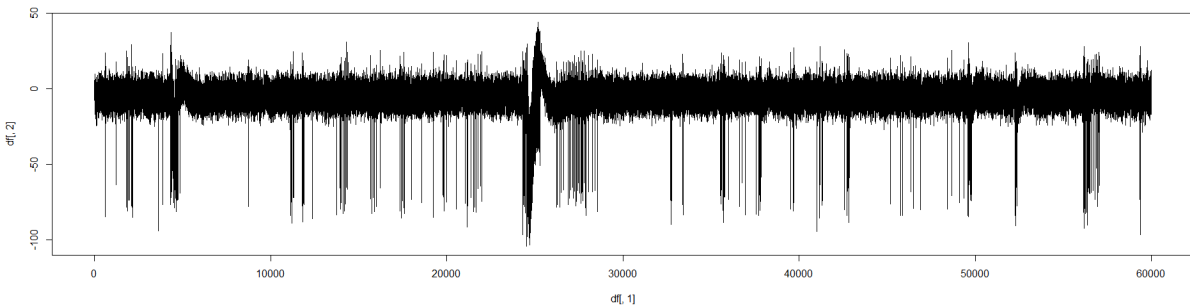
Cette commande doit donc à partir de data_mcs.txt créer un fichier channel_2.txt qui contiendra les données sur 2 colonnes (temps et potentiel) et avec un entête commençant par un %. Le nom de la deuxième colonne sera **channel_X** ou **X** sera le numéro de canal composé passé comme 3ème argument. Vous devriez donc obtenir un fichier channel_2.txt qui commence comme suis :

```
%time channel_2
0.000000 -4.580000
0.100000 -2.080000
0.200000 -0.420000
0.300000 -3.750000
...
```

Vous pourrez utiliser Python ou R pour visualiser vos résultats. Par exemple avec R et le code suivant :

```
df=read.table("channel_2.txt",header=T)
plot(df[,1],df[,2],type="l") # temps en ms
```

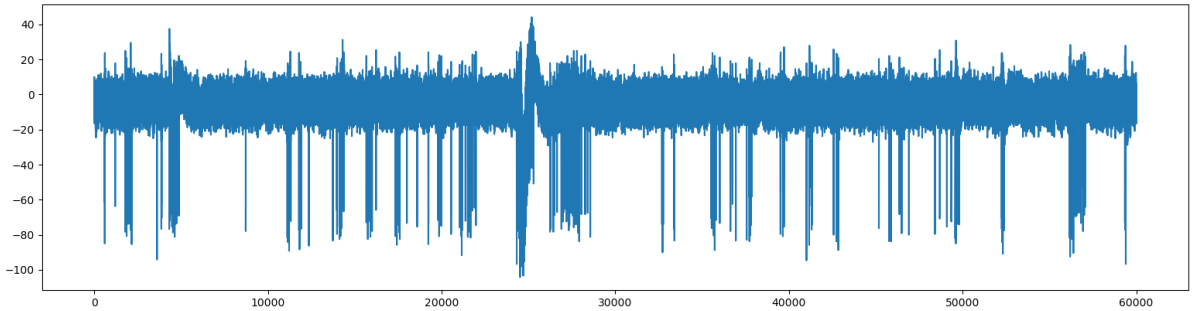
Vous obtiendrez :



Et avec Python et les commandes :

```
from pylab import *
from numpy import *
res_1=loadtxt("channel_2.txt",skiprows=1)
plot(res_1[:,0],res_1[:,1]) # temps en ms
show()
```

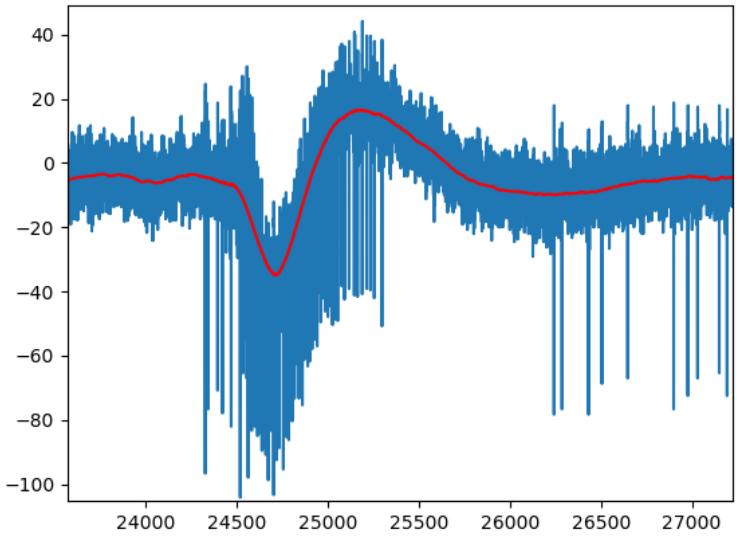
Vous obtiendrez :



Par la suite je vous proposerai des copies d'écran réalisées avec Python pour visualiser les différents résultats.

3 Commande *m_average*

Lorsque l'activité électrique est soutenue, surtout lors de bursts importants, on peut assister à des déviations de la ligne de base du signal comme c'est illustré sur la figure ci-dessous :



Le premier traitement que vous devrez effectuer sera donc d'écrire une commande m_average qui prendra en argument le fichier source, le nom du fichier de sauvegarde et un paramètre win_size et qui réalisera une moyenne mobile du signal qui sera sauvegardée dans un fichier. Donc la commande :

```
m_average channel_2.txt ma_channel_2.txt 1000
```

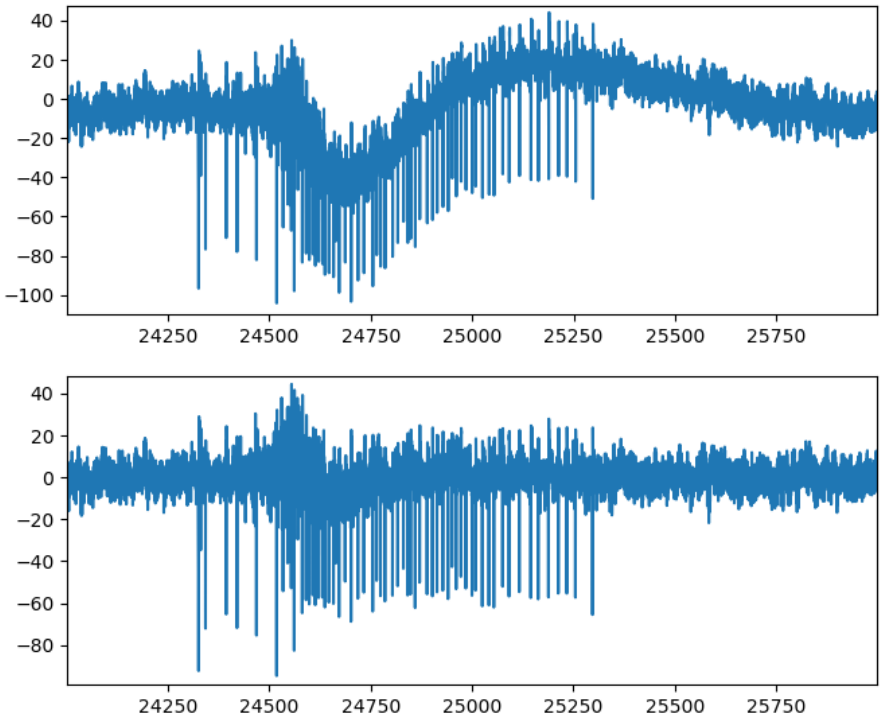
va générer un fichier qui commence comme suis :

```
%time av_value
0.000000 -4.979630
0.100000 -4.977984
0.200000 -4.982164
0.300000 -4.987988
...
```

La moyenne mobile au point **pos** c'est le calcul de la valeur moyenne du signal sur l'intervalle [**pos-win_size**, **pos+win_size**]. La fonction m_average doit donc réaliser ce calcul pour tous les points de channel_2.txt et retourner un fichier ayant le même nombre de lignes mais leur moyenne mobile. La figure ci-dessus montre la moyenne mobile tracée en rouge et qui représente ici la ligne de base du signal.

4 Commande *set_baseline*

Maintenant que vous savez calculer la ligne de base du signal (sa moyenne mobile) vous devez écrire une commande qui **soustrait** la ligne de base au signal original de façon à ce que celui-ci soit le plus stable possible. Son effet est visualisé dans la figure ci-dessous. En haut vous avez le signal original et en bas le signal dont la fonction set_baseline a soustrait la ligne de base calculée avec la moyenne mobile.



Cette commande set_baseline va donc prendre comme arguments un fichier d'entrée (ici le fichier initial data_mcs.txt), un fichier de sauvegarde (ici corr_channel_2.txt), le numéro du canal qui sera traité et un paramètre win_size jouant le même rôle que celui qui est spécifié dans la fonction m_average :

```
set_baseline data_mcs.txt corr_channel_2.txt 2 1000
```

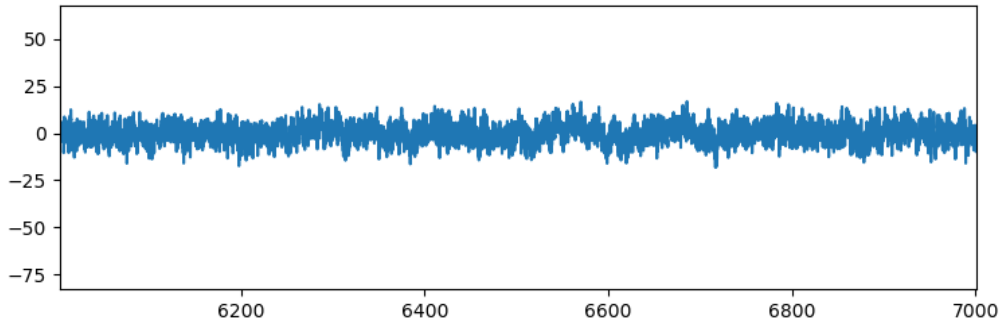
Le fichier corr_channel_2.txt (**corr_** pour corrected) devrait ressembler à ça :

```
%time c_channel_2
0.000000 0.399630
0.100000 2.897984
0.200000 4.562164
0.300000 1.237988
...
```

Le nom de colonne doit commencer par **c_** pour indiquer que c'est un canal corrigé.

5 Commande *spiketimes*

Cette commande va détecter les instants des potentiels d'action en utilisant une méthode de seuillage semi-automatisée. Dans un premier temps l'expérimentateur va localiser sur un enregistrement une région qui lui semble totalement dépourvue de spikes comme sur la figure ci-dessous (6000 à 7000 ms pour le canal 2) :



Il va ensuite appeler la fonction spiketimes comme suis :

```
spiketimes corr_channel_2.txt spikes_channel_2.txt 6000 7000 3
```

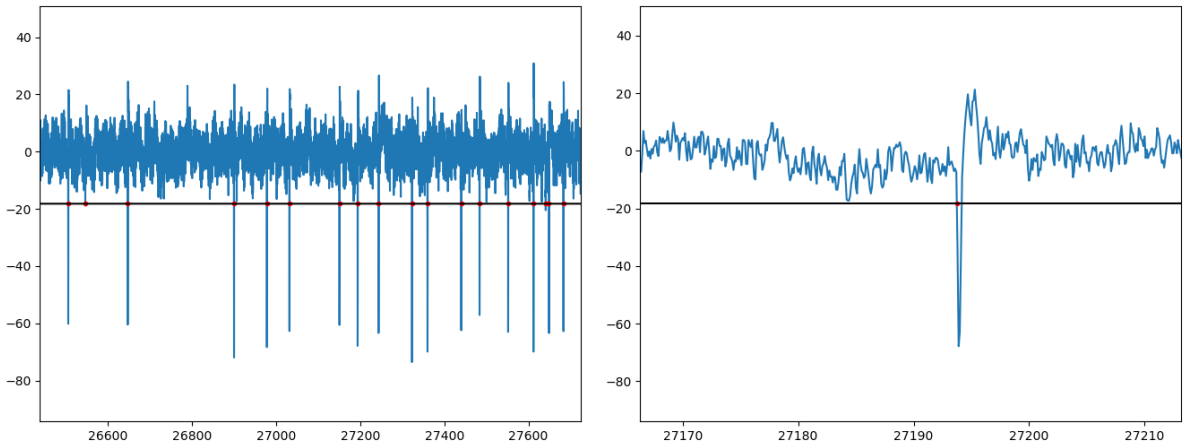
les arguments sont respectivement le fichier contenant le canal corrigé (obtenu après set_baseline), le nom du fichier de sortie, les valeurs de début et de fin de l'enregistrement supposément dépourvues de spikes et enfin la durée minimale (ref_lapse) entre 2 spikes (ici fixée à 3 ms).

La fonction va donc parcourir le fichier corr_channel_2.txt dans l'intervalle 6000 à 7000 ms et rechercher les valeurs min et max qui s'y trouvent. Seule la valeur min sera réellement utilisée ici. En effet elle va ensuite être utilisée comme un seuil pour parcourir le fichier en entier. Chaque fois que cette valeur sera dépassée (en négatif) un potentiel d'action sera détecté et son instant spécifié. La valeur ref_lapse est nécessaire pour spécifier la durée minimale entre 2 potentiels d'action afin d'éviter que, si le signal reste pendant plusieurs instants sous la durée du seuil, cela compte plusieurs spikes alors qu'en réalité un seul s'y trouve.

Le fichier de sortie doit comporter les colonnes suivantes : time, value, une colonne spike et une colonne threshold. La colonne value reprend la valeur du potentiel à l'instant time. La colonne spike contient la valeur **nan** tant que aucun potentiel d'action n'est trouvé et à l'instant ou il est détecté la valeur du seuil y est placée. La colonne threshold contient la valeur min calculée précédemment donc le seuil. Cette structure vise uniquement à rendre sa visualisation plus simple par exemple avec quelques lignes de Python.

```
%time value spike threshold
0.000000 0.399630 nan -18.192819
...
27193.500000 -7.039440 nan -18.192819
27193.600000 -6.203393 nan -18.192819
27193.700000 -7.455267 nan -18.192819
27193.800000 -33.706937 -18.192819 -18.192819 # ligne avec spike
27193.900000 -67.877146 nan -18.192819
27194.000000 -62.876107 nan -18.192819
...
```

La figure ci-dessous illustre la visualisation des potentiels d'action détectés (points rouges) le signal (bleu) et la valeur seuil (noir).



Comme on le voit la figure de droite montre le potentiel d'action détecté dans l'encadré ci-dessus à t=27193.8 ms.

6 Commande *overlay*

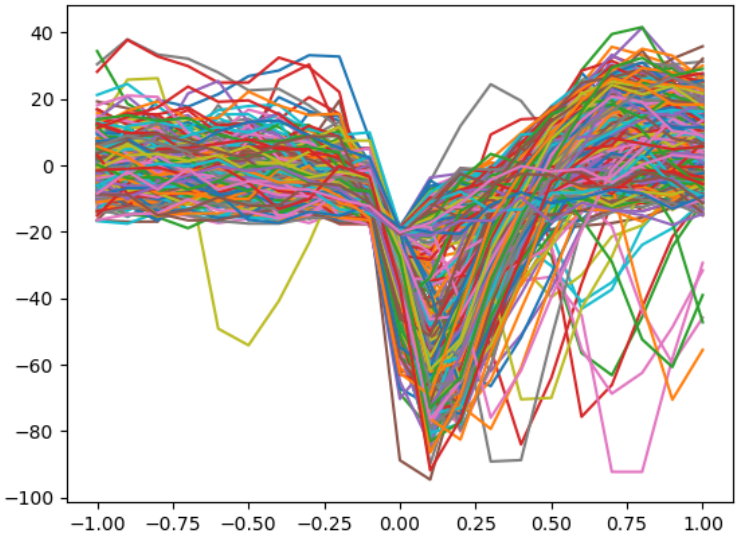
Cette commande récupère un fichier de sauvegarde des spikes, une durée avant le spike **before** et une durée après le spike **after** en ms. Pour chaque spike il récupère les valeurs du signal situées de **before** à after ms autour du spike (ici de -1 ms à +1 ms). Après exécution il affiche le nombre de potentiels d'actions ayant été trouvés ainsi que la durée totale de l'enregistrement :

```
overlay spikes_channel_2.txt overlay_channel_2.txt 1 1 # before et after
427 spikes found in 60000.1 ms
```

Il sauvegarde ensuite une matrice de ces séries temporelles centrées sur chaque potentiel d'action détectées et organisées dans un fichier de la façon suivante. La première colonne va de before à after en ms puis chaque colonne correspond au signal enregistré dans cet intervalle autour du spike. Ce fichier ne comporte pas d'entête et aura donc $n + 1$ colonnes si n spikes avaient été détectés.

```
-1.000000 1.029938 -5.233258 0.057461 -1.526107 ...
-0.900000 4.369620 -3.986382 5.057671 0.972019 ...
-0.800000 -3.548557 -1.490550 -2.020870 0.967021 ...
-0.700000 -1.469518 0.179865 2.980795 1.373068 ...
-0.600000 -3.550046 -1.900345 2.560795 0.122864 ...
-0.500000 -8.971005 -1.489510 2.983293 8.462449 ...
-0.400000 -9.800891 -4.819300 2.569125 13.872239 ...
-0.300000 -7.719498 -10.241384 -7.838791 20.544533 ...
-0.200000 -5.627682 -8.571799 -2.840040 15.551404 ...
-0.100000 -10.624373 -11.900130 -9.933163 -14.859640 ...
0.000000 -19.372347 -18.988256 -68.684203 -56.103808 ...
0.100000 -6.870537 -16.899090 -79.516912 -44.851309 ...
0.200000 -6.460429 -1.488051 -52.847741 -24.007561 ...
0.300000 -3.958407 -1.484928 -29.517116 -11.920480 ...
0.400000 -3.956179 -2.315342 -18.686282 -11.916107 ...
0.500000 -1.866288 -1.071174 -15.347321 -6.913818 ...
0.600000 -2.285759 0.172164 -7.848156 3.916392 ...
0.700000 -0.206928 5.166332 -4.522949 16.836602 ...
0.800000 3.962327 3.913833 -5.354198 27.259305 ...
0.900000 1.039888 0.159670 -2.021489 29.337641 ...
1.000000 -4.791277 -4.014703 3.400595 21.005142 ...
```

Le tracé de ce fichier donne le résultat suivant :



7 Commande *ISIH*

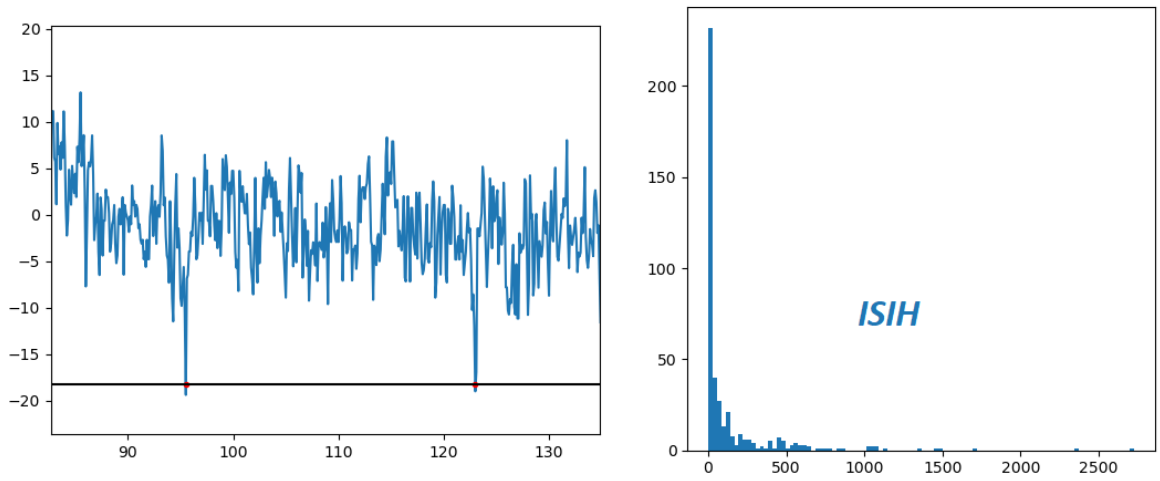
Cette commande va prendre un fichier de détection des spikes obtenu grâce à la fonction spiketimes et sauvegarder la liste des intervalles interspikes. Pour ce faire elle va parcourir tous les instants des potentiels d'action et calculer le temps écoulé entre chacun puis sauvegarder (sans entête) une colonne qui contient tous ces intervalles entre les spikes. Le fichier de sortie n'aura pas d'entête. S'il y n spikes alors logiquement il y aura $n-1$ intervalles. Exemple :

```
isih.exe spikes_channel_2.txt isih_2.txt
```

Le fichier isih_2.txt doit ressembler à ça :

```
27.500000
479.600000
4.700000
10.500000
...
```

On peut le vérifier sur le graphe. Les deux premiers spikes sont effectivement bien distants de 27.5 ms et Le fichier ainsi généré doit permettre de facilement tracer l'ISIH (*Inter Spike Intervals Histogram*) respectivement les graphes de gauche et droite sur la figure ci-dessous :



8 Commande *spike_sort*

Cette commande donne un *spike sorting* rudimentaire. Le spike sorting consiste à trier les sources des potentiels d'actions (souvent plusieurs neurones sont en contact avec l'électrode et contribuent au signal) en fonction des caractéristiques des spikes. Pour cet exemple il suffira d'utiliser l'amplitude du spike. A chaque fois qu'un spike est détecté, son amplitude doit être calculée (sa valeur minimale atteinte). Le programme calcule ensuite pour tous les spikes relus dans le fichier source (ici overlay_channel_2.txt) l'amplitude **amp** entre le spike avec la valeur la plus basse (**loc_min**) et celui avec la valeur la plus haute (**loc_max**) : **amp**=**loc_max**-**loc_min**.

Il crée 4 intervalles de même amplitude :

- intervalle 1 de loc_min à $loc\_min + \frac{amp}{4}$
- intervalle 2 de $loc\_min + \frac{amp}{4}$ à $loc\_min + \frac{2 \times amp}{4}$
- intervalle 3 de $loc\_min + \frac{2 \times amp}{4}$ à $loc\_min + \frac{3 \times amp}{4}$
- intervalle 4 de $loc\_min + \frac{3 \times amp}{4}$ à loc_max

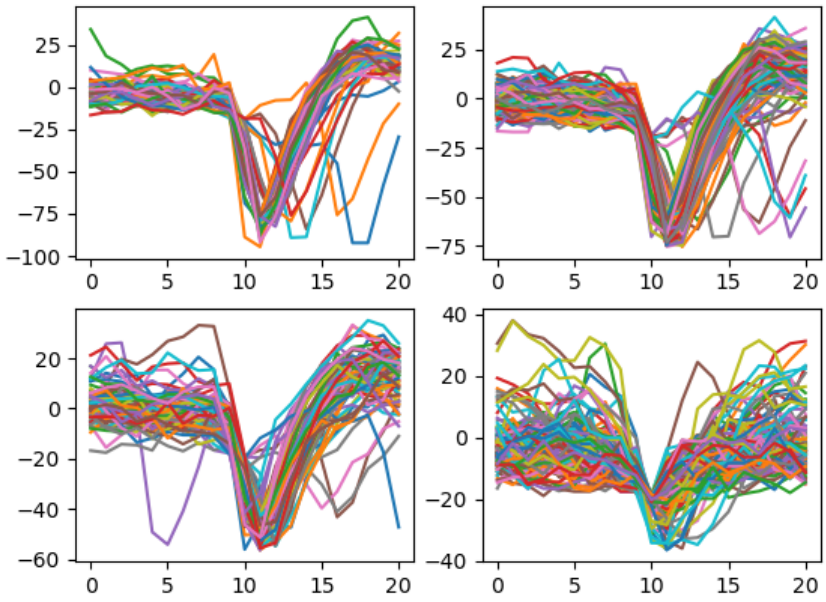
Par exemple si loc_min vaut -100 et loc_max -20 alors :

- amp=-20+100=80, intervalle 1 [-100,-80], intervalle 2 [-80,-60], intervalle 3 [-60,-40] et intervalle 4 [-40,-20]

Les spikes seront ensuite sauvegardés dans 4 fichiers séparés en fonction de appartenance à un intervalle. Chaque fichier sera nommé avec un numéro terminal **_X** situé dans son nom en fonction de l'intervalle. La commande devra être appelée comme suis :

```
spike_sort overlay_channel_2.txt channel_2_sort
```

Cette commande va créer alors automatiquement les fichiers channel_2_sort_1.txt, channel_2_sort_2.txt, channel_2_sort_3.txt et channel_2_sort_4.txt correspondant aux intervalles de même numéro. Leur visualisation devrait ressembler à ça :



De gauche à droite et de haut en bas sur la figure les intervalles 1, 2, 3 et 4.

Exemple d'exécution chainée (*pipeline*)

Idéalement l'encadré ci-dessous devrait pouvoir être exécuté globalement dans un fichier *bash* sans générer d'erreur et en créant tous les fichiers selon les instructions spécifiées :

```
extract data_mcs.txt channel_2.txt 2
m_average channel_2.txt ma_channel_2.txt 1000
set_baseline data_mcs.txt corr_channel_2.txt 2 1000
spiketimes corr_channel_2.txt spikes_channel_2.txt 6000 7000 3
overlay spikes_channel_2.txt overlay_channel_2.txt 1 1
427 spikes found in 60000.1 ms
isih.exe spikes_channel_2.txt isih_2.txt
spike_sort overlay_channel_2.txt channel_2_sort
```