Lucas Burget
Daniel Cavaleri
Andréa Asli
Chléa Schiff

Team 21
Maxime Emschwiller

Projet 2: Réseau de neurones

1. A quoi sert ce programme ?

a. Description du projet

Ce programme a pour but de modéliser un réseau de neurones. Il permet de créer un nombre donné de neurones et de les connecter les uns aux autres afin de simuler leurs interactions.

Il y a cinq types de neurones, répartis en deux catégories :

- Les neurones inhibiteurs : FS et LTS
- Les neurones excitateurs : RS, CH et IB

Selon son type, un neurone possède des paramètres cellulaires différents qui impacteront son évolution. Ces paramètres sont fixés lors de la création du neurone et ne changent pas lors de la simulation.

Les liens entre les neurones sont des liens directionnels, c'est-à-dire que les signaux ne pourront parcourir le lien que dans un seul sens. Par exemple, si le neurone A est connecté au neurone B, tel que ce soit le neurone A qui envoie un signal au neurone B, alors le neurone B ne pourra jamais envoyer de signal au neurone A. Dans cette paire de neurones, A sera le neurone “envoyeur” et B le neurone “receveur”.
La modélisation du lien entre deux neurones comprend donc la paire neuronale impliquée, le sens de ce lien, ainsi que son intensité. L’intensité du lien déterminera l’intensité du courant synaptique envoyé par le neurone A au neurone B.

Chaque neurone possède deux propriétés qui évoluent au cours du temps selon le courant synaptique envoyés par ses neurones voisins :

- Le potentiel de membrane
- Le temps de récupération

La valeur du potentiel de membrane définit l’état dans lequel se trouve le neurone : si cette valeur dépasse le seuil de décharge (30 mV), le neurone sera dans l’état **“firing”**, c'est-à-dire qu’il transmettra à ses neurones “receveurs” un courant synaptique.
Pour envoyer un signal, il faut donc nécessairement que le neurone “envoyeur” soit dans l’état **“firing”**.

Le potentiel et la récupération de chaque neurone sont mis à jour à chaque pas de simulation, selon la somme des courants synaptiques reçus par ses neurones “envoyeurs” qui sont en état **“firing”.**

Le but ultime de ce programme est de détecter, pour chaque pas de simulation, les neurones qui sont en état “firing” et ceux qui ne le sont pas. Afin de réaliser ce but, des fichiers de sorties sont générés par le programme.

b. Fichiers de sortie

L'exécution du programme génère trois fichiers de sortie sous format texte :

- Un fichier représentant les neurones en état “firing”, appelé ici *“fichier spikes”*.

- Un fichier qui liste les valeurs du potentiel, du temps de récupération et du courant synaptique d'un échantillon de neurones à chaque pas de simulation. L'échantillon comprend un neurone de chaque type de neurone présent dans le réseau. Ce fichier sera appelé ici *"fichier sample"*
- Un fichier qui contient les paramètres de chaque neurone du réseau, appelé *"fichier parameters"*

Le fichier de sortie principal, le *"fichier spikes"*, est une matrice de 0 et de 1. Les lignes représentent les pas de simulation, tandis que les colonnes représentent tous les neurones. Un "1" est imprimé dans la colonne et la ligne correspondante lorsqu'un neurone est en état "firing" à un certain pas de simulation. Autrement, un "0" est imprimé.

Ces trois fichiers générés nous serviront à créer un "raster plot" ainsi que des graphiques secondaires à l'aide d'un algorithme annexe. Un "raster plot" est un graphique représentant les occurrences temporelles d'une certaine donnée. Pour ce programme, le raster plot représente par des points noirs les neurones qui sont en état "firing". L'algorithme transforme ainsi les 1 de notre fichier en points noirs, ce qui nous permet de facilement repérer les neurones en état "firing".

2. Comment utiliser ce programme ?

a. Paramètres utilisateurs

Un certain nombre de paramètres sont choisis par l'utilisateur lors du lancement du programme. Nous allons les détailler ici. La lettre entre parenthèse sera utilisée pour spécifier le paramètre lors du lancement du programme (cf. partie 2.b.).

- Le nombre total de neurones présents dans le réseau (-n)
- Les proportions de chaque type de neurones (-T)
- Le temps total de simulation en nombre de pas (-t)
- Le nombre moyen de connections de chaque neurones (-c)
- Le modèle de dispersion pour choisir aléatoirement le nombre de connections de chaque neurone : "constant", "poisson" ou "over-dispersed" (-M)
- L'intensité moyenne des connections (-I)
- Le nom des trois fichiers de sorties (-o, -s, -p)
- La longueur de l'intervalle dans lequel les paramètres des neurones sont choisis aléatoirement, autour des valeurs spécifiques à leur type. (-d)

Aucun de ces paramètres n'est obligatoirement requis pour lancer le programme. Si la valeur d'un paramètre n'est pas spécifiée, la valeur par défaut prendra effet.

b. Lancement du programme

Le programme est lancé à l'aide de la commande ./NeuronNetwork.
Si aucun paramètre n'est spécifié à la suite de cette commande, ce sont les paramètres par défaut qui seront utilisés dans le programme.

Au contraire, si ./NeuronNetwork est suivi de paramètres utilisateurs, ils seront pris en compte dans la modélisation du réseau de neurone. Pour spécifier un paramètre, il faut utiliser la lettre indiquée dans la partie précédente précédée d'un tiret "-" et suivie d'un espace et de la valeur du paramètre.

Voici un exemple d'utilisation du programme en spécifiant chaque paramètre:

```
./NeuronNetwork -n 5000 -t 1000 -T 'FS:0.6,CH:0.2' -M constant -c 5
-l 20 -o spikes_file.txt -s sample_file.txt -p params_file.txt
```

c. Création des graphiques

Pour créer le raster plot et les graphiques à partir des fichiers de sortie générés par le programme, il faut télécharger le programme RasterPlots.R est effectuer cette commande:

```
Rscript RasterPlots.R spikes_file.txt sample_file.txt params_file.txt
```

avec les noms des trois fichiers de sortie.

Trois fichiers PDF sont alors automatiquement générés, sur lesquels apparaissent différents graphiques.

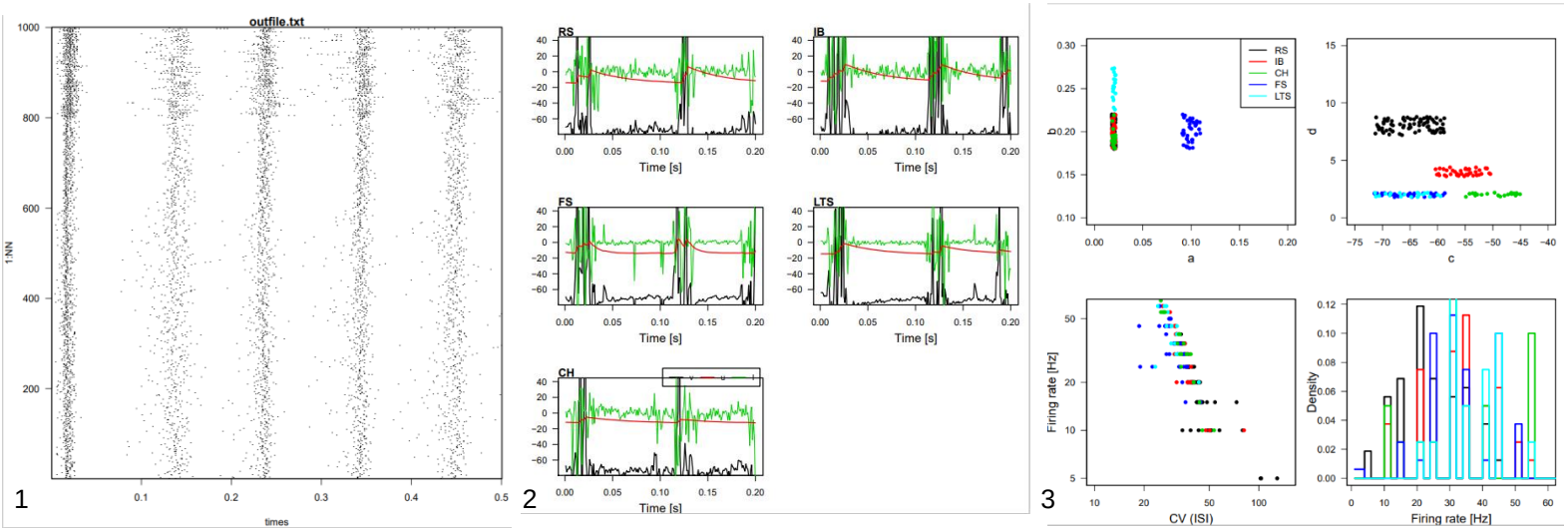

Figure 1 : capture d'écran des trois fichiers d'output
1 : le *"fichier spikes".* Les points noirs correspondent au neurones "firing".
2 : le *"fichier sample".* Les valeurs du potentiel (v, noir), du temps de récupération (u, rouge) et du courant synaptique (i, vert) sont représentées sous forme de graphique en fonction du temps.
3 : le *"fichier parameters".* Les deux graphiques du haut représentent la distribution des valeurs des paramètres de chaque neurone, tandis que les deux graphiques du bas représentent des valeurs statistiques.

3. Comment ce programme est-il conçu et testé ?

a. Conception des classes et des méthodes

Tout se passe au niveau du fichier **Main** où la **Simulation** est créée puis lancée. Cette classe est la classe principale du programme : Elle construit le Réseau de Neurone, lance le programme et met à jour les différents éléments du réseau.

Une **Simulation** est faite d'un **Réseau** qui est composé d'un nombre de **Neurone** donné. C'est dans cette classe que les paramètres insérés par l'utilisateur (cf partie 2.a.) sont testés et vérifiés. Dans le cas où un de ces paramètres n'est pas valide, le programme s'arrête et affiche un message d'erreur à l'utilisateur, pour qu'il insère des paramètres corrects.

Cette classe est caractérisée par *le nombre de pas total à effectuer, le nombre de neurones, la connectivité moyenne, l'intensité moyenne des connexions, le model choisi pour les nombres aléatoires et les proportions des différents types de neurones.*

La Simulation ainsi créée appelle la méthode qui lance la simulation, du temps 0 jusqu'à atteindre le nombre total de pas à effectuer : ***run().*** Cette méthode met à jour le réseau de Neurones (avec la méthode ***update()***) et appelle la méthode ***print()*** à chaque pas de simulation. Cette dernière a pour but de créer les différents fichiers de sorties mentionnées plus haut.

Un **Réseau** de Neurones est l'ensemble des **Neurones** et leurs connexions. Un **réseau** est caractérisé par *le vecteur de **Neurones*** qu'il contient*, les différentes proportions de chaque type de neurone, liste des liens unidirectionnels entre les neurones*. Au moment de la création du réseau, l'ensemble des neurones est créé et ils sont liés entre eux aléatoirement. Les liens entre ces différents neurones sont unidirectionnels. Chaque **Neurone** envoie un signal synaptique à un seul autre **Neurone** mais reçoit plusieurs signaux synaptiques de plusieurs **Neurones**.

Un **Neurone** peut être soit Inhibiteur soit Excitateur. Il possède quatre paramètres (a,b,c,d) et a trois attributs représentant son état (Potentiel, Récupération et le Courant). Ils sont influencés par un "noise" qui est choisi aléatoirement. Il existe différents types de neurones : le type détermine la valeur des paramètres.

Au cours de la Simulation, à chaque pas de temps, le potentiel et la récupération de chaque neurone sont mis à jour, avec la méthode : ***update()*** qui s'occupe aussi de déterminer si le Neurone est en état "firing". Le courant Synaptique sortant d'un neurone dépend donc du courant de tous ses voisins. Les neurones inhibiteurs envoient un courant synaptique négatif et les neurones excitateurs envoient un courant synaptique positif.

Pour pouvoir utiliser des nombres aléatoires dans la plupart de nos méthodes, une classe **RandomNumbers** est indispensable. Le générateur utilisé est un Mersenne Twister engin *mt19937.* Les nombres renvoyés dépendent de la loi de distribution choisie (elle peut être uniforme, normale, exponentielle ou de poisson).

b. Tests

Pour s'assurer que tout le programme fonctionne correctement, la conception de tests pour nos méthodes "complexes" est indispensable et nous permet de savoir si les résultats sont en effet les résultats attendus.

Nous avons créé des tests pour chacun des problèmes suivants : *la création et l'initialisation d'un **Réseau**, la création de liens entre neurones, la présence des voisins d'un **Neurone**, la mise à jour de l'état "firing" des neurones, la mise à jour du potentiel, le calcul de la valence d'un Réseau, et la mise à jour des différents éléments du réseau.*
En plus de ces tests, nous avons aussi mis en place le traitement des exceptions qui nous permet de gérer les erreurs d'exécution en arrêtant le déroulement du programme et en affichant le message d'erreur.