Table des matières

Première partie

Introduction

Ce rapport a pour objectif d'exposer les différents outils mis à disposition afin de tester différents routages dans le cadre d'une étude des champs neuronaux dynamiques.

Deuxième partie

Stimulus

0.1 Objectif

On peut exciter chacun des nœuds du graphe à différents moments. L'objectif est de voir le comportement du graphe (et du routage associé) selon différents événements. Ces événements sont caractérisés par l'envoie de stimulus, i.e l'ajout de paquet à la file d'attente d'un ou plusieurs nœuds à un ou plusieurs moments de la simulation.

0.2 Utilisation

Le framework pour la programmation d'envoi de stimulis se base sur le format xml. Tout les fichiers de programmation doivent se trouver sous le dossier statistiques/stimulis avec l'extension .stimulis (et non .xml). Cette extension est nécessaire aux fichiers de scripts notamment.

Commençons par un exemple. Admettons que l'on veuille envoyer un paquet de type spike au noeud zéro du graphe à l'instant t=0 :

asend.stimulis

```
<?xml version="1.0" encoding="UTF-8"?>
<programation class="main.java.network.generic.packet.Spike">
        <time t="0.0">
                <add indice="0"/>
        </time>
</programation>
```

On retrouve ici trois balise :

- <programation> : On stipule qu'on va commencer un programme d'envoi stimulis (il y a une faute à "programation" mais j'ai codé ça comme ça). L'attribut "class" est le nom de la class Java des packets que l'on veut envoyer en tant que stimulis.
- <time> : Pour spécifier les envois à un temps t donné. A noté que

$$dtStimulis = k * dtGraphe \ \forall k \epsilon \mathbb{N}^*$$

  i.e on ne peut pas envoyer des stimulis entre deux computations.
- <add> : On envoie un paquet à l'indice indice. Notez que les indices sont en representation 1D du graphe.

Un exemple d'envoi de paquets ipv4 sur un reseau :

ipv4.stimulis

```
<?xml version="1.0" encoding="UTF-8"?>
<programation class="main.java.network.generic.packet.IPv4Datagramme">
        <time t="0.0">
                <add indice="0">
                        <params> A </params>
```

```
                        <params> yes </params>
                        <params> ca fonctionne ! </params>
                </add>
        </time>
        <time t="0.1">
                <add indice="80">
                        <params> B </params>
                        <params> yes </params>
                        <params> ca fonctionne ! </params>
                </add>
        </time>
</programation>
```

On découvre la balise <params>. L'utilisation de cette balise est directement liée à la structure du constructeur de la class spécifiée dans la balise programation. Dans l'exemple précédent on utilisait des Spikes, objet simple ne prenant aucun paramètre lors de la construction, on a pas eu besoin d'utiliser la balise <params>. Par contre un paquet ipv4 transporte un message. Ce message est "A". Les valeurs "yes" et "ça fonctionne !" n'ont aucun effet sur la construction d'un objet ipv4 vu qu'il ne prend qu'un string en params.

Il faut regarder de plus près le code java.

Packet.java

```java
public class Packet {

    private int size;

    protected Packet(Object ... params){
    }

    //...
```

Chaque classe fille de Packet doit implémenter le constructeur protected.(ça doit être le premier constructeur dans le code pour des raisons d'introspection, même si il y a moyen de faire mieux).

Spike.java

```java
public final class Spike extends Packet {

    public Spike(Object ... params) {
        setSize(1);
    }

}
```

On voit que Spike n'utilise aucun de ses paramètres.

IPv4Datagramme.java

```
1  public final class IPv4Datagramme extends Packet{
2
3          private String message;
4
5          public static final int MESSAGE_INDEX = 0;
6
7          public IPv4Datagramme(Object ... params) {
8          setSize(16);
9          setMessage((String)params[MESSAGE_INDEX]);
10         }
11
12         //...
```

IPv4 utilise un seul paramètre. A noté que les objets récupérés seront de type String.

Troisième partie

Generation de données

0.3 Introduction

Afin de tester certaines propriétés du graphe, il est nécessaire de générer un jeu de donnée conséquent concernant différents caractéristiques du routage.

0.4 Spikes reçus jusqu'alors

Dans cette section, nous allons nous intéressé à la génération du jeu de donnée représentant la quantité de spike reçu par chaque nœud du graphe, de l'instant 0 à l'instant t. Pour ce faire, il faut utiliser le script generateData.py. Ce script s'occupe de lancer cette expérience :

J'ai un graphe de routage type probabilistic flooding model, j'envoi des stimulis à un ou plusieurs moments de la propagation. Ce graphe a une taille et un poid (probabilité de détruire un spike sortant d'un noeud). Je laissse tourner la propagation pendant t secondes.Je récupère la matrice répresentant le nombre total de spikes réçus par chaque noeuds jusqu'a l'instant t

On lance cette expérience plusieurs fois (itérations) et selon différents paramètres (taille, poids, programmation des stimulis).

Un exemple d'appel de cette commande :

```
python generateData.py --weigths 0.0 0.1 0.2 0.3 0.4 --times 1 2
--packet_initialisation twoa twob twoab --iterations 100
--tailles_grilles 9 --forcerewrite
```

Ici on génére les données pour les combinaisons de poids temps. On stipule que ce sera de taille 9 mais on aurait pu écrire

```
--taille_grilles 1 2 3 4 5 6 7
```

On veut que l'expérience soit réalisée selon plusieurs programmations (twoa twob et twoab). Les dites programmation doivent se trouver dans le dossier statistiques/stimulis sous la forme nom.stimulis (exemple : twoab.stimulis).

Les données généré se trouve sous le dossier statistiques/data (avec une arborescence dépendante des paramètres).

A noter que ce script n'est utilisable qu'avec le PFModel pour le moment mais il est facilement extensible à d'autre modèles.

Faire

```
python generateData.py --help
```

pour plus d'informations

Quatrième partie

Additivité

Il est nécessaire de conserver la propriété d'additivité du réseau lors de l'utilisation de nouveaux routages. Pour cela, posons une définition théorique de l'additivité.

0.5 Théorie(à refaire)

A partir de maintenant, on considère un graphe G(N,E) avec N l'ensemble de ses noeuds et E l'ensemble de ses arêtes. Chaque noeud possède une file d'attente.On peut lancer une propagation sur ce graphe.

Definition 1. *(Paquet)*
Un paquet $\mathtt{p}$ *est un objet se deplançant sur le graphe de noeuds en noeuds au fil du temps (lors de la propagation du graphe).*

Definition 2. *(Stimulis)*
Un stimulis $\mathtt{S}_{p,n,t}$ *est le placement du paquet* $\mathtt{p}$ *dans la file d'attente du noeud n à l'instant t.*

Definition 3. *(Configuration)*
Une configuration $\mathtt{C}$ *est un ensemble de stimulis.*

Definition 4. *(Additivité)*
Soit $\mathtt{S}_0, \mathtt{S}_1..\mathtt{S}_K$ *des stimulis.*
On pose la configuration $\mathtt{C} = \{\mathtt{S}_k \mid n\epsilon[0,K] \text{ et } K > 0\}$
On pose $\mathtt{C}_k = \{\mathtt{S}_k\}\ \forall n\epsilon[0,K]$

Soit $M_{t,n,\mathtt{C}}$ *la variable aléatoire qui représente l'ensemble des spikes reçu par le noeud n depuis le début de la propagation pour chaque nœud du graphe à l'instant t selon la configuration* $\mathtt{C}$.
On pose $M_{t,\mathtt{C}} = (M_{t,0,\mathtt{C}}, ..., M_{t,1,\mathtt{C}}, M_{t,card(N)\mathtt{C}})$ *et* $M_{t,\mathtt{C}_i} = (M_{t,0,\mathtt{C}_i}, ..., M_{t,1,\mathtt{C}_i}, M_{t,card(N)\mathtt{C}_i})$

On dit que le routage est **additif pour la configuration** $\mathtt{C}$ *si* $\forall t$

$$\mathbb{E}(M_{t,\mathtt{C}}) = \sum_{i=1}^{n} (\mathbb{E}(M_{t,\mathtt{C}_i}))$$

On dit qu'un routage est **additif** *si il l'est pour n'importe qu'elle configuration.*

cas particulier (t=0)

0.6 Utilisation du script

Afin de tester cette propriété d'additivité sur des resultats expérimentaux, nous avons crée un script en Scilab. Ce dernier lit un ensemble de donnée de

plusieurs répétition d'exprérience afin d'extraire une moyenne de ces données, puis il va calculer à quelle point le routage est additif sur ces résultats moyens.

scilab -nw -f Additivite.sce -args '9' '1 2' '0.0 0.1 0.2 0.3 0.4' '5 20 50 75 100' 'twoa twob' twoab 0

0.7 Résultats

Cinquième partie

Propagation des spikes

0.8 Utilisation du script

0.9 Résultats

0.10 Réflexions(intervalle pas possible car pas distrib...)

Sixième partie

Probabilistic flooding et I/E

0.11 Utilisation du script

0.12 Résultats

Septième partie

Annexe

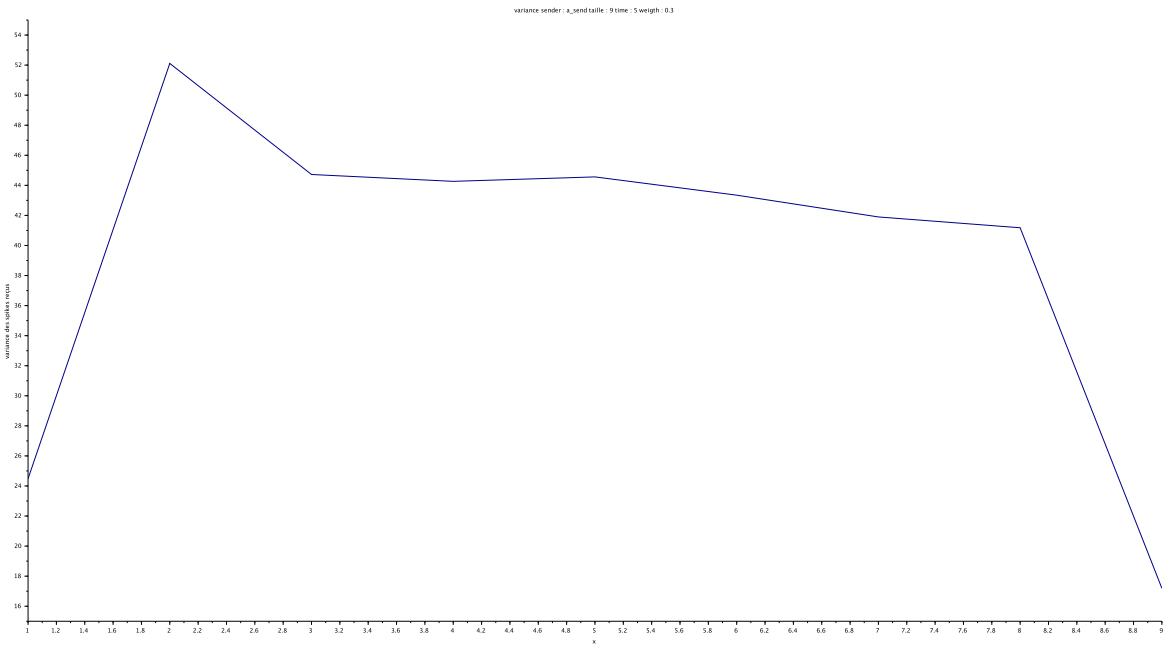

FIGURE 1 – Un jolie oiseau !

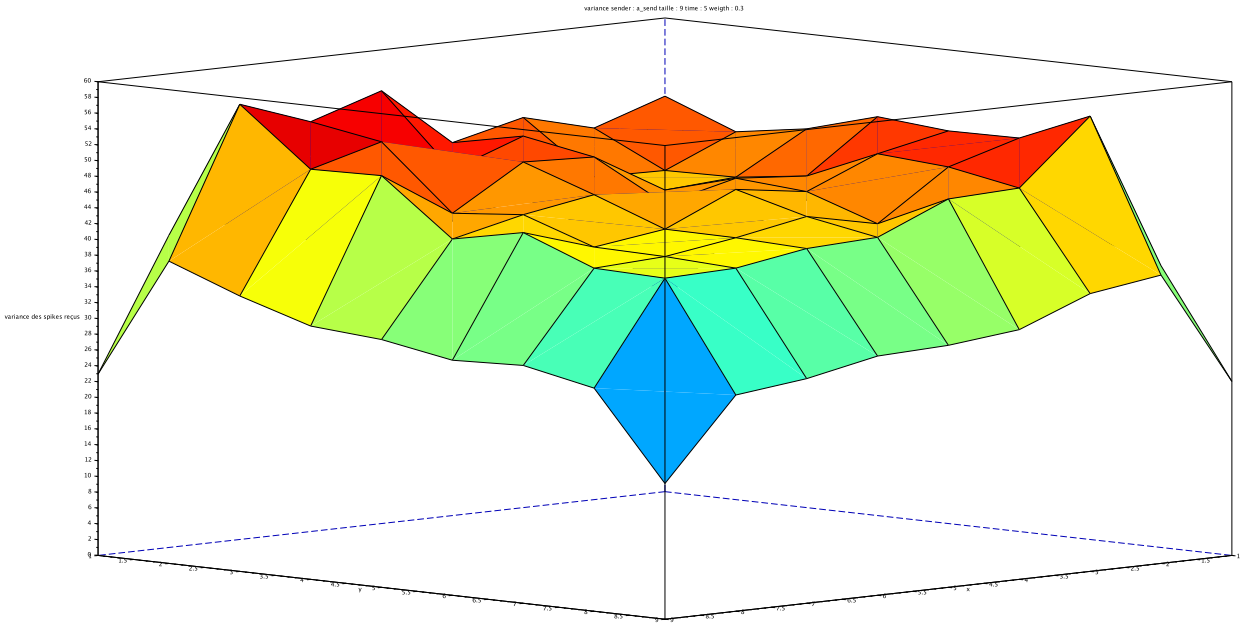

FIGURE 2 – Un jolie oiseau !

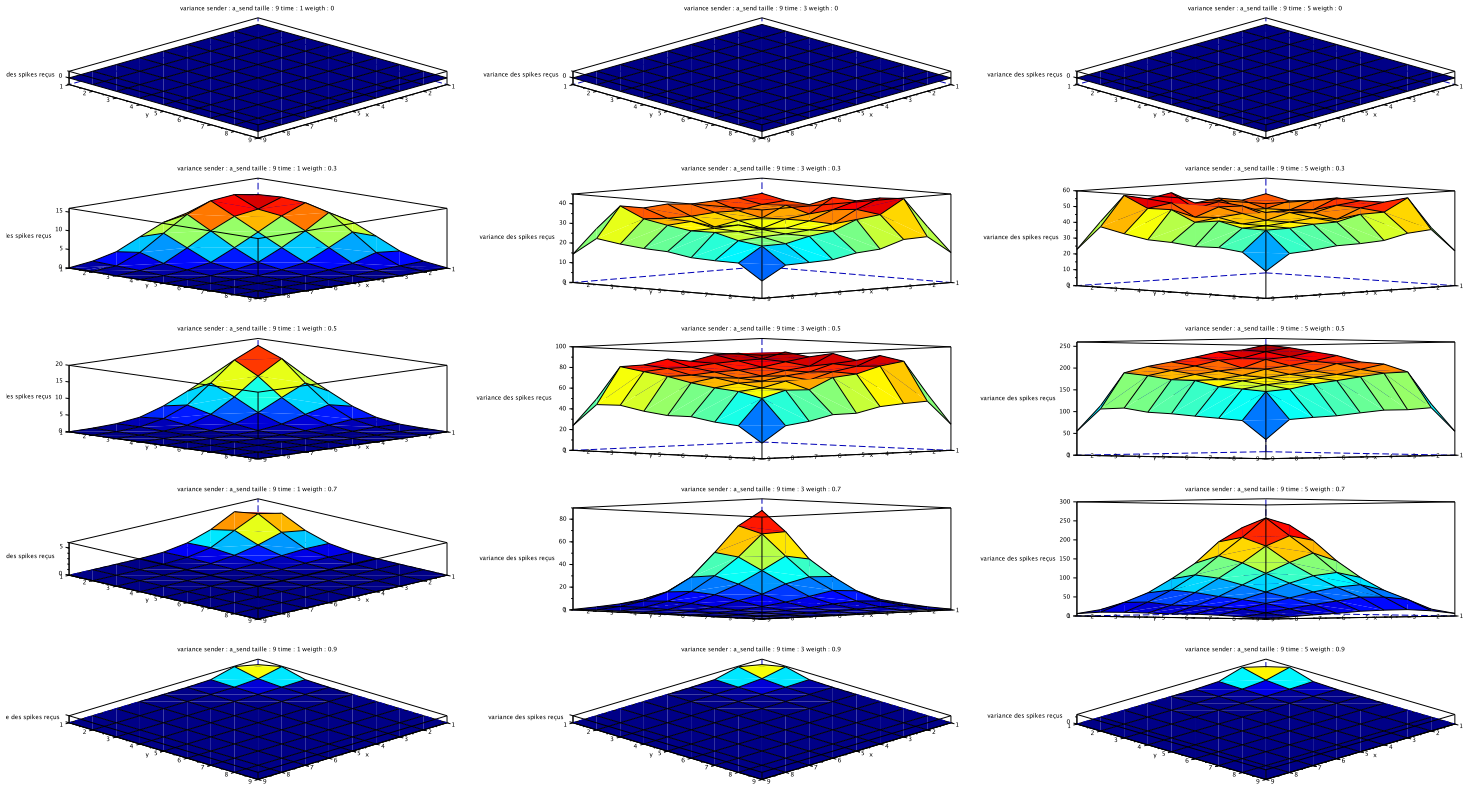

FIGURE 3 – Un jolie oiseau !