université de BORDEAUX

Projet de Recherche et Développement :

Élaboration d'un modèle computationnel modélisant le cycle éveil/sommeil chez le rongeur

Analyse et architecture

Client :

Héricé Charlotte, PhD
Attaché Temporaire d'Enseignement et de Recherche (ATER)
Institut Chimie et Biologie des Membranes et des Nano-objets (CBMN)
Université de Bordeaux

Auteurs :

Bielle Paul
Denet Lola
Guinot Charles
Hui Tongyuxuan
Niu Wenli
Étudiants en Master 1 de Bio-informatique
Université de Bordeaux

6 Mars 2020

Table des matières

Chapitre 1

Analyse du code C++

1.1 Architecture et fonctionnement

Les documents de référence [Schellenberger Costa *et al.*, 2016] incluent neuf fichiers en langage C++ et un fichier en MATLAB. Le résultat de cette combinaison de fichiers est l'obtention d'un graphe tel que sur la figure 1.1.

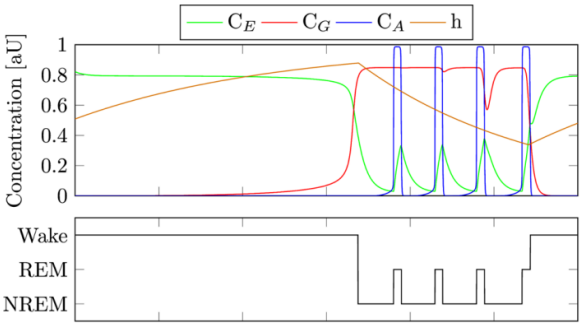

FIGURE 1.1 – Visualisation graphique des résultats attendus

L'architecture de ce modèle est représentée figure 1.2 et comprend trois populations neuronales induisant chacune un état du cycle éveil/sommeil, ainsi qu'une partie de régulation corticale.

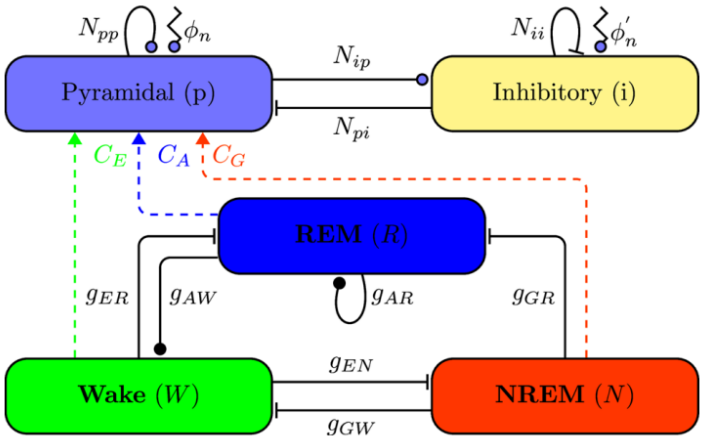

FIGURE 1.2 – Architecture du modèle avec les 3 populations et la zone corticale

Il existe deux fichiers principaux : le fichier `<Cortex_SR.cpp>` qui contient la fonction d'entrée `main()` et le fichier `<Cortex_SR_mex.cpp>` qui est le fichier source de MEX. (Le fichier MEX est un dérivé du langage C qui peut être appelé dans un environnement matlab. Les fichiers MEX sont des fichiers binaires produits à partir de sources en C et traités par le compilateur matlab. Il s'agit d'un programme dynamique qui peut être chargé et exécuté automatiquement par l'interrogateur matlab.) Les interactions d'inclusion et d'utilisation entre les différents fichiers et leur positionnement algorithmique seront présentés dans le

diagramme 1.4 de la partie suivante. Ici, nous allons préciser les paramètres et les fonctions de chaque fichier. Cela est résumé dans le tableau 1.3.

Nom du fichier	Type du fichier	Nombre des lignes	Nombre des fonctions	Nombre des paramètres	Structures de données utilisées	Lié à une plusieurs classe ou pas	Utilité
Random_Stream.h	fichier d'en-tête	56	6	3	int	oui	Génération du bruit
Cortical_Column.h	fichier d'en-tête	197	22	51	vector	oui	Déclaration des paramètres
Cortical_Column.cpp	fichier de C++	178	16	13	vector	oui	Initialisation populations
Sleep_Regulation.h	fichier d'en-tête	132	9	37	vector	oui	Déclaration des paramètres
Sleep_Regulation.cpp	fichier de C++	78	6	7	vector	oui	Initialisation populations
ODE.h (Ordinary Equation Solver)	fichier d'en-tête	42	1	0	vector	oui	Résoudre les équations différentielles
Cortex_SP.cpp	fichier de C++	74	1	4	vector	oui	Point d'entrée
Data_Storage.h	fichier d'en-tête	49	1	11	vector	oui	Stocker les données
Cortex_SR_mex.cpp	fichier de C++	129	2	11	mexArray	oui	Fichier source pour MEX
Plots.m	fichier de MATLAB	98	×	11	mexArray	×	Dessiner le graph

FIGURE 1.3 – Tableau d'analyse des fichiers

Voici le détail de chaque fichier :

1 - `<Sleep_Regulation.h>`

Les fichiers d'en-tête, dont le nom se termine normalement en `.h`, permettent de partager des déclarations entre plusieurs fichiers définitions composant un même programme.

`#pragma once` est une directive pré-processeur qui assure que le fichier source actuel ne peut être inclus qu'une fois dans la compilation.

`#include <vector>` et `#include <cmath>` se chargent respectivement d'introduire un module de vecteur et un groupe de fonctions mathématiques.

Ce fichier déclare une classe `Sleep_Regulation` qui comporte 9 fonctions et 37 variables et une classe `Cortical_Column` que nous expliquerons plus bas. Parmi ces déclarations, il y a deux fonctions basiques qui sont exécutées dans toutes les parties. La première fonction `inline std::vector<double> init (double value)` permet de retourner un vecteur pour exprimer toutes les variables qui facilitent le calcul de la valeur différentielle. Et la deuxième fonction `inline void add_RK (std::vector <double>& var)` montre une équation différentielle par la méthode d'intégration Runge-Kutta d'ordre 4 ($RK4$) [Baty, 2018]. Les autres paramètres et fonctions seront utilisés dans le fichier suivant pour établir les formules qui décrivent les relations entre les 3 populations.

2 - `<Sleep_Regulation.cpp>`

Un fichier d'en-tête est souvent inclus dans un fichier définition avec un même nom mais avec un format différent `.cpp` par une directive pré-processeur comme telle que : `#include "Sleep_Regulation.h"`.

Ce fichier propose six fonctions importantes [Diniz Behn et Booth, 2012] pour obtenir les fréquences de décharge neuronale selon les états d'éveil (Wake,F_W), NREM (F_N) et REM (F_R) ainsi que les concentrations des différents neurotransmetteurs : noradrénaline (C_E), acétylcholine (C_A) et GABA (acide gamma-aminobutyrique, C_G) et homéostasie (H). Ces dernières varient par unité de temps.

Chaque population neuronale possède une fonction d'entrée, qui représentera la somme des poids des synapses post-synaptiqus (connexions "entrantes") et de la concentration en neurotransmetteur de ces synapses. Les poids synaptiques sont exprimés avec une variable de type g_{XY} où X est la population pré-synaptique et Y est la population post-synaptique.

Les trois dernières fonctions se chargent d'exprimer les 7 équations principales suivantes avec le pas de temps dt.

- **La modélisation des fréquences de décharge neuronales par unité de temps :**

$$\frac{dF_W}{dt} = \frac{W_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_W - \beta_W}{\alpha_W}))) - F_W}{\tau_W} \tag{1.1}$$

$$\frac{dF_N}{dt} = \frac{N_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_N - \kappa_N \cdot H(t)}{\alpha_N}))) - F_N}{\tau_N} \tag{1.2}$$

$$\frac{dF_R}{dt} = \frac{R_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_R - \beta_R}{\alpha_R}))) - F_R}{\tau_R} \tag{1.3}$$

Ici, $\frac{dF_X}{dt}$ représente les décharges par unité de temps, τ_x est le constante de temps et Wmax, Nmax, Rmax correspondent respectivement aux fréquences maximales des décharges des 3 populations (popWake, popNREM et popREM). Dans l'expression tanh(X), X représente la somme des stimulations.

- **Le processus d'homéostasie :**

$$\frac{dH}{dt} = \frac{H_{max} - H}{\tau_{hw}} \cdot \mathcal{H}(F_W - \theta_W) - \frac{H}{\tau_{hs}} \cdot \mathcal{H}(\theta_W - F_W) \tag{1.4}$$

$Hmax$ la force homéostatique maximale du système. $\mathcal{H}(x)$ représente la fonction de *Heaviside*, qui renvoie 0 si $x < 0$ et 1 si $x \geq 0$.

- **La modélisation de neuromodulateurs :**

Les concentrations des différents neurotransmetteurs sont représentées par les équations 1.5, 1.6 et 1.7. Elles dépendent des décharges neuronales (*firing rates*) précédemment calculés :

$$\frac{dC_{WX_e}}{dt} = \frac{tanh(\frac{F_W}{\gamma E}) - C_{WX_e}}{\tau E} \tag{1.5}$$

$$\frac{dC_{NX_e}}{dt} = \frac{tanh(\frac{F_N}{\gamma G}) - C_{NX_e}}{\tau G} \tag{1.6}$$

$$\frac{dC_{RX_e}}{dt} = \frac{tanh(\frac{F_R}{\gamma R}) - C_{RX_e}}{\tau R} \tag{1.7}$$

La concentration C de chaque neurotransmetteurs dépend de l'activité synaptique F_W de la population produisant ce dernier (donc présynaptique).

Tous ces paramètres obtenus par les fonctions et ses changements au cours du temps vont être enregistrés dans le fichier `Data_Storage.h` sous la forme `Array`.

3 - `<Random_Stream.h>`

Ce fichier comporte deux classes générant respectivement une série de nombres aléatoires suivant une loi normale et une série de nombres aléatoires suivant la loi uniforme.

`#include <random>` est une directive d'inclusion pour introduire le module de génération de nombres aléatoires.

4 - `<Cortical_Column.h>`

Sur la base des fichiers présentés ci-dessus, ce fichier déclare une classe `Cortical_Column` (citée dans le fichier `Sleep_Regulation.h`) qui comporte 22 fonctions, 51 paramètres affichés dans le diagramme 1.6.

`connect_SR(Sleep_Regulation& SR_Network)` permet la liaison avec `Sleep_Regulation`. Par cette fonction, nous pouvons appeler tous les paramètres de `Sleep_Regulation.h`.

La fonction `Update_Bifurcation_Parameters (void)` permet de lier les deux grandes parties comme dans la figure 1.2.

`std::vector<randomStreamNormal> MTRands;` déclare une variable `MTRands` représentant le bruit qui suivent la loi de Normale (*Noisy Input*)dans la figure 1.2.

Tous les paramètres utilisés pour les calculs sont également stockés dans des vecteurs.

5 - `<Cortical_Column.cpp>`

Ce fichier est établi selon l'article [Schellenberger Costa *et al.*, 2016] qui se charge de lister toutes les fonctions nécessaires à l'expression de Vp ou Vi. Vp ou Vi est l'évolution du voltage de la membrane par unité de temps au sein des populations *"Pyramidal"* et *"Inhibitory"*. Les deux formules principales sont les suivantes :

$$Vp = (-I_{Lp} - I_{AMPA}(s_{ep}) - I_{GABA}(s_{gp}) - C_m * I_{KNa})/\tau p \tag{1.8}$$

$$Vi = (-I_{Li} - I_{AMPA}(s_{ei}) - I_{GABA}(s_{gi}))/\tau i \tag{1.9}$$

I_{Lp} et I_{Li} est le courant de perméabilité au sein des populations *"Pyramidal"* et *"Inhibitory"*. I_{KNa} est le courant de potassium. $I_{AMPA}(s_{ep})$, $I_{AMPA}(s_{ei})$, $I_{GABA}(s_{gp})$ et $I_{GABA}(s_{gi})$ représentent respectivement les courants excitateurs des synapses glutamatergiques (AMPA) et les courants inhibiteurs des synapses GABAergiques. τp et τi est le temps de membrane constant pour les différentes populations. Les paramètres obtenus par les fonctions et ses changements au cours du temps sont enregistrés dans le fichier `Data_Storage.h`. La méthode RK4 est toujours utilisée ici pour calculer les changements sur une courte période.

6 - `<ODE.h>`

Ce fichier ne contient qu'une fonction qui permet de combiner les deux grandes parties `Sleep_Regulation` et `Cortical_Column`. ODE permet de résoudre numériquement les équations différentielles selon les formules précédemment citées à chaque pas de temps.

7 - `<Cortex_SR.cpp>`

Comme cette partie est le point d'entrée, elle ne contient qu'une seule fonction `int main(void)`. Cette fonction initialise les deux grandes populations `Sleep_Regulation` et `Cortical_Column` puis appelle la fonction `Cortex.connect_SR(SR)` pour connecter les deux parties. Ensuite, avec l'aide de la boucle du `ODE(Cortex, SR)` au cours du temps, la fonction effectue la simulation.

8 - `<Data_Storage.h>`

Dans ce fichier, nous retrouvons la fonction `get_data()`. Les données obtenues par toutes les parties sont stockées dans une tableau :

`std::vector<double> pData` qui inclut toutes les valeurs des variables à chaque pas de temps et pour toute la durée de la simulation.

`pData[0][counter] = Col.Vp [0];` Vp le voltage membranaire de la population de neurones pyramidaux

`pData[1][counter] = Col.Vi [0];` Vi le voltage membranaire de la population de neurones inhibiteurs (interneurones)

`pData[2][counter] = SR.f_W [0];` f_W la décharge neuronale de la population d'éveil

`pData[3][counter] = SR.f_N [0];` f_N la décharge neuronale de la population NREM

`pData[4][counter] = SR.f_R [0];` f_R la décharge neuronale du REM

`pData[5][counter] = SR.C_E [0];` C_E la concentration de noradrénaline

`pData[6][counter] = SR.C_G [0];` C_G la concentration de GABA

`pData[7][counter] = SR.C_A [0];` C_A la concentration d'acétylcholine

`pData[8][counter] = SR.h [0];` h homéostasie

`pData[9][counter] = Col.g_KNa [0];` g_KNa la force des synapses sodium-potassium

`pData[10][counter]= Col.sigma_p [0];` *sigma_p gain neuronal*

9 - `<Cortex_SR_mex.cpp>`

La dernière partie se charge de déclarer les paramètres du graphe et d'initialiser un fichier MATLAB pour tracer les courbes comme dans la figure 1.1 qui représente l'activité corticale. Nous pouvons compiler ce fichier avec *Matlab Executable(MEX)* pour générer un fichier en format `.mex`. Un fichier source de MEX en langage C++ ainsi que celui-ci, il existe toujours une directive `#include "mex.h"`. Dans ce fichier, la fonction d'entrée `main()` est changée par une fonction définie suivante :

```
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

mxArray est la structure des données définie par `mex.h`, *nlhs* est le nombre des variables de sortie, *plhs* est le vecteur des pointeurs des variables de sortie, *nrhs* est le nombre des variables d'entrée, *prhs* est le vecteur des pointeurs des variables d'entrée.

Par conséquent, cette fonction peut stocker toutes les variables de sortie et les variables d'entrée après la boucle de `ODE` dans le `mxArray`.
Afin de fonctionner ce fichier dans MATLAB, son fichier compilé est nommé comme `mexFunction` obligatoirement.

10 - `<Plots.m>`

Ce fichier peut extraire les données préparées dans `mxArray` pour dessiner le graphe (plot) dans MATLAB. Les variables de l'axe X et l'axe Y de chaque plot sont fixés par ses codes.

Sur les bases des analyses ci-dessus, nous pouvons établir l'architecture globale du modèle (figure 1.2).

1.2 Architecture et diagramme des classes

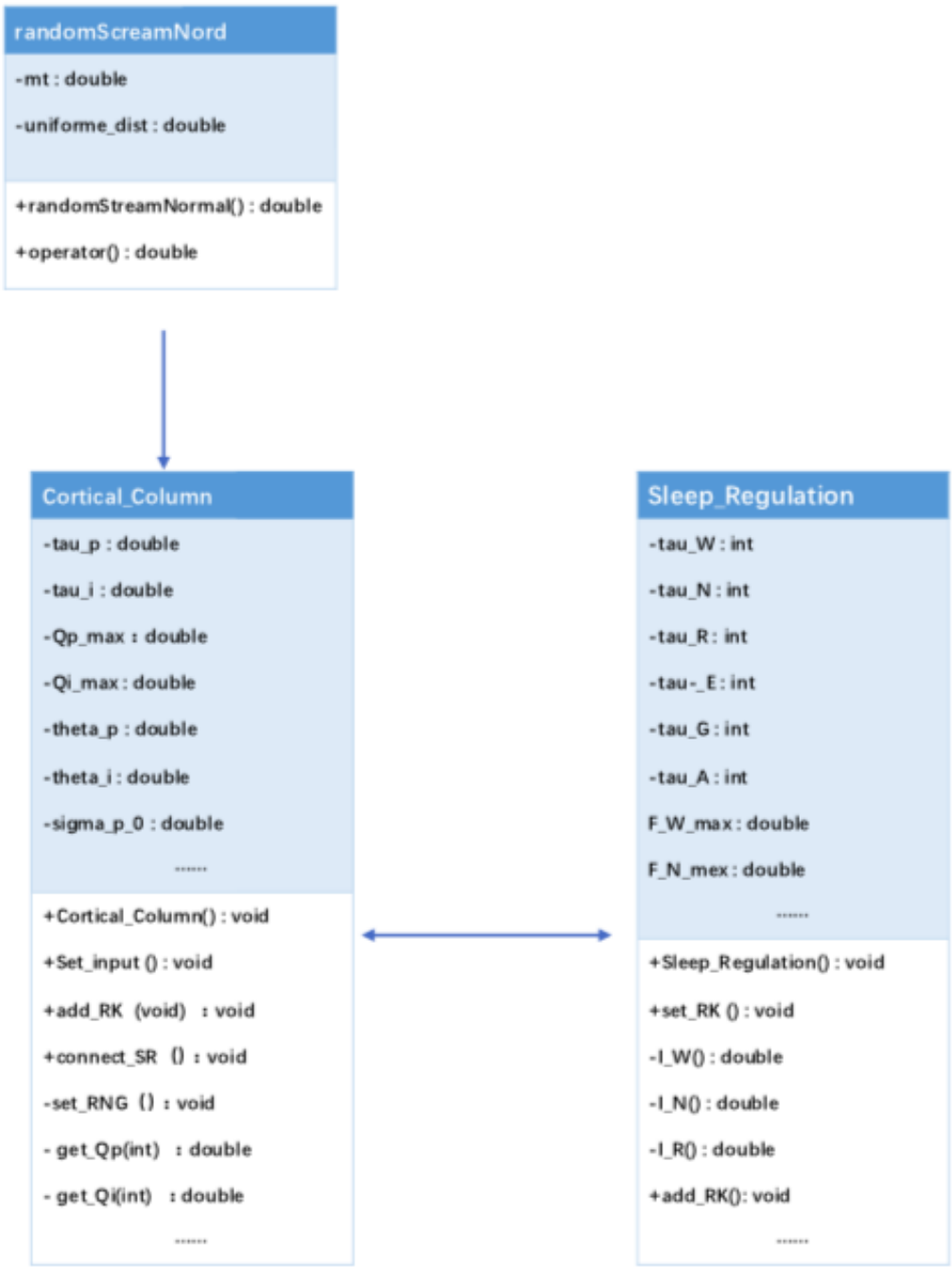

FIGURE 1.4 – Diagramme des classes

Noua avons utilisé 3 classes, cette figure est pas complexe,Nous avons donné quelques exemples de variables et de fonctions. (les diagrammes de classe sont trop longs, nous ne pouvons pas les relier ensemble dans un papier). la classe randomStreamNormal est dans le fichier 'header'(Random_Stream.h).
Elle est utilisé dans Column_Column.h. La classe Cortical_Column et la classe Sleep_Regulation utilisent les données de l'autre.

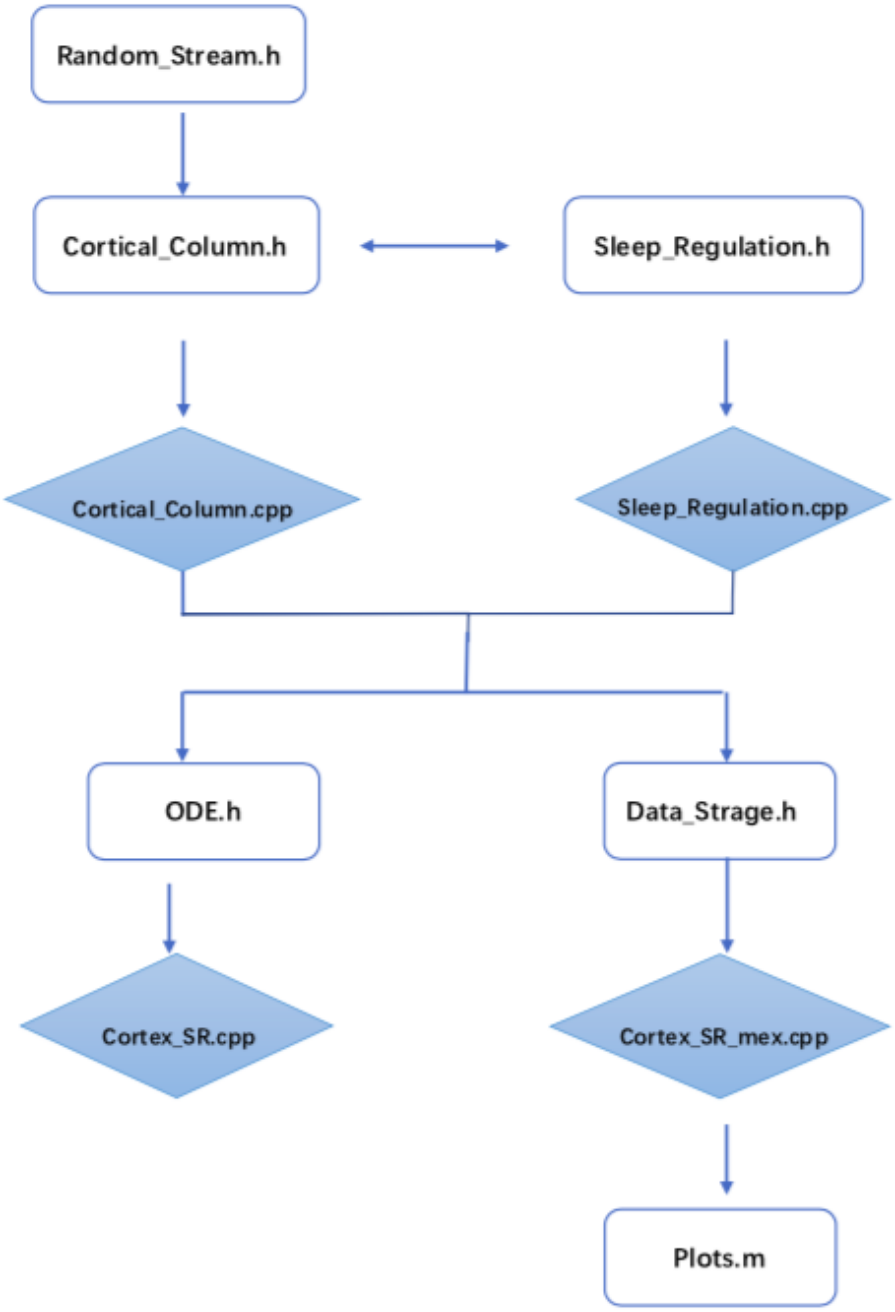

FIGURE 1.5 – Architecture du programme

Dans cette figure, il y a 3 fichiers 'header' qui ont 3 classes (normalement, les classes et les fichiers 'header' sont les même noms). Le ficher Cortical_Column.cpp utilise les fonctions viennent de Cortical_Column.h. Pareil pour les autres fichiers (Sleep_Regulation.cpp, Cortex_SR.cpp et Cortex_SR_mex.cpp), le sens de la flèche indique que les fonctions sont référencée. Au final, Cortex_SR_mex .cpp mélange toutes les fonctions et collecte toutes les données. Le fichier plots.m est de Matlab, c'est un langage qui est utilisé ici pour visualiser les données. On peut l'utiliser pour dessiner des images.

Il s'agit d'un diagramme appelé Cortical Column. Il se compose de trois parties. La première partie est le nom de la classe. La deuxième partie écrit toutes les variables. La troisième partie écrit toutes les fonctions. "+" Signifie public , "-" signifie privé.

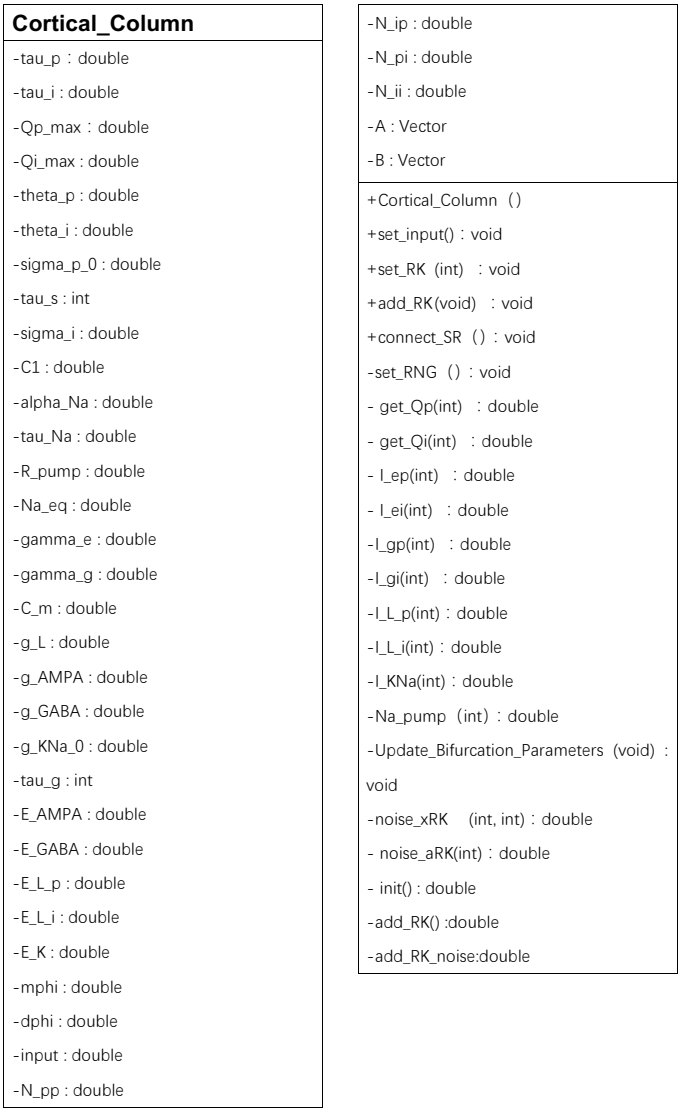

FIGURE 1.6 – Diagramme de la classe Cortical Column

Chapitre 2

Analyse du code Python et interface graphique

2.1 Architecture et fonctionnement

Le programme est articulé entre différents fichiers qui alimentent le fichier `Main.py`.

`Manageparameters.py` permet de charger les paramètres de tout un modèle via 2 fonctions :

- `readparameters()` qui lit un fichier `.txt` ,en extrait tout les paramètres et les sauvegarde dans un dictionnaire de dictionnaire.
- `writeparameters()` qui sauvegarde tous les paramètres utilisés pendant la simulation dans un autre fichier `.txt`.

`GUI.py` est le fichier python qui gère toute l'interface graphique. Il ne possède qu'une seule classe avec différentes fonctions permettant l'affichage des boites de dialogues, des paramètres, et de toutes les fonctions qui permettent au programme de fonctionner.

`SleepregulationOOP` est divisé en plusieurs classes :

- `Network` : permettant l'initialisation de la classe `Network`.Cette classe est utilisée pour gérer la simulation. Elle contient plusieurs fonctions, notamment pour l'initialisation, la création du bruit blanc gaussien, la mise en place des paramètres, le lancement de la simulation, la création des hypnogrammes, l'écriture des résultats dans un fichier, l'enregistrement, les méthodes de modification du réseau, et le débuggage.
- `NeuronalPopulation` : Classe permettant la représentation d'une population neuronale. La population neuronale est stockée sous forme d'un dictionnaire et différents paramètres, sous forme de fonction, peuvent s'y appliquer en lien avec la concentration de neurotransmetteur et leur bombardement sur la population.
- `HomeostaticSleepDrive` : Celle-ci permet une simulation du processus homéostatique intervenant dans la régulation du cycle sommeil/éveil par la création d'objets à comportement cyclique.
- `Connection` :Cette classe permet de générer des connexions entre les différentes populations de neurones. En effet, elle gère les concentrations des injections avec les poids associés.
- `Injection` : Cette dernière classe gère la nature des injections.

`graphic.py` : fichier contenant toutes les fonctions utilisées pour l'affichage des graphiques. Utilisation de `matplotlib`, et `tkinter` principalement. Les fonctions permettent l'affichage des fenêtres, leur aménagement, la création du graphique, des graphiques de moyennes, la comparaison avec les contrôles, la lecture des fichiers `.csv` et la création de graphiques qui en sont issus.

`Stats.r` : Contient le code nécessaire à l'analyse des données sous R.

`main.py` : `Main` permet (par l'importation des fichiers mentionnés ci-dessus) de faire fonctionner le programme capable de modéliser des injections de neurotransmetteurs sur des populations de neurones et d'en mesurer les effets directs sur le sommeil. Il offre alors un interface claire, avec une prise en main facile, directement dédiée à l'utilisateur avec notamment des menus dont les différentes sections sont cliquables par des boutons. Ainsi, le client peut régler les paramètres de sa simulation et choisir de faire apparaître différents résultats sous une forme statistique.

Nom des fichiers	Nombre de lignes	Nombre de fonctions et utilité
manageparameters.py	81	2 fonctions : readparameters : lis les paramètres des populations du modèle
Main.py	150	2 fonctions
GUI.py	511	28 fonctions
graphic.py	440	9 fonctions
SleepRegulationOOP.py	509	49 fonctions

2.2 Conformité avec le programme C++

Les liens entre le code C++ et python sont nombreux même si l'organisation est parfois un peu différente au sein des programmes. Les fonctions remplies restent très conformes.

Le premier fichier `CortexSR.CPP` joue le même rôle que `main.py`. En effet il est le fichier principal pour les tests de compilation et d'exécution des tests, avec des choix de simulation tout à fait similaire au fichier python.

Le second fichier `CorsterSRmex.cpp`, permet l'analyse statistique des simulations. Il s'accorderait donc avec les fichiers `Stats.r` ainsi que `graphique.py`. Ces derniers fichiers sont tout de même plus complets dans le cadre d'analyses descriptives et c'est pour cette raison que le `.cpp` fait appel à un fichier `plots.m` pour les représentations graphiques.

Le troisième fichier `CorticalColumn.cpp` est structurellement très apparenté avec `Sleepregulation.py` du programme python. Effectivement il permet lui aussi la modélisation des populations neuronales, des connexions entre elles et des différentes stimulations par les divers neurotransmetteurs. Les multiples régulations par les neurotransmetteurs sont elles aussi représentées sous forme d'itérations.

Le dernier fichier `Sleepregulation.cpp` tient lui aussi un équivalent python mais intégré dans le `Sleepregulation.py` mentionné précédemment. En effet, il est comparable à la classe `HomeostaticSleepDrive` puisqu'il modélise les cycles naturels du sommeil.

Finalement les fonctions de `Mageparameters.py` sont remplies par le fichier `Cortex.cpp`. Un autre différence notable est que de nombreux fichier `.h` sont associés au code `.cpp` afin de faciliter les modification du code source.

Chapitre 3

Réalisation

Précédemment, nous avons réalisé l'analyse d'un programme réalisé en langage C++ qui a fait l'objet d'une publication [Schellenberger Costa *et al.*, 2016]. La seconde partie a consisté en un même travail d'analyse mais portait sur un programme réalisé par des étudiants du Master de Bioinformatique de l'Université de Bordeaux en 2019 dans le cadre d'un projet de développement et de recherche. En effet, ils s'étaient basés sur le programme publié [Schellenberger Costa *et al.*, 2016] pour en établir la traduction en un autre langage et en y ajoutant une interface graphique [Darnige *et al.*, 2019]. Ce projet a été réalisé sous la supervision du Dr Héricé Charlotte qui nous a confié la suite du développement.

Le programme ainsi traduit convenant à l'équipe de recherche pour laquelle il a été réalisé, nous avons décidé, en accord avec notre client, de poursuivre notre travail sur la base de ce dernier. En effet, le programme est fonctionnel pour l'étude du modèle chez les humains et est suffisamment intuitif pour que nous poursuivons dans la même lignée. Cela étant, notre travail consistera a y intégrer le modèle pour les rongeurs de sorte que toutes les fonctionnalités pré-existantes soient utilisables aussi bien pour ce modèle que pour le modèle humain. Dans ce programme, les paramètres du modèle humain sont récupérés depuis un fichier texte. Par conséquent, nous tenterons d'établir le même type d'utilisation pour l'intégration du modèle chez le rongeur. Nous ne savons pas encore précisément quelles modifications nous devrons apporter au programme existant en terme de programmation pure. Cependant, nous devrons faire en sorte que chaque utilitaire demeure fonctionnel.

Une fois que nous aurons procédé à l'adaptation des fonctionnalités de base au nouveau modèle, nous travaillerons à adapter les autres outils de sorte que la simulation de tests lésionnels et d'injections soient possibles depuis l'interface graphique actuelle. Il en est de même pour l'outil statistique.

Selon le temps nécessaire à la réalisation de ces adaptations, nous pourrons nous concentrer sur l'amélioration du programme en terme de conformité biologique. En effet, notre modèle ne tient compte que de trois populations neuronales. Même s'il est biologiquement acceptable, il serait plus réaliste de l'adapter au modèle à six populations neuronales [Fleshner *et al.*, 2011]. Ainsi, il sera nécessaire d'ajouter des équations différentielles et d'adapter les deux modèles existants.

Bibliographie

[Baty, 2018] Baty, H. (2018). Approche numérique à l'usage du physicien pour résoudre les équations différentielles ordinaires.

[Darnige *et al.*, 2019] Darnige, E., Grimaud, A., Gruel, A. et Kuntz, A. (2019). Developing a computational model of paradoxical sleep.

[Diniz Behn et Booth, 2012] Diniz Behn, C. et Booth, V. (2012). A fast slow analysis of the dynamics of rem sleep. *Society for Industrial and Applied Mathematics*.

[Fleshner *et al.*, 2011] Fleshner, M., Booth, V., B.Forger, D. et Diniz Behn, C. (2011). Circadian regulation of sleep-wake behaviour in nocturnal rats requires multiple signals from suprachiasmatic nucleus. *Philosophical Transactions of The Royal Society*.

[Schellenberger Costa *et al.*, 2016] Schellenberger Costa, M., Born, J., Claussen, J.-C. et Martinetz, T. (2016). Modeling the effect of sleep regulation on a neural mass model. *Journal of Computational Neuroscience*.