Programming Project Report

Developing a Computational Model of Paradoxical Sleep

Eden Darnige
Arthur Grimaud
Amélie Gruel
Alexia Kuntz

Supervised by Dr. Charlotte Héricé

May 2019

Master de Bioinformatique de Bordeaux

Abstract

Sleep regulation is an important biological mechanism involved in mental health, physical health, and safety. Many models have been conceived in order to simulate the different phases of the sleep/wake cycle to address these domains.

Dr. Héricé and Dr. Sakata's team are working on a sleep regulation model based on three neuronal populations. The reference model has been primarily developed by Dr. Costa *et al*, published in 2016. The program presented in this report has developed upon the model in a more flexible manner.

The same equations from Dr. Costa *et al*, 2016, are retained and applied to a selection of interconnected neuronal populations and cycles. In this way, the program is applicable to any type of sleep regulation mode, whether it is based on three, five or more populations, and whether or not it includes circadian oscillations or homeostatic sleep drive.

The program developed by the students of Bordeaux's Master of Bioinformatics achieves the goal of model flexibility and more. The use of a graphical user interface strengthens the model's user-friendliness and usability. Through this interface, different model parameters can be easily loaded. The corresponding results can be saved, displayed and statistically processed by a few clicks of a button. The user will also be able to study the results in the case of simulating lesions or microinjections.

Keywords : Computational model · Sleep regulation · Wake · REM sleep · NREM sleep · Homeostatic Sleep Drive · Neurotransmitters · Firing rates · Injections · Lesion · Neuronal population

Contents

Introduction

Nearly two centuries of research on sleep has led to a better understanding of the mechanisms that control sleep and wakefulness cycles. Despite the importance of sleep in biological maintenance, the functionality of the different stages of sleep, rapid eye movement (REM) and non-rapid eye movement (NREM, including three sub-steps in humans), has not yet been fully discovered.

Many mathematical models have been developed in order to better comprehend the role of sleep and its different phases. Dr. Héricé of the Shuzo Sakata team at the Strathclyde Institute of Pharmacy and Biomedical Sciences draws upon previous research completed by Dr. Costa *et al.*, as well as other existing mathematical and biological resources in order to design computational models representing the oscillations between wake, REM and NREM sleep in rodents.

Dr. Costa and colleagues [1] produced and made public a program in C++ and MatLab modeling the homeostatic cycle of three neuronal populations and its influence on the sleep cycle in 2016. In this document, the work conducted by four bioinformatics students on the development and adaptation of this computational model as part of a project within Bordeaux's Master of Bioinformatics is presented.

To make the model more accessible to researchers on Dr. Héricé's team, the model was translated into Python with a graphical user interface. In addition, other features, such as the inclusion of homeostatic sleep drive as described in Diniz *et al.*, 2011 [2] and certain statistical analyses, were included.

Ultimately, this program will be used to predict the role of different types of neurons in the sleep cycle and to study the effect of lesions between different neuronal populations as well as microinjections of certain neurotransmitters on REM sleep and the sleep cycle in rodents.

The first chapter will introduce the project analysis, including its context, the corresponding state of the art and the model's functional and non-functional requirements.

The second chapter will demonstrate the model's design, and just how the different requirements will be met.

The third chapter will exhibit the model's completion, with the details of its architecture and explanations of the code. Potential further developments of the model and improvements will also be discussed.

The conclusion will succinctly summarize the results obtained after restating the initial objectives.

Finally, the bibliography and the appendix can be found at the end of the document.

Chapter 1

Analysis

1.1 Context

Dr. Charlotte Héricé is a member of Dr. Shuzo Sakata's team within the Strathclyde Institute of Pharmacy and Biomedical Sciences (SIPBS), a research center located within Strathclyde University, in Glasgow, Scotland, UK. This international institute shelters a School of Pharmacy and many Masters of Science. Numerous research teams work there on subjects related to the development of new medicines and treatments [3].

Dr. Sakata's team aims to understand how sensory information is processed by brain circuits, in order to develop better strategies for the improvement and restoration of sensory abilities, with an emphasis on hearing [4].

Dr. Héricé specializes in paradoxical, or REM, sleep, and is designing a computational model of the involved brain structures. In order to study REM sleep, the team mainly uses *in vivo* ensemble recording, behavioral approaches and optogenetics.

Optogenetics consists of the insertion of genes coding for a photosensitive protein within a rat's genome. This enables the specific activation of certain neurons, followed by the observation of its impact on the rat's behavior.

This technique has many benefits, such as the fact that the animal is its own control sample, but it also presents some major flaws. Indeed, a lot of preparation and thus a lot of time and human and material resources are needed. It demands complicated electronics and generates a heavy animal cost since their brain then undergoes immunohistochemical studies to determine the cellular impact of the activated protein.

Dr. Héricé aims to minimize these drawbacks by designing her computational model, modeling the neuronal network involved in the REM sleep generation and maintenance.

1.2 State of the art

In mammals and some bird species, sleep is subdivided into two phases, REM and NREM [5]. REM sleep is associated with high brain activity and muscle atonia. NREM sleep is a phase of sleep in which brain activity is lower and is characterized by low-frequency waves on an electroencephalogram (EEG) [6]. Several neuronal populations located within the brainstem and hypothalamus have been identified as being involved in transitions between wakefulness, REM sleep and NREM [7]. Neurotransmitters secreted by these populations, such as serotonin and acetylcholine, have also been identified [8]. Thus, several networks of interactions between these neuronal populations involved in sleep regulation have been proposed (e.g [9]).

Thanks to the knowledge of the neural networks involved in sleep regulation, several mathematical models describing transitions between different states of consciousness have been formulated. One model of interest in this project was created by Dr. Diniz Behn *et al.* [10], and describes the activity of each neuronal population using their firing rate (FR). Depending on the FR of a neuronal population, the concentration of secreted neurotransmitters corresponding to that population can be determined. The neurotransmitter secreted by a pre-synaptic population may have an inhibitory or excitatory effect on the post-synaptic population, thus modifying the FR of the post-synaptic population. In the model described by Dr. Diniz Behn and colleagues, the specified neural populations have a formalism close to biological reality. Each of these regions promotes a state of consciousness (wakefulness, REM sleep or NREM sleep), so it is possible to determine the state of consciousness of the model according to the activity of these regions.

1.2.1 Two-process model

Two processes affecting transitions between states of consciousness have been identified by [11]. The S process, or sleep drive, is a phenomenon that increases the propensity to sleep during the awakening phase and vice versa. Process C, or circadian rhythm, increases the willingness to sleep or wake up depending on external factors. The S process has been taken into account in the simulation of Dr. Diniz Behn and colleagues [10] and is implemented through a variable acting on the neuronal population corresponding to the Ventrolateral Preoptic Core. The simulation of process C has also been discussed in other models [12, 13].

1.2.2 Micro-injection simulations

Thanks to the formal structure using the FR of neural populations and neurotransmitter concentration, it is possible to simulate micro-injections of agonists or neurotransmitter antagonists in a target population. Simulation of the effect of an antagonist on a neurotransmitter is performed by decreasing the concentration of this neurotransmitter in the target population and conversely for the agonist [10].

1.2.3 Simulation of the electroencephalogram profile

In *in vivo* experiments, the state of consciousness is determined by an EEG that measures the activity of the cortex. A model aims to simulate EEG profiles from neurotransmitter concentrations emitted by neuronal populations [1, 14]. These EEG simulations allow the simulation results to be compared with those obtained using experiments *in vivo*.

1.2.4 Three population model

A model of a simplified sleep regulation network has been developed by Dr. Diniz Behn *et al.* [12] as seen in Figure 1.1. In this model, the number of neural populations was reduced to three by grouping populations promoting a similar state of consciousness among themselves. In this three-population model, the same format as that described by Dr. Diniz Behn and colleagues [10] was used.

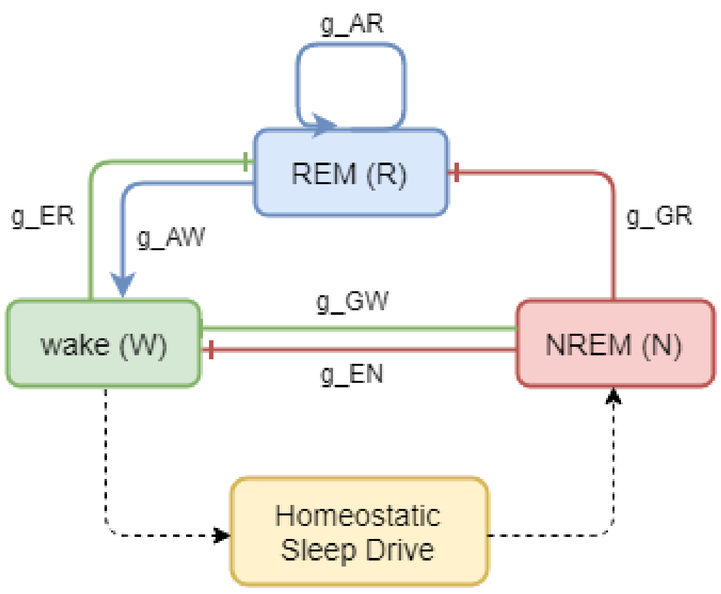


Figure 1.1: Diagram representing the connectivity within the 3 population model. Dotted lines indicate connectivity between homeostatic sleep drive and neuronal populations. Solid lines with arrows indicate excitatory synaptic inputs between neuronal populations, while solid lines with cross ends indicate inhibitory synaptic input between neuronal populations. Finally, g indicates the synaptic weight of the neuromodulator.

1.3 Functional and non-functional requirements

Several functional and non-functional requirements have been identified in order to develop this model. The functional requirements describe the elements related to the technical functionality of the model, while non-functional requirements specify certain criteria that can be used to judge the operation of the model.

1.3.1 Model correctness and efficiency

First and foremost, the model has to produce biologically acceptable results in a relatively short time-span. To verify the correctness, the results will be compared to those of Costa *et al.*, 2016 [1], which will serve as a benchmark. Therefore, the results of this publication must be retrieved by running their program on one of the CREMI computers licensed by MatLab.

The model should be flexible in order to execute any user requests. For example, the number of neuronal populations involved and their parameters has to be modifiable, as well as the connections between those populations or the different cycles influencing them.

Moreover, the model has to produce results for different specific initial conditions. It has to be able to compute the effect of agonists and antagonists injections, as well as lesions.

1.3.2 Inputs and outputs

The model reads the parameters from an input file and must produce the results in a readable format easily accessed by the user in order to perform further analyses. There are different types of results generated by the code: population firing rates, neurotransmitter concentrations, homeostatic sleep drive, and a hypnogram describing the current state (wake, REM, NREM). The output file should be customizable in terms of which information is saved and the file name.

1.3.3 Model accessibility

1.3.3.1 Code comprehension

The program has to be coded in an accessible language, if possible, one that is frequently used by the client. Every function has to be explained using comments in English to communicate with the biologists of the anglophone lab in Scotland. A user with little to no experience in computer science should be able to easily understand it.

1.3.3.2 Launching the program

The program must be easily launched, for example, by executing only one script instead of several python and R scripts successively.

Furthermore, Dr. Héricé's laboratory mainly uses the Windows environment. However, the main users will be the client and her Principal Investor who work on Mac; the program has to be easily executable on this Unix environment.

1.3.3.3 Modifying the parameters

The different parameters must be easily modifiable by the user. This program will be used by researchers investigating the impact of different initial parameters.

1.3.4 Treatment of results

1.3.4.1 Visualization of results

The user can choose to visualize the results in the form of three graphs. This includes a graph depicting the curves of neurotransmitter concentrations, a graph illustrating the firing rates, and a hypnogram which can be used to identify the rodent's different sleep/wake cycle phases (wake, REM sleep, NREM sleep).

1.3.4.2 Statistical analysis of the results

Statistical tests, in particular Analysis of Variance (ANOVA) and Tukey's HSD Post Hoc test, as well as graphical representations should be created from the results in order to assess the significance of the different cycles within an experiment. One-way ANOVA analyzes differences between mean groups in cycle samples. If a significant difference is detected, it is logical to continue with Tukey's HSD Post Hoc test to determine which of the cycles are different from each other.

1.3.5 Communication

Since Dr. Héricé is located in Glasgow, Scotland, whereas the model was developed in Bordeaux, France, it was necessary to find an efficient way to communicate the different scripts and results. The software development platform `GitHub` was used to collaborate within the group and with the client.

Chapter 2

Conception

To ensure the biological correctness of the model, the program is based on an existing mathematical model. Dr. Costa *et al.*'s initial C++ and Matlab code, available on `GitHub` [1], has been used as the main reference.

2.1 Programming languages

2.1.1 Python

This model has been adapted in Python, an interpreted, high-level programming language used in Dr. Héricé's laboratory. It supports functional, procedural and object-oriented programming which has been used throughout the model adaptation.

Currently, two versions of Python are in use: Python 2 and Python 3. Indeed, Python 2 is still relevant because of legacy systems written in this language and specific libraries not yet available on Python 3.

This program is developed in Python 3 since it has been written from scratch and all the libraries required are available on Python 3. Furthermore, Python 3 adoption is growing rapidly among the programming community and is frequently used by Dr. Héricé's team.

2.1.2 R language

For the statistical analysis of the results, R is used. R is a powerful environment for biological data analysis with a huge set of libraries available for different statistical type analysis, and it is a preferred language of Dr. Héricé and the team of biologists.

2.2 Libraries, modules, and packages

2.2.1 Scientific computing

The `NumPy` library is very useful in mathematical equations as well as in data manipulation. It functions by providing high-performance, multidimensional arrays as well as the basic tools that compute and manage the arrays. These arrays take up less memory as compared to Python lists. Tools from this library were used in the core of the model to create random noise and to complete the neurotransmitter concentration and firing rate equations.

2.2.2 Plotting in Python

Costa *et al.* used MatLab to produce their graphs. `Matplotlib` is used to provide an environment that is an effective open source alternative for MatLab.

A module of the 2D plotting library `Matplotlib`, `Pyplot`, is used to generate the neurotransmitter concentrations, firing rate, and hypnogram graphs.

As `NumPy` and `MatPlotLib` are both used, the `Pylab` module seemed to be interesting. Unlike `Pyplot`, `Pylab` gets installed alongside `Matplotlib`, and imports `Pyplot` and `NumPy`. It provides specific plotting features which are convenient when arrays are used.

Finally, `Graphviz` has been used to generate a graph representing the connectivity within the model, similar to Figure 3.

2.2.3 Plotting in R

`Ggplot2` was used to visualize the statistical analysis data in R. In order to view all three output graphs in one file, the package `gridExtra` was required.

2.2.4 Graphical User Interface

As Python's standard graphical user interface, the Tk-interface (`Tkinter`) module was used to allow users to enter parameters, upload files, generate graphs, and do statistics all from one tabbed window. The use of buttons and text-boxes in order to configure the model facilitates the user experience.

`ttk`, a `Tkinter` submodule, provides access to themed widgets. Some are already available on `Tkinter`, but `ttk` proposes six new widgets. `Notebook`, one of these six, has been used to manage a collection of window and to display one at a time.

To select an existing file and to save a file within the Graphical User Interface, `filedialog` has been used. It's a `Tkinter` submodule providing access to the directory structure of an operating system in dialog windows.

2.2.5 The Python Standard Library

2.2.5.1 OS Module

To indirectly call the R script within Python, the OS module is used. The OS module provides an interface with the underlying operating system that Python is running on. The operating system then launches the R script.

2.2.5.2 CSV

CSV (Comma Separated Values) format being the most common import and export format for spreadsheets and databases, the `CSV` of the `Python Standard Library` is imported in order to manage the results.

2.2.5.3 Math

Several mathematical functions not available on the Python Language Reference have been needed to write the model. To have access to them and to perform some statistical analyses, the `math` module has been used.

2.2.5.4 Graphivz

The `Graphviz` package is used to create graphic descriptions in the DOT language. It can be used to visualize the network within the model.

2.3 Inputs

In the initial C++ code, all of the parameters were specified in the script, constraining users who wanted to modify the parameters to browse through all of the files.

Since biologists with very rudimentary knowledge in computing might use this program, it has to be more accessible than the original one by allowing for easily modifiable inputs. Instead of putting all of the parameters in the script, a text file regrouping them all and a Python script in order to read this file were created.

2.4 Outputs

The results of the model had to be saved in a CSV file. This way, the user is able to visualize graphs from precedent simulations. It avoids having to launch the same simulation over and over in order to obtain the graphs. The user can use freshly generated results as well.

As mentioned above, graphs are generated depicting a hypnogram and the evolution of the firing rates and neurotransmitter concentrations over time. The data has to be stored in order to reuse it.

Statistical analyses of the results need to possible for biologists. Indeed, it allows them to compare the results of different simulations, and to see the significance of the parameters. Sleep cycles can be compared through the visualization of graphs, notably bar graphs, generated by `R`.

2.5 Sleep regulation model

2.5.1 Mathematical Model

2.5.1.1 Neuronal population equations

Two variables are used to describe a neuronal population. First is the firing rate, which is the action potential frequency in the population. Second is the concentration of released neurotransmitters by the neuronal population. The mathematical model that defines the firing rate and neurotransmitter concentration values over time has been formulated by [10]. In this model, the firing rate equation (2.1) is the following:

$$F'_x = \frac{X_{max} - (0.5\left\{1 + \tanh\left[\left(\sum_i g_{i.x}C_i - \beta_x/\alpha_x\right)\right]\right\}) - F_x}{\tau_x} \tag{2.1}$$

X_{max} is the maximal firing rate in Hz of a neuronal population , with x referring to a specific neuronal population. The constant α_x governs the slope of the $tanh$ sigmoid function as shown in Figure 2.1. β_x determine the activation threshold of the neuronal population. τ_x is the time constant determining the activation rate of a neuronal population. The expression $\sum_i g_{i.x}C_i$ is the sum of the weighted neurotransmitter concentrations of the pre-synaptic populations. The concentration of released neurotransmitter is strongly correlated to the firing rate of the population, the following equation (2.2) has been used to determine the neurotransmitter concentration:

$$Ci'_x = \frac{\tanh(F_x/\gamma_i) - C_i}{\tau_i} \tag{2.2}$$

As for the firing rate function, τ_i is the associated time constant and γ_i determines the slope of the *tanh* function similarly like α_x. F_x refers to the firing of the neuronal population in which the neurotransmitter concentration is calculated.

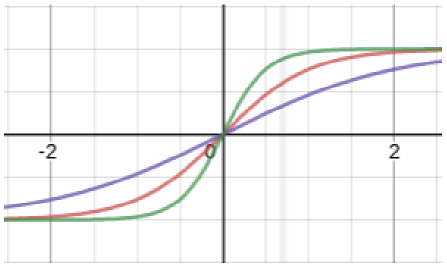


Figure 2.1: Influence of α constant on the equation $f(x) = \tanh\left[(x - \beta_x)/\alpha_x\right]$ (Green curve: $\alpha = 0.5$, Red curve: $\alpha = 1$, Blue curve: $\alpha = 2$)

2.5.1.2 Homeostatic sleep drive equation

The homeostatic sleep drive is a force that increases the propensity to sleep when the model is in the wake state. This force is represented through this variable and is determined by the following equation (2.3).

$$h' = H\left[\left(\sum F_x\right) - \theta_w\right]\frac{(1-h)}{\tau_{hw}} - H\left[\theta_w - \left(\sum F_x\right)\right]\frac{h}{\tau_{hs}} \tag{2.3}$$

The expression F_x represents the firing rate of wake-promoting neuronal populations. H corresponds to the Heaviside step function: this function returns 1 if the parameter is positive and 0 if the parameter is negative. tau_{hw} and tau_{hs} are two constants that are used to adjust the wake or sleep-promoting effect of the homeostatic sleep drive. θ is the threshold of transitions between sleep-promoting and REM-promoting effect of the homeostatic sleep drive.

2.5.1.3 Injection equations

The injections of an agonist and antagonist of a neurotransmitter cause respectively the increase or decrease of the neurotransmitter effect in the postsynaptic population. To simulate the effect of injection, the mathematical model formalized in [10] was used. In the injection model formalism used, the agonist or antagonist concentration varies according to a bolus administration. A bolus injection consists of the injection of the full dose at a single time. Thus, the decay of agonist or antagonist concentration over time was taken into account. The administered drug decay is calculated thanks to the equation 2.4, with τ being the constant governing the decay rate.

$$P'_i = -\frac{P_i}{\tau_i} \tag{2.4}$$

The injection of a neurotransmitter agonist does not only modify its effects, but the endogenous release of neurotransmitter is also affected. To simulate this effect, the weight of the neurotransmitter concentration was replaced by the variable m_i. This variable value is determined by the following conditions in 2.5.

$$(P_i \leq i_{Min} \Rightarrow m_i = 1) \wedge \left(P_i > i_{Min} \Rightarrow m_i = 1 - \frac{P_i - i_{Min}}{i_{Max} - i_{Min}}\right) \tag{2.5}$$

Both the agonist and antagonist injection simulations use the equation 2.4. The increasing or decreasing effect of the targeted neurotransmitter takes effect on the sum of weighted pre-synaptic neurotransmitter of the postsynaptic population, by modifying the concentration of the targeted neurotransmitter. The effect of an agonist is modeled by the equation 2.6. The injection of an antagonist in modeled by the equation 2.7. C_i is the concentration of the endogenous pre-synaptic neurotransmitter.

$$C_i(t) = m_i(t)C_i(t) + P_i(t) \tag{2.6}$$

$$C_i(t) = [1 - P_i(t)] - C_i(t) \tag{2.7}$$

2.5.2 Equation resolution methods

The approximated result of the previously described ordinary differential equation (ODE) has been obtained thanks to iterative resolution methods. Two methods were tested, namely the Euler's method and Runge-Kutta method of the fourth order (RK4).

2.5.2.1 Euler's method

The Euler method uses a discretization approach to approximate the result of an ordinary differential equation. This method implies the declaration of an initial condition, from this initial value the next approximate points of the function at $t + \delta t$ is determined by using the tangent of the curve at the time t. Then all the values of the equation at $t + n * \delta t$ are calculated iteratively. This method has the benefit of being simplistic, but it implies the use of a relatively small δt in order to limit the deviation from the original solution of the ODE.

2.5.2.2 Runge-Kutta 4th order method

While the Euler's method uses only the tangent at $t+$ to determine the value of the ODE at $t + \delta t$, the RK4 method requires the calculation of the tangent of the curve at 4 intervals between t and $t + \delta t$, then uses the weighted average of these tangents to calculate the value of the ODE at $t + \delta t$. As is for Euler's method, the RK4 method implies the determination of the initial values to start with. Due to the multiplication of operations to get the value at $t + \delta t$, the RK4 method requires more calculation time than Euler's method for the same δt. However, the deviation from the ODE solution is reduced with the RK4 method which able to use longer δt, thus reducing the computation time.

2.5.3 Noise

Random noise can be introduced into the system to generate more robust and biologically acceptable results. Additive white Gaussian noise (AWGN) has a continuous, normal distribution among frequencies. AWGN values can be calculated by generating random samples around a given mean with a set standard deviation. The option to set the mean and standard deviation of the noise depending on the model is included in the parameters.

In this model, the noise is measured in Hz and can be directly added to the result of the next step firing rate equation.

2.5.4 Object-oriented programming

The model translates *Costa et al.'s* C++ code to Python using object-oriented programming. Contrary to the C++ reference program, different classes were created corresponding to different biological elements such as neuronal populations, connections, homeostatic sleep drive, etc., in order to implement the sleep regulation model in a flexible way.

The classes were designed as such to be similar to the simulated biological system. The `NeuronalPopulation` and `HomeostaticSleepDrive` classes are analogous and will be referred to as "compartments." A compartment class is composed of one or more variables that describe its state. The equations that govern the variable's changes over time are also stored in the compartment's objects. In the biological system, neuronal populations are interconnected throughout synaptic connections, the class `Connection` simulates these synapses and enables the compartments to retrieve the value of other compartments' variables. Connectivity between additional populations can be entered in the parameters.

The whole compartment-connections system, as well as the simulation parameters, are managed through the `Network` class. `Network`'s superclass `NetworkGUI` contains additional methods that are used to generate the graphical user interface.

The classes from the reference model developed by Costa *et al.* are depicted in Figure 2.2. There have been a number of additions and modifications in the development of the model in Python 3 in order to improve its flexibility and capabilities. These differences can be seen when comparing this architecture to the class diagram represented in Figure 2.3: this class diagram illustrates our program, although it omits the object attributes used to store equation constants in `HomeostaticSleepSDrive` and `NeuronalPopulation`.

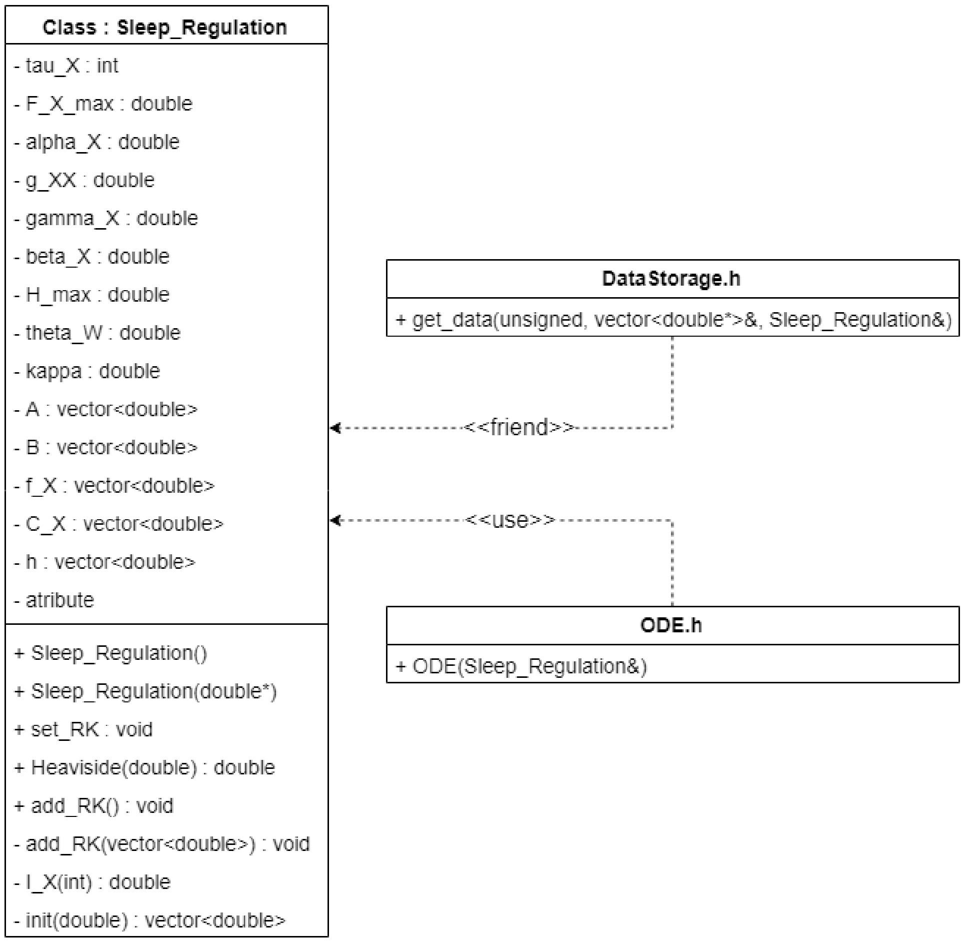


Figure 2.2: Diagram of model architecture from Costa *et al.* 2016.

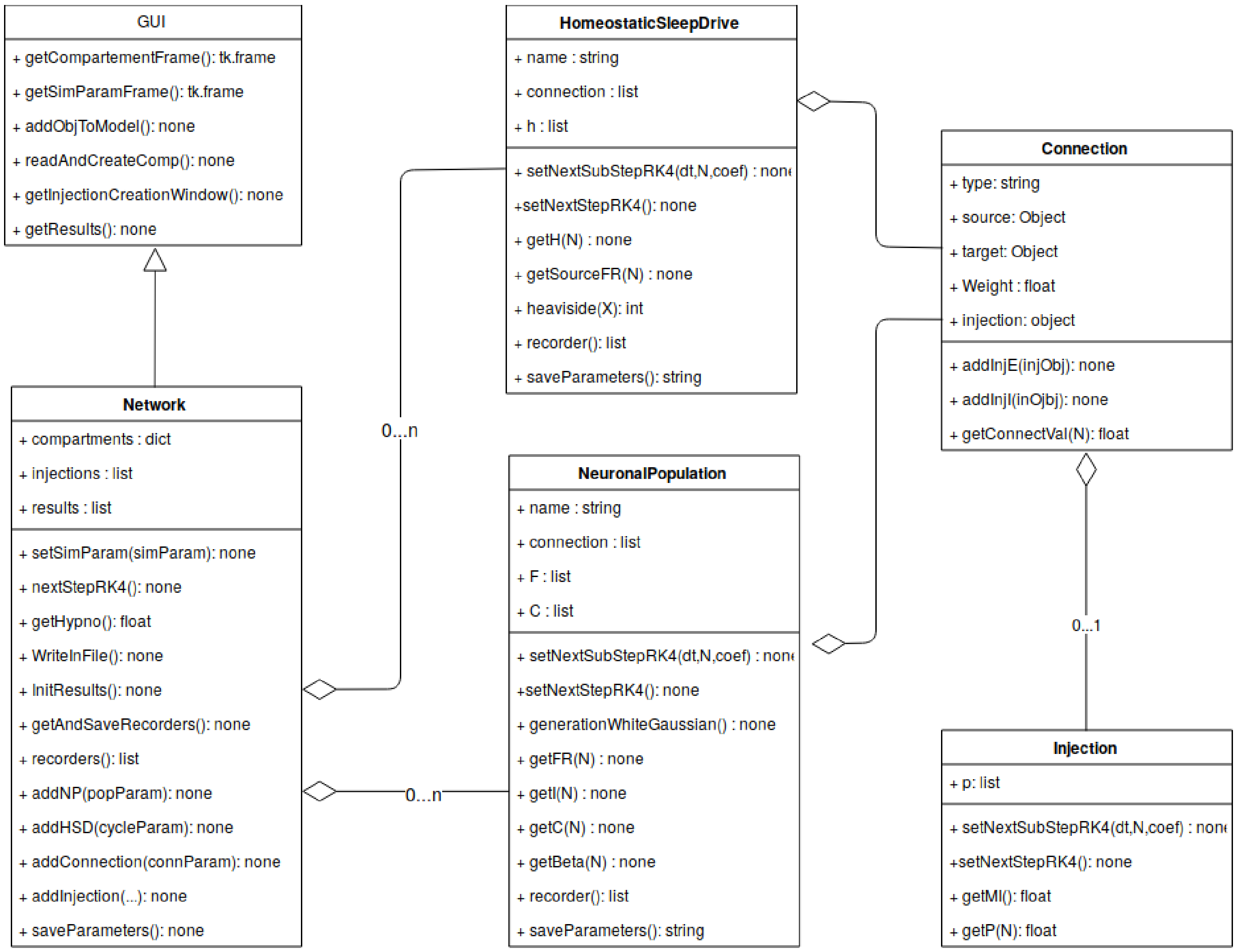


Figure 2.3: Class Diagram of the sleep regulation model presented in this document.

Chapter 3

Implementation

The program operating the model consists of five interlinked scripts written in Python 3 and one R script, as seen in Figure 3.1.

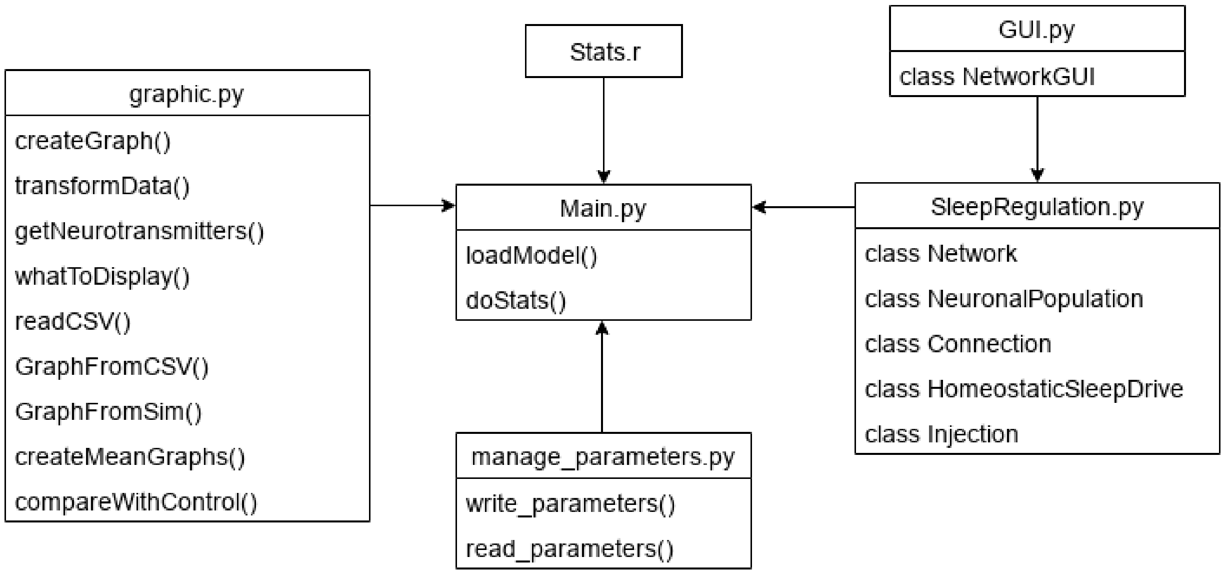


Figure 3.1: Diagram representing the different interlinked files of the program and the corresponding classes and functions.

The file `Main.py` imports the functions, classes and importations defined in the five other scripts. It returns the information obtained from one file to another in order to run the model correctly. It also initiates the graphical interface by importing `TKinter` and its packages `ttk` and `filedialog` (see 2.2.4) in order to create the master window and its different tabs.

Once the program is launched, `manage_parameters.py` handles the parameter inputs and the saving of the parameters (see 3.2).

These parameters are taken into account by `SleepRegulation.py`, which creates the different elements (neuronal populations, cycles, connections, etc.) and manages the different equations. This script imports `NumPy` and `math` (see 2.2.5 and 2.2.1) in order to realize complex mathematical calculations; it also imports the `CSV` (see 2.2.5) which eases the saving of the results as a .CSV file.

`GUI.py` is imported into the previously mentioned `SleepRegulation.py` script and handles the graphical interface linked to the model: it allows the user to modify the parameters, display the different elements, etc. In order to do so, it imports the `TKinter` library and its package filedialog (see 2.2.4). It also imports the library graphivz (see 2.2.5), thus giving the user the possibility to visualize the different compartments of the model and their connections in the form of a diagram.

graphic.py lets the user visualize the results according to their needs (see 3.3.3). It creates the different graphs thanks to the library MatPlotLib and its packages PyLab and PyPlot (see 2.2.2). Reading the results files saved in .CSV format is eased by the importation of the CSV library. It also computes the standard deviations and the standard errors of the mean in the mean graph, as seen in 3.12, using the Numpy package (see 2.2.5).

Statistics can be performed through the script Stats.r, which is used to process the significance of the results and returns a graph and text files as output of the statistical analyses (see 3.3.2). This script is called by the main script even though it is coded in R thanks to the os module (see 2.2.5). It uses the packages ggplpot2 and gridExtra (see 2.2.3).

3.1 Sleep regulation model

The three population model has been efficiently implemented thanks to the flexibility of the program, which has been achieved by using object-oriented programming.

3.1.1 Equation implementation

3.1.1.1 Equations and variable storage

Each equation has been implemented in separate methods. Within the Neuronal population class, the firing rate and neurotransmitter concentrations are in the GetF() and GetC() methods respectively. In the Homeostatic sleep drive class, the equation has been set in the getH() method. The variables F, C and h are stored as the object's attributes. In order to use the RK4 ODE resolution method, the attributes used to store variables are lists of length 5, with the four last elements of the list being used to store the 4 iterations necessary in the RK4 method.

3.1.1.2 Method "Next Step"

The compartment variables are updated by calling "next Step" methods, these methods take in and calculate a parameter at time period δt and set the new values of the variables at $t + \delta t$ from the initial value of the variable. The procedure used to obtain the next step variable values differs depending on the ODE resolution method used.

Using Euler When the Euler method is performed, the methods of the compartment classes, nextStepEuler() are used. This method calculates the variable's next value by applying the Euler method calculation of the following form:

$$variable = variable + \delta t * ODE \tag{3.1}$$

The value of the variable is stored as the first element of the variable list.

Using Runge-Kutta of the fourth order In order to achieve the RK4 resolution method, two next step methods are used. The setNextSubStepRK4() method takes in three parameters the time period of the step dt, the RK4 iteration number N and the coefficient of the iteration.

This method is called four times successively in each simulation step. The four last elements of the variable list are calculated, and then the `setNextStepRK4()` calculates the weighted average of the four Runge-Kutta iterations and stores this value as the first element of the variable list. It is during this step that noise is added as a parameter from the `additiveWhiteGaussianNoise()` method (see 3.1.4) for neuronal population instances.

3.1.2 Connections

In the mathematical model, the connections between the neuronal populations are established in the firing equation where the weighted sum of the pre-synaptic neurotransmitter concentration is used. The homeostatic sleep drive connections are made thanks to the beta variable in the firing rate equation. With the aim of creating a flexible network, a connection class was created to manage all of the different types of interactions.

3.1.2.1 Connection class

The `Connection` object is characterized by four main attributes. The type of connection is stored as a string in the attribute *type*. The *source* and *target* attributes are used to store references to the compartment objects implied in the connection. The weight of the connection is also stored as an attribute of the connection class. The method `getConnectVal` returns the value of a connection depending on the type of the connection, for instance, if the type of the connection is "NP-NP", which corresponds to the connection between two neuronal populations, `getConnectVal` returns the neurotransmitter concentration of the source neuronal population multiplied by the weight. In the case of an "HSD-NP" connection, `getConnectVal` returns the variable h of the source compartment, which has to be a `Homeostatic Sleep Drive` object.

3.1.2.2 Storage and use of connections

Connection objects are stored in a list in the target compartment of the connection. In this way, the compartment objects can access the `connectVal` method of the connection in order to be used in the equations. For instance, the expression $\sum_i g_{i.x} C_i$ is implemented in the method `getI` : this method returns the sum of all of the values returned by the `connectVal` method for each connection of type NP-NP in the neuronal population. The `getBeta` method of the neuronal population class and the method `getSourceFR` of the `Homeostatic Sleep Drive` class fulfill the same purpose for the other types of connections. A summary of the functionality of the connections is depicted in Figure 3.2.

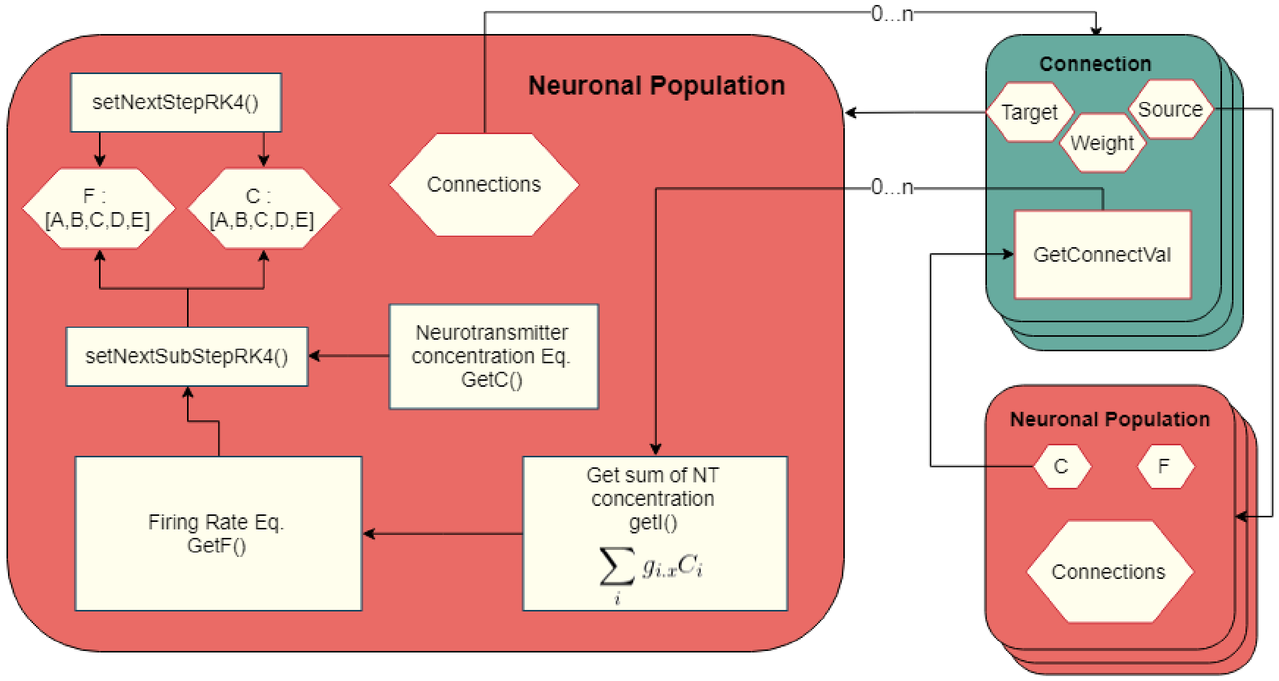


Figure 3.2: Functioning of NP-NP connections and equations resolution with the RK4 method (Rounded rectangle: object; rectangles: methods; hexagons: attributes. Names followed by parenthesis are method names).

3.1.3 Injections and lesions

3.1.3.1 Lesions

Simulating a brain lesion between two neuronal populations would be equivalent to reducing the weight of a connection to zero. As the concentration of the pre-synaptic neurotransmitter is multiplied by the weight of the connection in the firing rate equation, setting this weight to zero will neglect the corresponding connection. Hence, through the parameters modification window, it is possible to set these connection weights to the desired value.

3.1.3.2 Injections

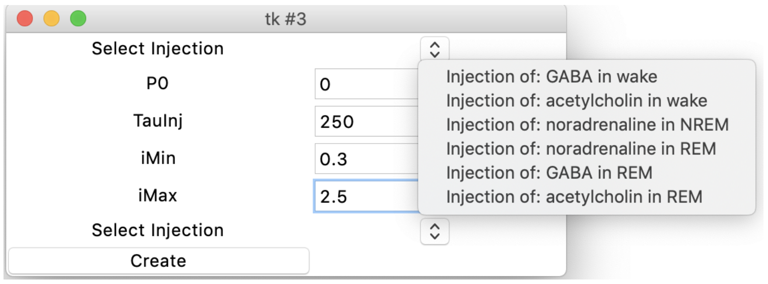
The equations determining the effects of the injections act on the connections results. As the simulated injections are bolus, the effect of the injection fluctuates over time. In order to simulate such an effect, a class `Injection` has been designed and behaves in the same way as a compartment type class.

Injection class The injection objects possesses "next step" methods which are called each step of the simulation to update the injection's variables, like the neuronal population objects. The injection equations are implemented in the injection class in `getMi` and `getP`.

Insertion of injection in the network Given the fact that injections imply a modification of a "NP-NP" connection value, the connection class has been modified to heed injections. Attributes of the connection class are able to store an injection object. New types of connections, namely

"NP-MII-NP" and "NP-MIE-NP," were defined. These connection types simulate an antagonist or agonist injection, respectively. Simulating agonist and antagonist effect with the same injection object is achieved by means of the `getConnectVal` connections method. This method applies different calculation depending on the type of injection.

Injection creation window An injection creation window has been added to the graphical interface. A seen in Figure 3.3, this window sets the location of the injection and the type of neurotransmitter, as well as the type of the injection whether agonist or antagonist. The constants of the injection parameters are also set in this window.


tk #3
Select Injection
Injection of: GABA in wake
Injection of: acetylcholin in wake
Injection of: noradrenaline in NREM
Injection of: noradrenaline in REM
Injection of: GABA in REM
Injection of: acetylcholin in REM
Select Injection
Create

P0	0
TauInj	250
iMin	0.3
iMax	2.5

Figure 3.3: Injection creation window.

3.1.4 Noise

Within the Network class, a method generating additive white Gaussian noise with the help of `NumPy` tools is defined as `additiveWhiteGaussianNoise()`. The default mean and standard deviation values are extracted from the default parameters, but can be changed using the graphical interface in the "Run" tab.

From the `nextStepRK4()` method, the noise generating method is called, and the resulting frequency (in Hz) is stored in a variable. As mentioned in 3.1.1.2, the noise then is passed as an argument in `setNextSubStepRK4()` if the instance is one of a neuronal population and not of homeostatic sleep drive.

3.2 Inputs

The script `manage_parameters.py` enables the user to create a .txt file containing all of the parameters. The parameters are read through the graphical interface, and can be modified and saved in a new .txt file.

A format has been implemented in order to identify the different types of parameters between the neuronal populations, the cycles, and the simulation. It uses the following rules:

- a block of text corresponding to a neuronal population and the corresponding concentration is delimited by "*". It opens with "* population =" followed by the name of the current neuronal population, and ends with a "*".

- a block of text instantiating a cycle is identified by "+". It begins with "+ cycle =" followed by the name of the cycle, and ends with a "+".

- a block of text defining the simulation's parameters, such as the duration, the Δt and the interval of time between each saved result, is delimited by "#".
- lines corresponding to comments begin with "//" followed by a space.
- each parameter has to be defined using its name followed by a "=" and its value. These three elements have to be separated by spaces, not tabulations.

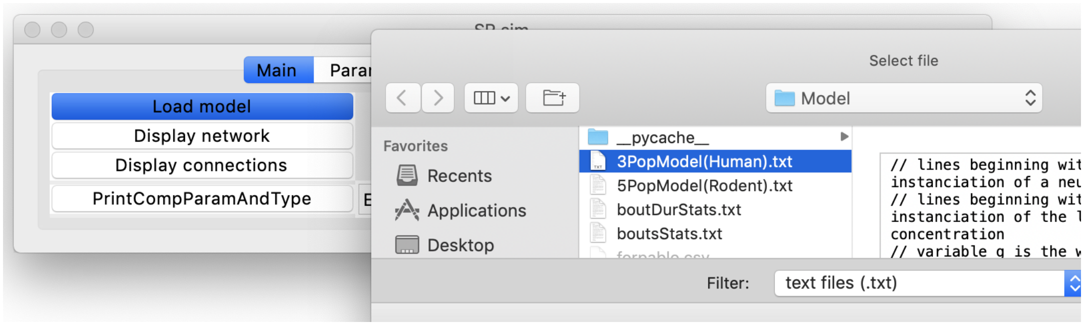
The script saves the modified parameters using the same format, in order to enable the user to reuse those modified parameters simply by using the new .txt file.

The file `manage_parameters.py` uses two functions in order to fulfill its role:

`read_parameters()` is called when the user select the button "Load model" in the graphical interface (as seen on Figure 3.4). It reads each line of the parameters file and using its format in order to create four dictionaries of dictionaries: one containing all of the neuronal populations as a dictionary; one containing all of the cycles in the form of dictionaries; one containing all of the simulation's parameters; and one containing all of the connections between each population and cycle. Each cycle's and neuronal population's dictionary contains all of their respective parameters as values, using the parameter's name as the key. The connections dictionary returns a list of neuronal populations for each population used as keys. The list contains the source populations, where the connections come from towards the target population (used as a key).

`write_parameters()` calls the method `save_parameters()`, present in each of the network's compartments and in its self. This method returns a characters string using the precedent formalism, that `write_parameters()` then writes in an output .txt file. Thanks to the graphical interface, the user chooses when to save the new parameters and under which name they will be saved, by selecting the button "Save Parameters" (as seen on Figure 3.5).

The main script uses the dictionaries produced by `read_parameters` in order to initiate the model thanks to its function `loadModel()`. It instantiates the different objects of the network, using the classes defined in the `SleepRegulation.py` (cf 3.1).

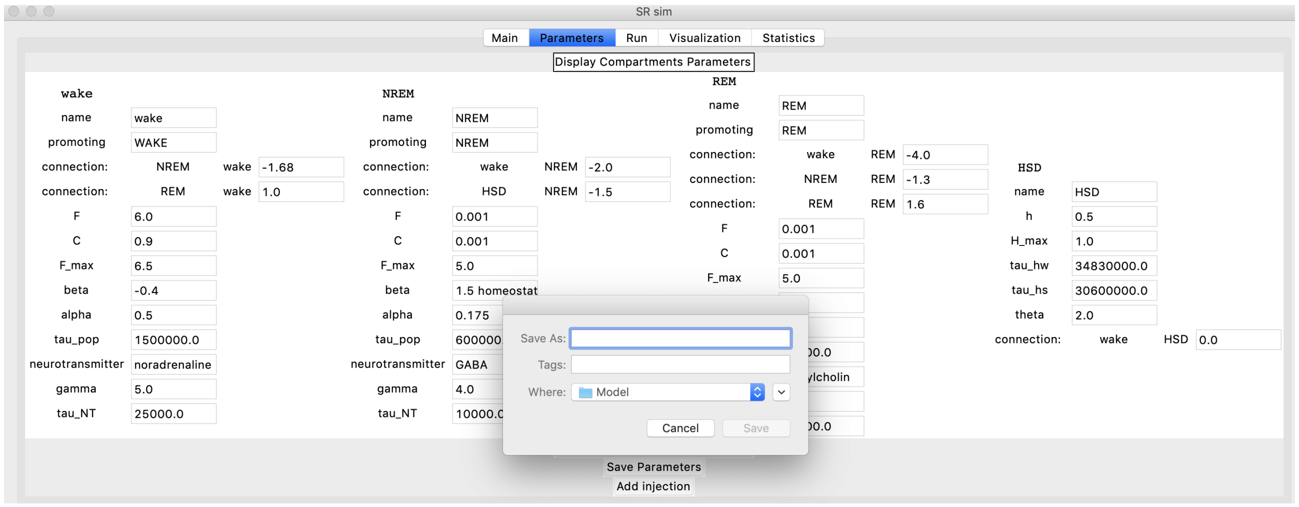


__pycache__
3PopModel(Human).txt
5PopModel(Rodent).txt
boutDurStats.txt
boutsStats.txt
fornable.csv

```
// lines beginning wit
instanciation of a neu
// lines beginning wit
instanciation of the l
concentration
// variable g is the w
```

text files (.txt)

Figure 3.4: Screenshot showing how the user can upload the parameters file of their choice by selecting the "Load model" button.



```
wake
name               wake
promoting          WAKE
connection:        NREM    wake   -1.68
connection:        REM     wake   1.0
F                  6.0
C                  0.9
F_max              6.5
beta               -0.4
alpha              0.5
tau_pop            1500000.0
neurotransmitter   noradrenaline
gamma              5.0
tau_NT             25000.0

NREM
name               NREM
promoting          NREM
connection:        wake    NREM   -2.0
connection:        HSD     NREM   -1.5
F                  0.001
C                  0.001
F_max              5.0
beta               1.5 homeostat
alpha              0.175
tau_pop            600000
neurotransmitter   GABA
gamma              4.0
tau_NT             10000.0

REM
name               REM
promoting          REM
connection:        wake    REM    -4.0
connection:        NREM    REM    -1.3
connection:        REM     REM    1.6
F                  0.001
C                  0.001
F_max              5.0
                   00.0
                   ylcholin
                   00.0

HSD
name               HSD
h                  0.5
H_max              1.0
tau_hw             34830000.0
tau_hs             30600000.0
theta              2.0
connection:        wake    HSD    0.0

Save As:
Tags:
Where:   Model
```

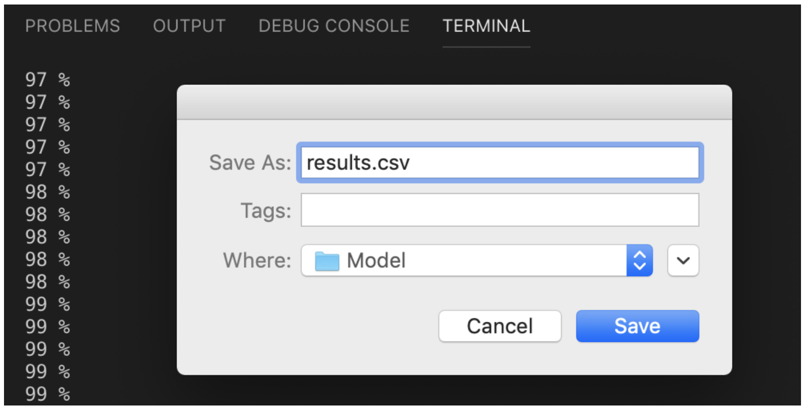
Figure 3.5: Screenshot showing how the user can save the new parameters under a new .txt file by selecting the "Save parameters" button.

3.3 Outputs

3.3.1 Saving the results

Once the simulation is done, the results obtained are saved using the `writeInFile()` method from the class Network and the `cvs` library. It writes the results in a CSV file, separated by tabulations. Each column corresponds to a variable. By default, it saves the time, the hypnogram values, the firing rates and concentrations of each neuronal population and the homeostatic sleep drive at a given moment.

This method is called by the `getResults()` method from the class `NetworkGUI` after launching the simulation. It asks the user under which name it should save the results as once the simulation is over (cf 3.6).



```
97 %
97 %
97 %
97 %
97 %
98 %
98 %
98 %
98 %
98 %
99 %
99 %
99 %
99 %
99 %
```

```
Save As:  results.csv
Tags:
Where:    Model
```

Figure 3.6: Screenshot showing how the user can save the results under a new .CSV file once the simulation is over, meaning once it passes 99% as displayed in the terminal.

Once saved, the results can be treated.

3.3.2 Statistical analysis of the results

When noise is introduced into a simulation, the user may choose to run the simulation multiple times and to analyze the collective hypnogram results.

The hypnogram results can be statistically analyzed using an R script in order to assess their significance. Once obtained, the statistical analyses are represented graphically using the `ggplot2` library, as demonstrated in Figure 3.7.

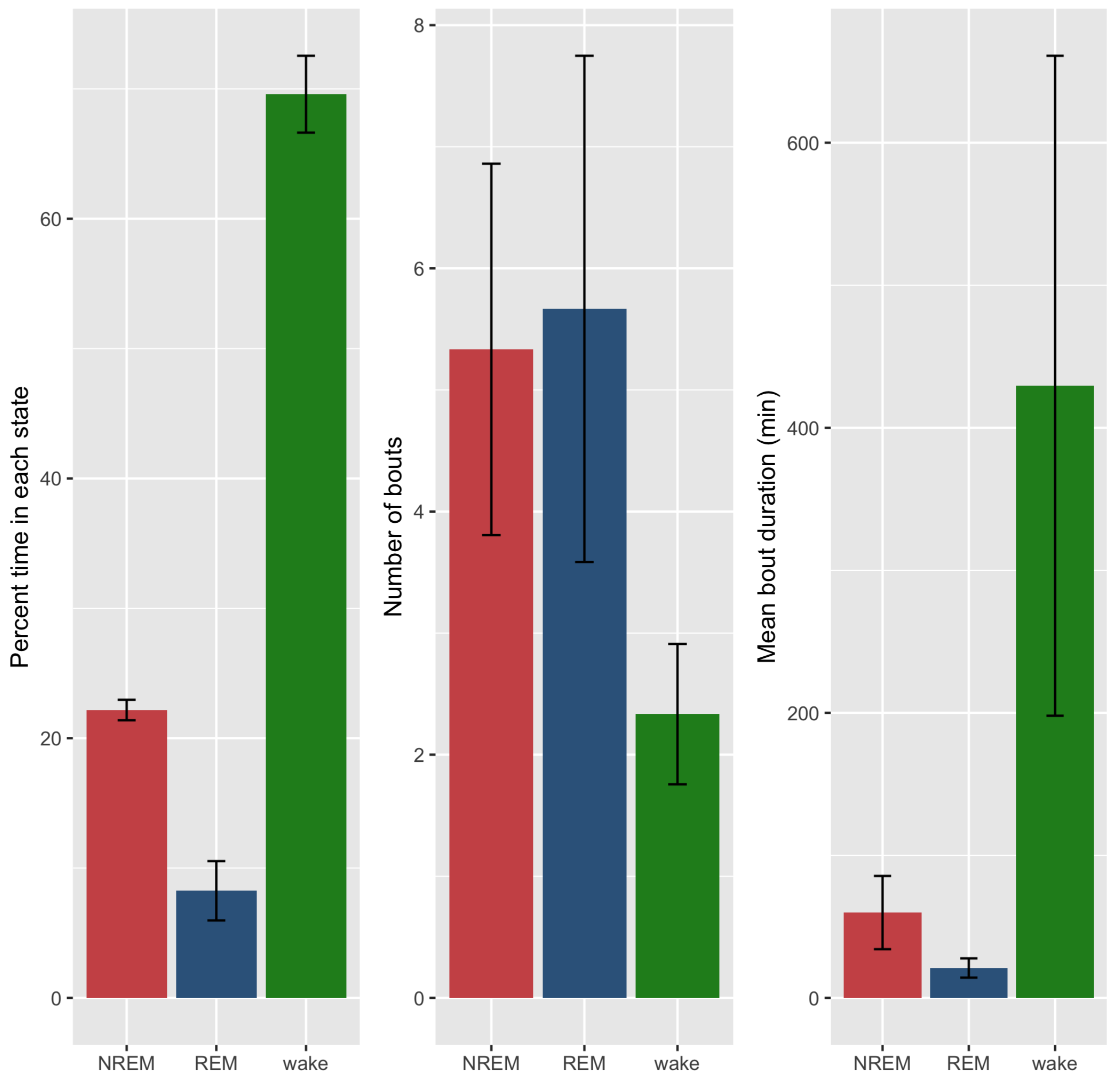


Figure 3.7: Example of statistics graphic saved to the directory containing the model as "plotStats.png." Bars represent mean values from all input data with standard deviation. In this example, n = 3 simulations.

In the `Main.py` script, the function `doStats()` is called within the graphical user interface. The user is prompted to select multiple results files (all saved in the same directory) to analyze. This function uses the OS module to call the R script, passing the name of the selected results files as arguments.

Within the R script, the hypnogram results of each file are extracted and added to a list of lists. Each element of this list corresponds to the hypnogram results of one file. The list is passed through an algorithm that counts the number of occurrences in each state (wake, REM, NREM), the number of bouts in each state, and the length of the different bouts for each file.

Once dataframes with these results are constructed, .txt files are produced with statistical analyses (one-way ANOVA and Tukey HSD Post-Hoc) as well as three bar graphs in a .png file. The bar graphs depict the mean percent time spent in each state, the mean number of bouts in each state, and the mean duration of bouts by state in minutes (Figure 3.7). Each graph has a corresponding statistical analysis .txt file.

3.3.3 Visualization of results

The results can be visualized in the form of a graphic depicting the firing rates and neurotransmitter concentrations evolution over time, as well as a hypnogram. According to the user's choice, it can be produced directly from the simulation's results or from precedent results saved in a CSV file. The mean of multiple results can also be represented on a graph, with or without the standard deviation. The user makes their choice through the graphical interface (as seen in Figure 3.8).

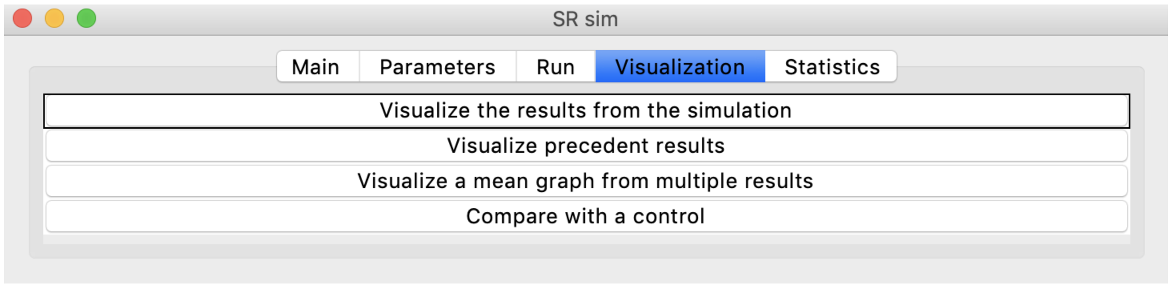

SR sim
Main
Parameters
Run
Visualization
Statistics
Visualize the results from the simulation
Visualize precedent results
Visualize a mean graph from multiple results
Compare with a control

Figure 3.8: Screenshot showing how the user can choose to display a graph either from the simulation (by selecting the "Visualize the results from the simulation" button) or from precedent results (by selecting "Visualize precedent results). Additionally, the user can choose to display a graph that considers the mean of multiple results files ("Visualize a mean graph from multiple results") or compare a result to a control ("Compare with a control")

The graphics are produced by the script `graphic.py` using `Matplotlib` and particularly the associated package `PyLab`.

One main function `createGraph()` reads a dictionary of lists containing the different results: each variable saved (time, homeostatic sleep drive, hypnogram, etc.) is the key to the corresponding list of values. This dictionary is translated by the function `transformData()`: it returns another dictionary with keys such as "firing rates", "time", "concentrations", "hypnogram", "standard deviation", etc; this way, the script is much more flexible and can handle any type of model, with any neuronal populations and cycles. `createGraph()` also uses a dictionary containing the different neuronal populations of the current model and the corresponding neurotransmitters, in order to use them as labels on the "Concentrations" graph (as seen on figure 3.10). This dictionary is obtained thanks to the function `getNeurotransmitters()`, which translates a string given by the method `fileHeader()` in the `Network` class.

The main function then asks the user which of the variables they want to display thanks to the graphical interface created by `whatToDisplay()` (see Figure 3.9), then creates on, two or three stacked graphics:

- one with the evolution of the firing rates of each population over time
- one with the evolution of the neurotransmitter concentrations and of the homeostatic sleep drive over time
- one with the hypnogram, varying between 0 (NREM state), 0.5 (REM state) and 1 (wake state). The previous numbers have been replaced on the y axis by the name of the corresponding state.

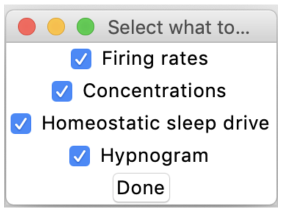


Figure 3.9: Screenshot showing how the user can choose what to visualize within the firing rates and concentrations graphs.

The file `graphic.py` also contains four other functions, which are used depending on the user's choice.

If the user wants to display a graphic directly from the simulation results, they must select the button "Visualize the results from the simulation" (cf 3.8), which then calls the function `GraphFromSim()`. This function translates the results from the simulation into a dictionary readable by calls `createGraph()`.

If the user would rather display a graphic from precedent results, they have to select then the button "Visualize precedent results" (cf 3.8), which then calls the function `GraphFromCSV()`. This function translates the data from the CSV file into a readable dictionary through the function `readCSV()`, also implemented in `graphic.py`, then calls `createGraph()`.

The button "Visualize a mean graph from multiple results" is useful if the user wishes to display the graph corresponding to the mean of multiple results and the corresponding hypnogram. The associated function `createMeanGraph()` computes the mean of each value for every variables contained in the selected files as well as the corresponding standard deviation, thanks to the library `statistics`, then calls the main function `createGraph()`. The user can choose whether to display the standard deviation, the standard error of the mean or nothing using the interface graphic as seen on figure 3.11 ; for example, the standard error of the mean appears as seen on the graph 3.12.

Finally, if the user wishes to compare two sets of results, using one as control, they can select the button "Compare with a control", which calls the function `compareWithControl()`. The user is asked whether he wants to use one result or the mean of multiple files as control, and similarly for the data to be compared to the control. `compareWithControl()` uses then either `graphFromCVS()` or `createMeanGraph()` to create two superimposed graphs. When used to display the control graph, the second function is given the option "control" as argument : this argument is used by the main function `createGraph()` to draw the corresponding graph in dotted and transparent lines. This function can be used to compare the effect of microinjections or lesions to a control, or the mean of multiple results obtained with the Euler method to those obtained with RK4. This last example can be seen on figure 3.13.

Using the `MatPlotLib` library, the graphic produced can be saved under the image type of the user's choice by selecting the save icon under the graph (cf 3.10).

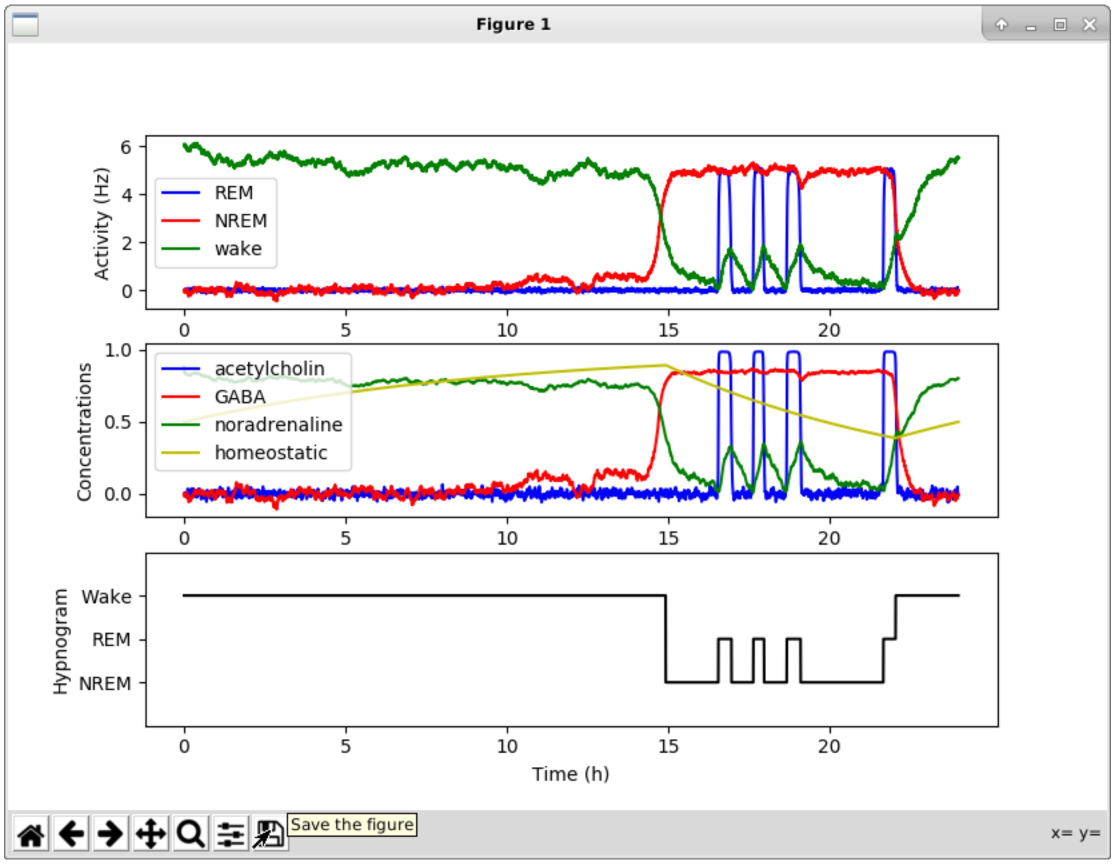
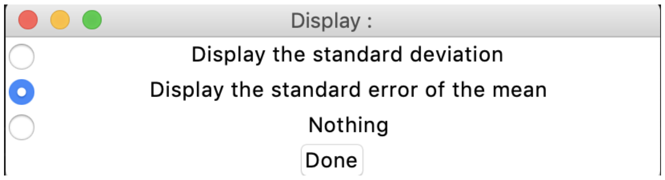


Figure 3.10: Screenshot showing how the user can save the graphics with the save icon. The displayed graph corresponds to what can be obtained after a simulation of 24 hours using RK4, with added noise and without injections or lesions.


Display :

( ) Display the standard deviation
(●) Display the standard error of the mean
( ) Nothing

Done

Figure 3.11: Screenshot showing how the user can choose between standard deviation and standard error in the firing rates and concentrations graphs.

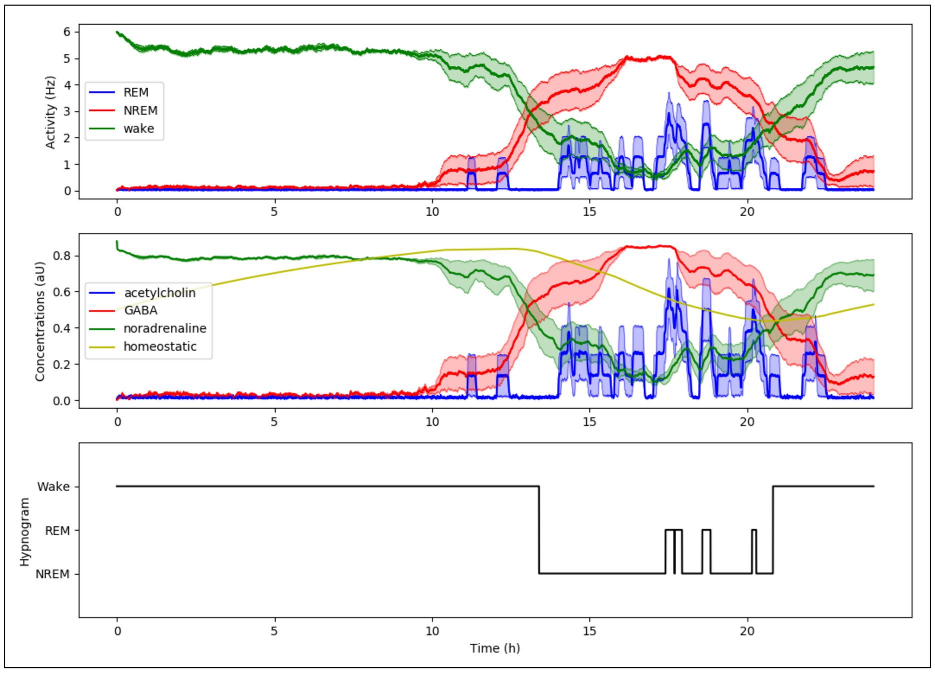


Figure 3.12: Graph representing the mean of 8 different results obtained over 24h using RK4, without lesion or microinjection, and showing the standard error of the mean in transparent.

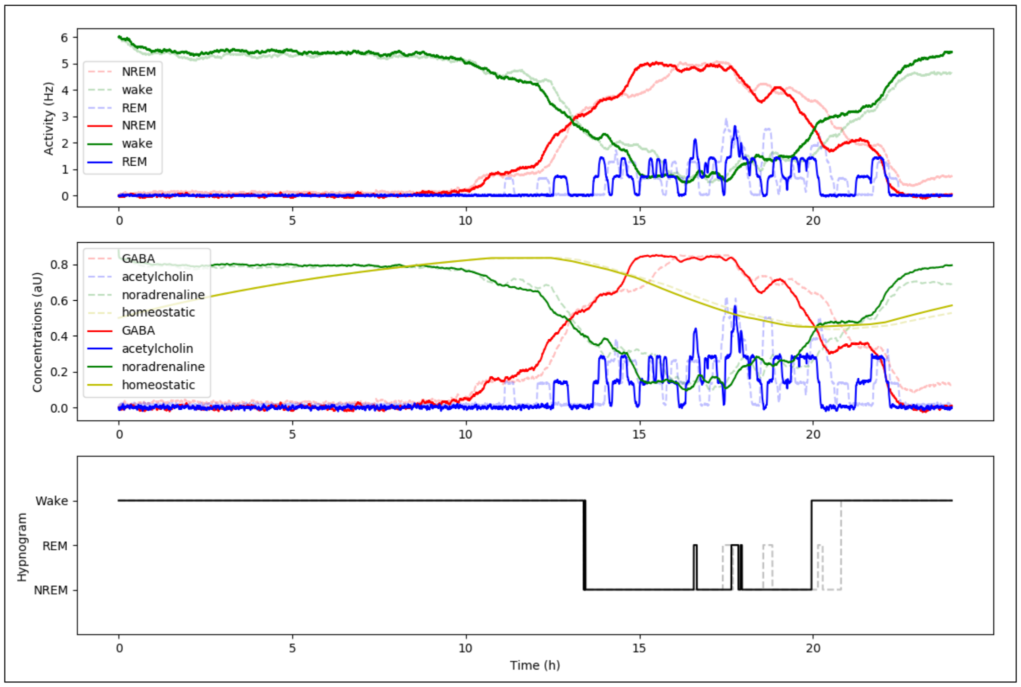


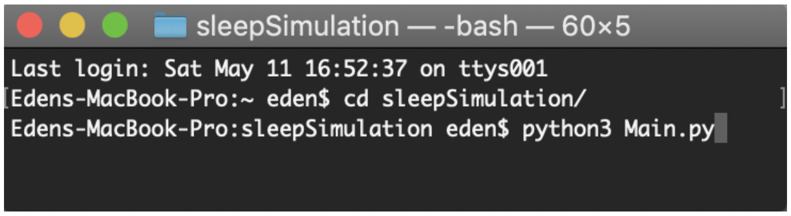
Figure 3.13: Graph comparing the mean of 8 results obtained using RK4 (solid lines) to the mean of 8 results obtained using Euler (light and dotted lines). Both sets of results have been obtained over 24h, without lesion or microinjection.

3.4 Accessibility of the model

3.4.1 Launching the program

From a "Main" Python script, functions are appropriately called by the system to insert parameters, run the simulation, to view the graphs generated, and to perform statistical analyses.

The user only has to use one line of code (the one that can be seen on 3.14), and all the rest can be done by selecting different buttons or entering text in the graphical interface.

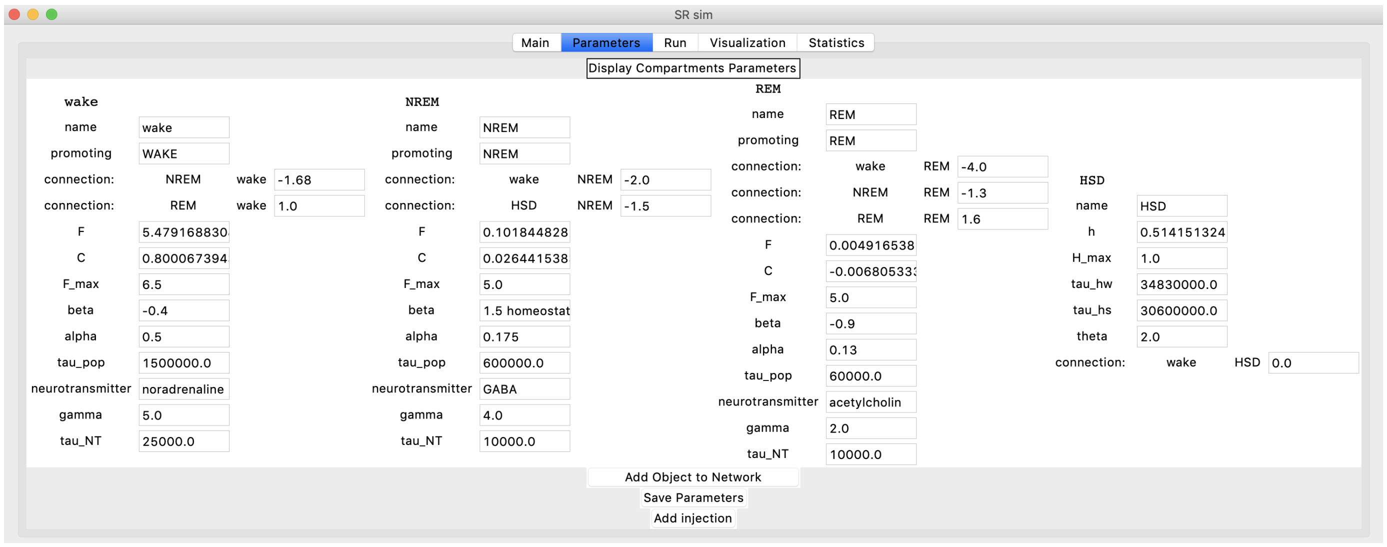


```
Last login: Sat May 11 16:52:37 on ttys001
Edens-MacBook-Pro:~ eden$ cd sleepSimulation/
Edens-MacBook-Pro:sleepSimulation eden$ python3 Main.py
```

Figure 3.14: Screenshot of the command to execute in order to launch the model from the terminal.

3.4.2 Modifying the parameters

The different parameters are easily modifiable by the user using the graphical interface. On the "Parameters" tab, the different parameters of the current model are displayed (as seen on 3.15) from the file parameters that was initially loaded. The user can directly type in the new parameter at the correct location, and it is automatically taken into account by the network. This means that the parameters are automatically modified and that the user doesn't have to click on a "Use new parameters" button.



wake

name	wake		
promoting	WAKE		
connection:	NREM	wake	-1.68
connection:	REM	wake	1.0
F	5.479168830		
C	0.800067394		
F_max	6.5		
beta	-0.4		
alpha	0.5		
tau_pop	1500000.0		
neurotransmitter	noradrenaline		
gamma	5.0		
tau_NT	25000.0		

NREM

name	NREM		
promoting	NREM		
connection:	wake	NREM	-2.0
connection:	HSD	NREM	-1.5
F	0.101844828		
C	0.026441538		
F_max	5.0		
beta	1.5 homeostat		
alpha	0.175		
tau_pop	600000.0		
neurotransmitter	GABA		
gamma	4.0		
tau_NT	10000.0		

REM

name	REM		
promoting	REM		
connection:	wake	REM	-4.0
connection:	NREM	REM	-1.3
connection:	REM	REM	1.6
F	0.004916538		
C	-0.00680533		
F_max	5.0		
beta	-0.9		
alpha	0.13		
tau_pop	60000.0		
neurotransmitter	acetylcholin		
gamma	2.0		
tau_NT	10000.0		

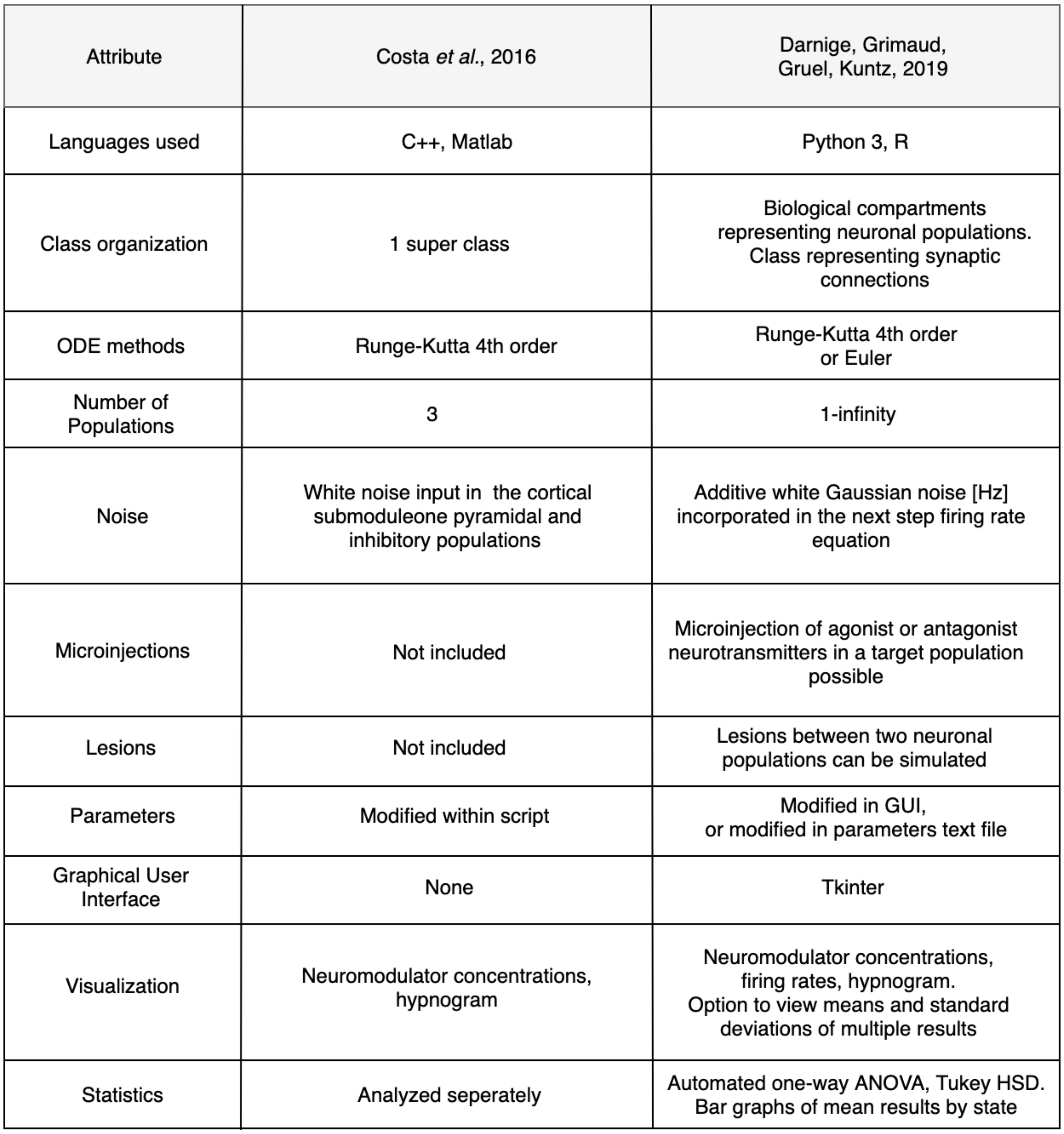
HSD

name	HSD		
h	0.514151324		
H_max	1.0		
tau_hw	34830000.0		
tau_hs	30600000.0		
theta	2.0		
connection:	wake	HSD	0.0

Figure 3.15: Screenshot showing how the user can easily modify the parameters.

3.5 Model comparison

A summary of the differences between the reference Costa *et al.* model and the model developed in this project is illustrated in Figure 3.16.



Attribute	Costa *et al.*, 2016	Darnige, Grimaud, Gruel, Kuntz, 2019
Languages used	C++, Matlab	Python 3, R
Class organization	1 super class	Biological compartments representing neuronal populations. Class representing synaptic connections
ODE methods	Runge-Kutta 4th order	Runge-Kutta 4th order or Euler
Number of Populations	3	1-infinity
Noise	White noise input in the cortical submoduleone pyramidal and inhibitory populations	Additive white Gaussian noise [Hz] incorporated in the next step firing rate equation
Microinjections	Not included	Microinjection of agonist or antagonist neurotransmitters in a target population possible
Lesions	Not included	Lesions between two neuronal populations can be simulated
Parameters	Modified within script	Modified in GUI, or modified in parameters text file
Graphical User Interface	None	Tkinter
Visualization	Neuromodulator concentrations, hypnogram	Neuromodulator concentrations, firing rates, hypnogram. Option to view means and standard deviations of multiple results
Statistics	Analyzed seperately	Automated one-way ANOVA, Tukey HSD. Bar graphs of mean results by state

Figure 3.16: Comparison of Costa *et al.*, 2016 and the model developped in this project.

While the example of a three population model is presented throughout this document, the model is not limited to three populations as shown in Figure 3.16. For example, the results for a five population model can be seen in the Appendix.

3.6 Further development

3.6.1 Circadian regulation

With the sleep-wake cycle being closely related to the circadian clock, the inclusion of dark and light periods might make the model more realistic. To model these periods, the suprachiasmatic nucleus (SCN) and his properties can be used. Indeed, the SCN, situated in the hypothalamus, interacts with many regions of the brain, and thus regulates many physiological processes, including sleep-wake behaviour in a 24h cycle.

Fleshner and colleagues have described three different models of interactions between the SCN and the sleep-wake regulatory network [13]. These three models can theoretically be added to the five model populations by adding the SCN as three different `Neuronal Population`, each defined by their own parameters. However, due to a lack of time and the fact that it was not a priority task asked by our client, there was no testing done on this.

3.6.2 Neural mass model of the sleeping cortex

In the initial code of Costa *et al* [1], the `Sleep_Regulation` class is not the only class. There is a second class concerning the cortex, named `Cortical_Column`, based on the membrane voltage of the excitatory population and on the sodium-dependent potassium current.

Adding the `Cortical_Column` class would allow for the creation of bifurcation diagrams and electroencephalograms (EEG) where the hypnograms and graphs depicting the firings rates and neurotransmitter concentrations evolution over time are generated in this model. The different states of sleep could be determined by the identification of K-Complexes and slow cortical oscillations in the EEG [14].

The `Cortical_Column` class cannot be implemented in the current model as a `NeuronalPopulation` instantiation because none of the parameters are shared. This means that the creation of a new class is necessary, and thus an adaptation of the whole code. This could be done fairly easily because it was already present in the C++ and Matlab code, but it was not a priority from our client.

3.6.3 Known issues

Due to the structure of the code, the units of the parameters were not added to the Parameters tab within the GUI. The user can refer to the user manual for a table containing the parameter units.

Occasionally, a "TypeError" occurs when declaring the parameters that create a lesion. In this case, the user may need to relaunch the program.

Conclusion

Dr. Héricé and the Sakata team continue to develop upon Dr. Costa's sleep cycle model published in 2016 [1]. Computational models representing the oscillations between wake, REM and NREM sleep can facilitate and reduce the amount of labor required to study sleep cycles as compared to using rodent optogenetics. In this document, the steps made in the development of this computational model of paradoxical sleep have been described.

Many improvements upon the reference Costa *et al.* model have been made in terms of flexibility and functionality. These improvements are emphasized in Figure 3.16. Notably, the inclusion of a graphical interface increases the accessibility of the program in order to be used by a wider audience of researchers studying sleep cycles.

Model flexibility has been enhanced by introducing modifiable parameters and a number of simulation options. In addition, the opportunity to microinject agonist or antagonist neurotransmitters in a target population is presented, as well as simulating lesions between two neuronal populations.

Visualizing the results can be customized through data selection. The firing rates, neuromodulator concentrations, and hypnogram of sleep states are generated. There is even an option to display a mean graph of multiple results, with or without the standard deviation, for the firing rate and concentration graphs.

Statistical analyses have been automated to treat multiple results files from a simulation containing noise. An output graph of the mean percent time spent in each state, the mean number of bouts in each state, and the mean duration of bouts by state in minutes is determined from hypnogram results. The R script is also accessible separate from the model, allowing researchers to further customize their analysis in terms of statistical tests and graphical representations.

Although this model is designed to meet the needs of a lab that studies sleep in rodents, the model is not limited to rodent data. Indeed, human parameters can be used to simulate neuromodulator concentrations and sleep cycle phases.

Acknowledgements

We would like to express our deepest appreciation to Dr. Charlotte Héricé for giving us the opportunity to work on this exciting project. Under her supervision, we have acquired essential knowledge in the field of computational modeling. Thank you for taking the time to teach us all the way from Scotland.

A special gratitude we dedicate to Dr. Marie Beurton-Aimar for guiding us through the process of developing a professional project.

Finally, we thank Dr. André Garenne for sharing his wisdom. His advice has been a great help in structuring the code and understanding the mathematical equations that drive the model.

Bibliography

[1] Michael Schellenberger Costa, Jan Born, Jens Christian Claussen, and Thomas Martinetz. Modeling the effect of sleep regulation on a neural mass model. *Journal of Computational Neuroscience*, 41:15–28, August 2016.

[2] Cecilia G. Diniz Behn and Victoria Booth. A population network model of neuronal and neurotransmitter interactions regulating sleep–wake behavior in rodent species. *Sleep and Anesthesia*, pages 119–138, 2011.

[3] SIPBS. What is sipbs ?, 2019. https://www.strath.ac.uk/science/strathclydeinstitute ofpharmacybiomedicalsciences/whatissipbs/ [Accessed: 07/03/19].

[4] Shuzo Sakata. Dr shuzo sakata, university of strathclyde, 2019. https://www.strath.ac.uk/staff/sakatashuzodr/ [Accessed: 07/03/19].

[5] Robert W. McCarley. Neurobiology of REM and NREM sleep. *Sleep Medicine*, 8(4):302–330, jun 2007.

[6] Vincenzo Crunelli and Stuart W Hughes. The slow (less than 1 hz) rhythm of non-rem sleep: a dialogue between three cardinal oscillators. *Nature Neuroscience*, 13(1):9–17, dec 2009.

[7] J. Hobson, R. McCarley, and P. Wyzinski. Sleep cycle oscillation: reciprocal discharge by two brainstem neuronal groups. *Science*, 189(4196):55–58, jul 1975.

[8] Reidun Ursin. Serotonin and sleep. *Sleep Medicine Reviews*, 6(1):55–67, feb 2002.

[9] Clifford B. Saper, Thomas E. Scammell, and Jun Lu. Hypothalamic regulation of sleep and circadian rhythms. *Nature*, 437(7063):1257–1263, oct 2005.

[10] Cecilia G. Diniz Behn and Victoria Booth. Simulating microinjection experiments in a novel model of the rat sleep-wake regulatory network. *Journal of Neurophysiology*, 103:1937–1953, January 2010.

[11] A.A. Borbely. A two process model of sleep regulation. *Human Neurobiology*, pages 195–204, 1982.

[12] Cecilia G. Diniz Behn and Victoria Booth. A fast-slow analysis of the dynamics of REM sleep. *SIAM Journal on Applied Dynamical Systems*, 11(1):212–242, jan 2012.

[13] M. Fleshner, V. Booth, D. B. Forger, and C. G. Diniz Behn. Circadian regulation of sleep-wake behaviour in nocturnal rats requires multiple signals from suprachiasmatic nucleus. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1952):3855–3883, sep 2011.

[14] Arne Weigenand, Michael Schellenberger Costa, Hong-Viet Victor Ngo, Jens Christian Claussen, and Thomas Martinetz. Characterization of k-complexes and slow wave activity in a neural mass model. *PLoS Computational Biology*, 10(11):e1003923, nov 2014.

Appendix

Five population model

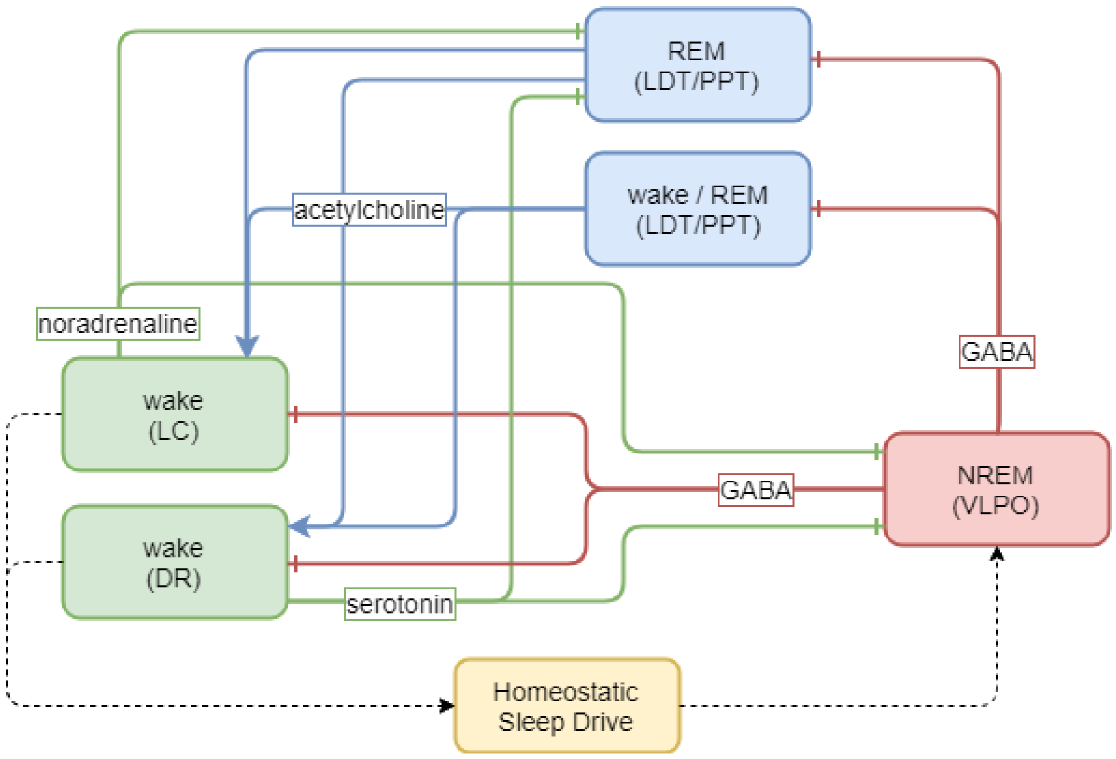


Figure 1: Diagram representing the connectivity within the 5 population model. Pointed lines indicate connectivity between homeostatic sleep drive and neuronal populations. Solid lines with arrows indicate excitatory synaptic inputs between neuronal populations, while solid lines with cross ends indicate inhibitory synaptic input between neuronal populations.

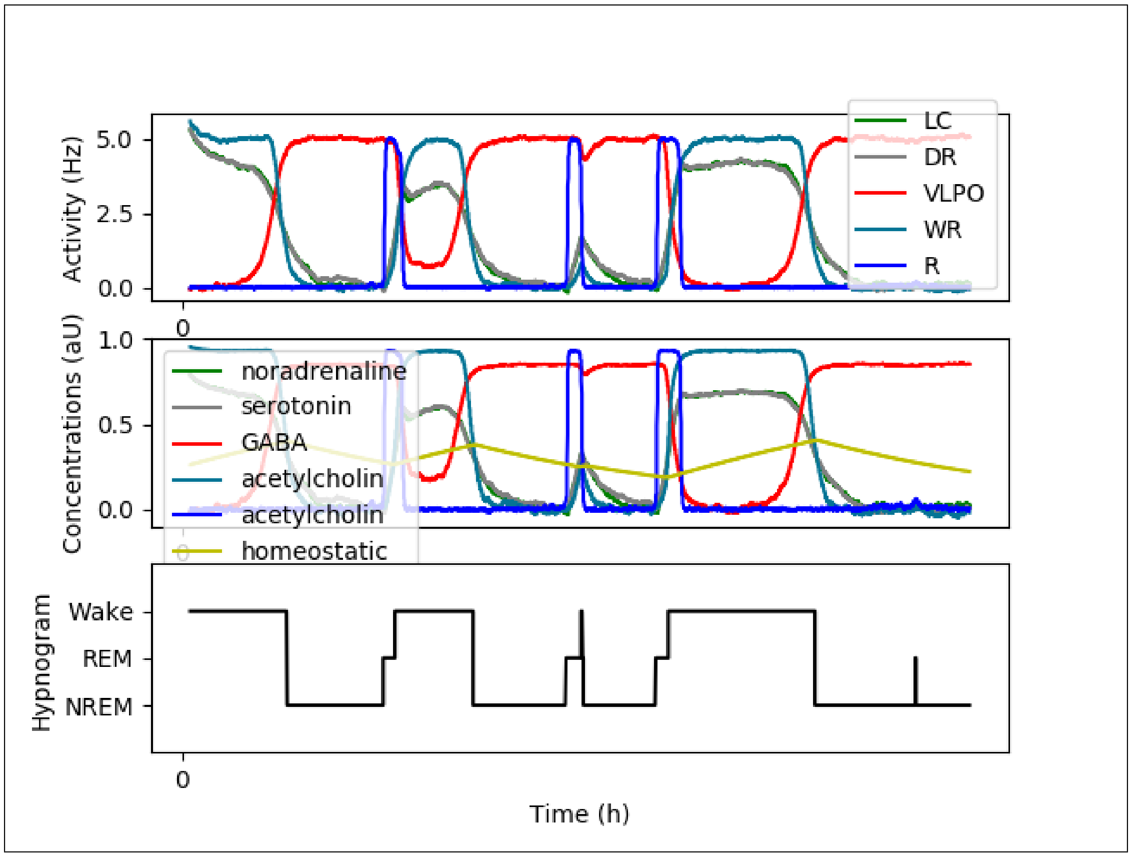


Figure 2: Firing rates, neurotransmitter concentrations and hypnogram generated by the 5 population sleep regulation model in rodents [10]. The graph was generated using Euler's method with a time step of 2ms during 1000s. Gaussian noise with a standard deviation of 0.001 and a mean of 0 was applied to the firing rates. The hypnogram was generated with a Wake threshold of 0.64 and a REM threshold of 0.9.

.

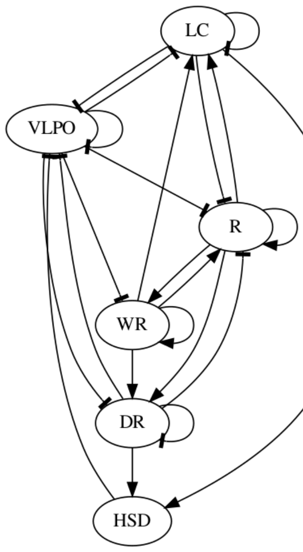


Figure 3: Network of the 5 population sleep regulation model. The graph was automatically generated from the software using the `graphViz` module.