

# Computational Tools for the Simulation and Analysis of Spike Trains

by

JV Afable

Supervisor: Prof. Paul Marriott

A research paper presented to the  
University of Waterloo  
In partial fulfillment of the requirements for the degree of  
Master of Mathematics  
in  
Statistics

Waterloo, Ontario, Canada, 2024

© JV Afable 2024

# Abstract

This paper presents a set of tools and a workflow for replicating and modifying a spiking neural network simulation of the olfactory bulb using NEURON. Key concepts in computational neuroscience are first reviewed, including spike trains, neuron models, network architectures, and the biological circuitry of the olfactory bulb. The process of replicating an existing olfactory bulb simulation study is then described in detail. Modifications to the model are explored, investigating the effects of changing the random seed and adjusting mitral-granule cell network connectivity. Results demonstrate consistent network behavior across seeds, but a strong dependence of mitral and granule cell spiking activity on connectivity between these populations. The computational workflow establishes a framework for replicating and extending published neural simulations.

# Table of Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b> |
| 1.1      | Objectives . . . . .  | 1        |
| 1.2      | Neuron Simulation . . . . .                                   | 2        |
| 1.2.1    | Spike Trains . . . . .  | 2        |
| 1.2.2    | Local Field Potential . . . . .                               | 3        |
| 1.2.3    | Stochasticity . . . . .                                       | 3        |
| 1.2.4    | Encoding Problem . . . . .                                    | 4        |
| 1.2.5    | Modeling Perspectives . . . . .                               | 5        |
| 1.3      | Neuron Modeling . . . . .                                     | 5        |
| 1.3.1    | Leaky-Integrate-and-Fire Model . . . . .                      | 5        |
| 1.3.2    | Hodgkin-Huxley Model . . . . .                                | 6        |
| 1.3.3    | Network Coupling Schemes . . . . .                            | 7        |
| 1.4      | The Olfactory Bulb . . . . .                                  | 9        |
| 1.4.1    | Neural Circuitry and Oscillatory Dynamics . . . . .           | 9        |
| 1.4.2    | Network Architectures Simulating the Olfactory Bulb . . . . . | 11       |
| 1.5      | Olfactory Bulb Simulation Study . . . . .                     | 12       |
| 1.5.1    | Key Findings . . . . .  | 12       |
| 1.5.2    | Computational Details . . . . .                               | 13       |

|          |  |           |
|----------|--|-----------|
| <b>2</b> | <b>Simulation Replication</b>              | <b>16</b> |
| 2.1      | Downloading the Model . . . . .            | 17        |
| 2.2      | Running the Simulation in Python . . . . . | 19        |
| 2.3      | Reproducing Plots in R . . . . .           | 20        |
| 2.3.1    | Figure 2A . . . . .                        | 20        |
| 2.3.2    | Figure 2B . . . . .                        | 22        |
| 2.3.3    | Figure 2C . . . . .                        | 23        |
| 2.3.4    | Figures 2D1, 2E1, 2F1 . . . . .            | 24        |
| 2.3.5    | Figures 2D2, 2E2, 2F2 . . . . .            | 26        |
| 2.3.6    | Figures 2D3, 2E3, 2F3 . . . . .            | 28        |
| <b>3</b> | <b>Simulation Modifications</b>            | <b>30</b> |
| 3.1      | Random Seeds . . . . .                     | 30        |
| 3.2      | Sparse and Full Connectivity . . . . .     | 33        |
| <b>4</b> | <b>Conclusion</b>                          | <b>38</b> |
|          | <b>Bibliography</b>                        | <b>39</b> |
|          | <b>Appendix</b>                            | <b>42</b> |
| <b>A</b> | <b>Running Neuron Simulation in Python</b> | <b>43</b> |
| <b>B</b> | <b>R Codes for Plot Reproduction</b>       | <b>52</b> |
| B.1      | Figure 2A . . . . .                        | 52        |
| B.2      | Figure 2B . . . . .                        | 53        |
| B.3      | Figure 2C . . . . .                        | 55        |
| B.4      | Figures 2D1, 2E1, 2F1 . . . . .            | 56        |
| B.5      | Figures 2D2, 2E2, 2F2 . . . . .            | 57        |
| B.6      | Figures 2D3, 2E3, 2F3 . . . . .            | 58        |

# Chapter 1

## Introduction

### 1.1 Objectives

This paper aims to explore and document a workflow for replicating and extending simulations created for NEURON (NEURON 2024), a popular simulation environment for modeling networks of neurons, by interfacing simulations through Python, a popular high-level programming language, and performing statistical analyses through R, a programming language for statistical computing and data analysis. We will demonstrate this workflow by replicating a paper with biologically plausible neuron simulations in order to generate a rich set of examples that can be used to validate new statistical methodologies. In particular, we will be attempting to partially replicate and analyze the findings from a simulation study (Li and Cleland 2017) which will serve as a concrete demonstration of our workflow. Overall, this paper hopes to provide a clear and replicable set of processes for performing and extending similar types of studies and simulations that future students and researchers may use for their neuron simulation endeavors.

The remainder of Chapter 1 will provide an overview on a number of topics. First, a brief review of essential concepts underlying neuron simulation along with basic neuron models and network coupling schemes. Afterwards, the structure of the olfactory bulb and its modeling will be discussed in order to provide context for the selected simulation study (Li and Cleland 2017). Lastly, a brief overview of key findings from the simulation study (Li and Cleland 2017) along with the key computational details will be provided.

## 1.2 Neuron Simulation

The following section on neuron simulation will provide a brief review of essential concepts underlying neuron simulation along with basic neuron models and network coupling schemes. This section will assume that the reader is already familiar with the presented concepts. It is intended to be a high-level overview.

### 1.2.1 Spike Trains

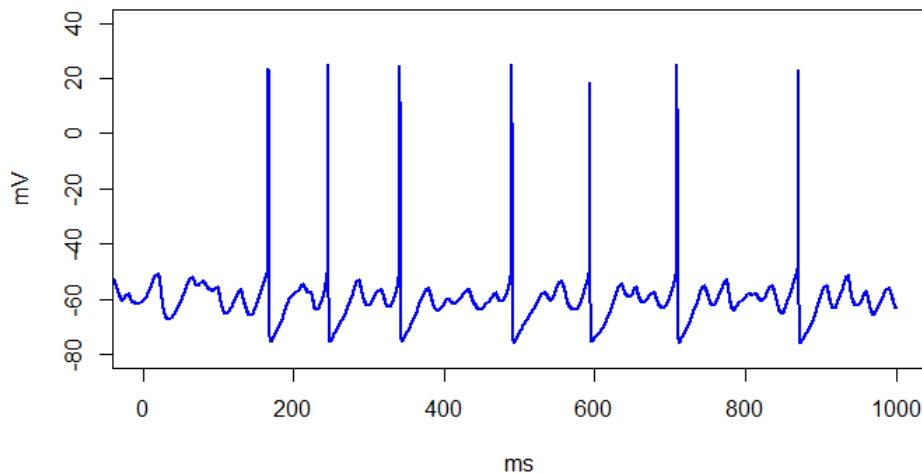


Figure 1.1: Example spike train showing multiple action potentials (spikes) over 1 second, with subthreshold membrane potential fluctuations between spikes. Spikes exhibit an all-or-nothing response rising above around 20 mV, with minor variations in peak voltage between individual spikes.

In Computational Neuroscience, a fundamental area of research is focused on the modeling of neuron action potentials (APs), otherwise known as spikes, with sequences of spikes referred to as spike trains (Kass et al. 2018). Physically, spikes are generated by the movement of ions against concentration gradients across a neuron’s membrane through ion pumps. This creates a difference in electrical potential between the neuron and the extracellular environment (Dayan and Abbott 2001). When a neuron’s membrane electrical potential rises past a threshold, a spike occurs (Dayan and Abbott 2001). Neuron

spikes can be thought of as identical events, where a spike either occurs or does not occur, exhibiting an *all-or-nothing* behavior (Katz 1966).

Figure 1.1 depicts a spike train with multiple spikes occurring within 1000 milliseconds or 1 second. Spikes can be seen rising past some threshold between  $-20$  mV to  $20$  mV. Although each spike is considered to be identical, note that there are minor differences in peak electrical potential between each spike. Subthreshold potentials are also visible, but since they do not rise past the threshold, they are not considered spikes. This illustrates the *all-or-nothing* behavior mentioned previously.

### 1.2.2 Local Field Potential

Local Field Potentials (LFPs) arise from the activity of a network of neurons (Kass et al. 2018). The LFP contains information on synaptic processes that cannot be observed from only the activity of a few neurons (Einevoll et al. 2013). Millions of neurons at a time may contribute to an LFP which is typically observed through a recording electrode (Hagen et al. 2016). These recording electrodes capture the synaptic input of local neuron populations along with those farther from the electrode itself (Hagen et al. 2016). Due to this, there is ambiguity in the signal represented in the LFP, making it more difficult to interpret than spikes (Einevoll et al. 2013). This ambiguity may be addressed in computational neuron simulations where all neurons contributing to the LFP may be defined, observed, and measured (Einevoll et al. 2013).

However, it is not clear what measurements in a simulation correspond to the an LFP, and several methods have been suggested to calculate it. Some of these methods include pooling spike trains, taking the average membrane potential, and summing the synaptic input currents onto pyramidal neurons (Einevoll et al. 2013; Buehlmann and Deco 2008). In particular, the study (Li and Cleland 2017) replicated in this paper simply takes the average membrane potential across all mitral cells in the olfactory bulb neuron network. It may be the case that the LFP is a weighted average of the input signals of a neural population (Buehlmann and Deco 2008). More precise calculations of the LFP may need to consider spatial distance as well as the specific biophysical features of neurons in the vicinity of a recording electrode (Hagen et al. 2016; Einevoll et al. 2013).

### 1.2.3 Stochasticity

Spike trains exhibit stochasticity, often varying from trial to trial even when presented with the same stimulus (Dayan and Abbott 2001). Numerous factors contribute to this variability, such as fluctuations in arousal and attention, randomness in biophysical processes

influencing neuron firing, and the impact of additional cognitive activities occurring during an experiment (Dayan and Abbott 2001; Rolls and Deco 2010). Moore et al. (1966), and Calvin and Stevens (1967, 1968) delve into the reasons behind randomness in neuron spike trains, pinpointing both internal factors, like membrane potential fluctuations, and external influences, such as synaptic inputs, as key contributors. Calvin and Stevens (1968) further link this variability to synaptic noise and the internal conversion of synaptic current to action potentials, noting the role of thermal noise and neurotransmitter release randomness.

Additionally, the variability in neural spike trains stems from both the basic firing mechanisms of neurons and the complex dynamics of neural circuits. Spike trains can be seen as stochastic point processes, with spikes occurring randomly and unpredictably (Moore et al. 1966). This randomness means that the pattern of observed spikes is just one of many possible outcomes that could arise under the same conditions. Panzeri et al. (2007) point out the challenges of analyzing such data when it is limited by the finite number of trials in real experiments. These limited experimental data may lead to systematic error (bias) and statistical error (variance) when trying to estimate true stimulus-response probabilities (Panzeri et al. 2007). The variability is also increased by the inherently noisy processes of the brain (Rolls and Deco 2010). Additionally, uncertainty occurs because only a small sample out of the total population of neurons is typically observed in most experiments. Altogether, these emphasize how stochasticity is a fundamental part of how neurons communicate and process information.

It is worth noting that in their review of mathematical and statistical perspectives in computational neuroscience, Kass et al. (2018) mention very little about the sources of stochasticity in neuron network data. There is plenty of discussion on methods to deal with this stochasticity, but there is almost no discussion on why such stochasticity exists in the first place. This may indicate a potential gap in knowledge on the topic.

### 1.2.4 Encoding Problem

Research in the field is often concerned with determining how neurons convey information in their spike firing pattern. This is known as the *encoding problem* (Theunissen and Miller 1995). As such, it is often an object of interest to observe behaviors such as the spike firing rate and specific spike timings. Both of these represent the two paradigms in neural encoding, the *rate coding* and the *temporal coding* respectively (Kass et al. 2018). These can be observed by conducting experiments and recording the output of live neurons. Alternatively, computational simulations can be done based on neuron models.



### 1.2.5 Modeling Perspectives

Various neuron models have been created based on different perspectives of the brain’s mechanisms. One such perspective is the brain-as-a-computer which helped motivate the idea of a perceptron, which laid the groundwork for modern artificial intelligence research (Kass et al. 2018). The perceptron (Rosenblatt 1958) incorporated Hebb’s learning rule (Hebb 1949) and what McCulloch and Pitts called “A logical calculus of the ideas immanent in nervous activity” (McCulloch and Pitts 1943). However, as the statistician George Box famously said, “All models are wrong, but some are useful.” Although this model has found many applications, it does not precisely model the mechanisms of the brain, lacking any notion of spikes or spike trains. Given this, it would be more useful for our purposes to make use of more realistic models based on the biophysical properties of the brain. These would be more desirable for the goal of understanding the actual physical brain.

An alternative modeling perspective sees neurons as electrical circuits (Kass et al. 2018). This highlights how neurons can be viewed as electrically excitable components that fire when the “electrical flow” or action potential has reached a certain threshold. This modeling perspective incorporates more realistic biophysical mechanisms occurring within neurons as compared to the brain-as-a-computer approach allowing for greater insight on how neurons operate. A simplified neuron model based on this viewpoint is the *Leaky-Integrate-and-Fire* (LIF) model (Lapique 1907). On the other hand, a more complex yet biophysically accurate model is the Hodgkin-Huxley model (Hodgkin and Huxley 1952).

## 1.3 Neuron Modeling

### 1.3.1 Leaky-Integrate-and-Fire Model

The Leaky-Integrate-and-Fire (LIF) model offers a simplified but effective approximation of how neurons work (Kass et al. 2018). The model represents the membrane conductance as a single passive leakage term, ignoring all active membrane conductances and synaptic inputs (Dayan and Abbott 2001). The model’s simplicity requires fewer computational resources compared to more detailed models, making it a practical choice for simulations with limited computing power. As of writing in 2024, modern laptops are capable of running 1 second of a “small” network of 150 neurons using the Hodgkin-Huxley model in around 2 hours as shown in Chapter 2. The practical need for the LIF model may be necessary as network sizes reach the thousands or for longer simulation times.

The LIF model models the membrane conductance as a leakage term:

$$i_m(t) = \bar{g}_L(V(t) - E_L) \quad (1.1)$$

where  $i_m(t)$  represents the leakage current,  $\bar{g}_L$  is the membrane conductance,  $V(t)$  is the membrane potential, and  $E_L$  is the equilibrium potential (Dayan and Abbott 2001). The membrane potential can then be determined by the following equation:

$$C_m \frac{dV(t)}{dt} = -\bar{g}_L(V(t) - E_L) + \frac{I_e(t)}{A} \quad (1.2)$$

where  $C_m$  represents the membrane capacitance,  $\frac{dV(t)}{dt}$  is the rate of change of the membrane potential over time,  $\bar{g}_L$  is the leak conductance,  $V(t)$  is the membrane potential,  $E_L$  is the leak equilibrium potential,  $I_e(t)$  is the external current, and  $A$  is the membrane area. Multiplying both sides of equation (1.2) by the specific membrane resistance  $r_m$ , which is defined as  $r_m = \frac{1}{\bar{g}_L}$ , simplifies the equation by canceling the leak conductance term on the right-hand side and introducing the membrane time constant  $\tau_m = C_m r_m$  on the left-hand side. The input current term is modified by the factor  $\frac{r_m}{A}$ , resulting in the expression for the total membrane resistance  $R_m = \frac{r_m}{A}$ . We obtain equation (1.3):

$$\tau_m \frac{dV(t)}{dt} = -(V(t) - E_L) + R_m I_e(t) \quad (1.3)$$

which is the fundamental form of the integrate-and-fire model (Dayan and Abbott 2001). The LIF model provides an approximation of the biophysical processes in the neuron.

### 1.3.2 Hodgkin-Huxley Model

A more precise and biophysically accurate model that incorporates the physical mechanisms underlying the neuron is the Hodgkin-Huxley model (Hodgkin and Huxley 1952). The Hodgkin-Huxley model is comprised of a set of differential equations that describe the change in membrane potential over time based on ionic currents.

The membrane conductance is modeled as:

$$i_m(t) = I_L(t) + I_K(t) + I_{Na}(t) \quad (1.4)$$

where  $I_L(t)$ ,  $I_K(t)$ ,  $I_{Na}(t)$  denote the leak, potassium, and sodium currents (Hodgkin and Huxley 1952; Dayan and Abbott 2001). These ionic currents are defined as follows:

$$I_L(t) = \bar{g}_L(V(t) - E_L), \quad (1.5)$$

$$I_K(t) = \bar{g}_K n^4 (V(t) - E_K), \quad (1.6)$$

$$I_{Na}(t) = \bar{g}_{Na} m^3 h (V(t) - E_{Na}), \quad (1.7)$$

where  $\bar{g}_L$ ,  $\bar{g}_K$ , and  $\bar{g}_{Na}$  are the maximum conductances for leak,  $K^+$ , and  $Na^+$ , channels, indicating the membrane's permeability to these ions when the channels are fully open. The equilibrium potentials for these ions,  $E_L$ ,  $E_K$ , and  $E_{Na}$ , represent the membrane potential at which the net flow of the respective ion through its channel is zero. The gating variables  $m$ ,  $n$ , and  $h$  represent the probability of certain states of the ion channels (open, closed, inactivated), where  $m$  and  $h$  are associated with  $Na^+$  channels, and  $n$  is associated with  $K^+$  channels. These gating variables are dimensionless and range from 0 to 1. The full equation for the membrane potential  $V$  is given by:

$$C_m \frac{dV(t)}{dt} = -\left(\bar{g}_L(V(t) - E_L) + \bar{g}_K n^4(V(t) - E_K) + \bar{g}_{Na} m^3 h(V(t) - E_{Na})\right) + \frac{I_e(t)}{A} \quad (1.8)$$

which is similar in form to the LIF model presented in (1.3) as they both model the membrane potential (Dayan and Abbott 2001). The gating variables  $n$ ,  $m$ , and  $h$  are defined by the following differential equations:

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n, \quad (1.9)$$

$$\frac{dm}{dt} = \alpha_m(1 - m) - \beta_m m, \quad (1.10)$$

$$\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h \quad (1.11)$$

where  $\alpha_n$ ,  $\alpha_m$ , and  $\alpha_h$  determine the opening rate of the channels while  $\beta_n$ ,  $\beta_m$ , and  $\beta_h$  determine the closing rate of the channels (Hodgkin and Huxley 1952; Dayan and Abbott 2001).

### 1.3.3 Network Coupling Schemes

Having discussed some models for single neurons, we now discuss some schemes for modeling networks of neurons. Due to limited experimental data, the precise connectivity patterns between neurons are still partially unknown (Gerstner et al. 2014). However, coupling schemes approximating their connections may be used in simulations. These coupling schemes often assume random connectivity between and within neuronal populations. Gerstner et al. (2014) discuss some coupling schemes along with considerations for scaling with the number of participating neurons.

The simplest coupling scheme is full connectivity between all neurons in a population. Every connection between neurons will have the same interaction strength  $w_{ij}$ . The interaction strength  $w_{ij}$  denotes how strong the connection between neuron  $i$  and  $j$  would be.

Gerstner et al. (2014) suggest that an appropriate scaling law as the number of neurons  $N$  increases would be

$$w_{ij} = \frac{J_0}{N} \quad (1.12)$$

where  $J_0$  would be the mean interaction strength desired. The weights may also be obtained from a Gaussian distribution with mean  $J_0/N$  and standard deviation  $\sigma_0/\sqrt{N}$  in order to introduce variance in the overall interaction strength while maintaining a certain overall mean interaction strength (Gerstner et al. 2014).

Another coupling scheme would be random coupling with fixed probability. With  $N^2$  potential connections, we can randomly choose connections with probability  $p$ . The number of presynaptic input links  $C_j$  to a postsynaptic neuron  $j$  would have a mean value of  $pN$  with variance  $p(1-p)N$  (Gerstner et al. 2014). Alternatively, we can go through each neuron one-by-one and randomly select  $C = pN$  presynaptic partners for it where each neuron can be only be picked once. Similarly, we could also pick exactly  $pN$  links instead of simply imposing  $pN$  as the average (Gerstner et al. 2014). The interaction strength can be scaled with

$$w_{ij} = \frac{J_0}{C} = \frac{J_0}{pN} \quad (1.13)$$

in order to maintain the mean input to each neuron even if the number neurons in the simulation increases (Gerstner et al. 2014).

We could also consider distance dependent connectivity. Each neuron  $i$  is assigned a location  $x(i)$ . We may then assume full connectivity with strength  $w_{ij}$  modeled as

$$w_{ij} = w(|x(i) - x(j)|) \quad (1.14)$$

where  $x(i)$  and  $x(j)$  denote the location of postsynaptic and presynaptic neurons,  $w$  is a function into the real numbers. We may assume that  $w$  vanishes for distances  $|x(i) - x(j)| > d$  (Gerstner et al. 2014). Alternatively, we may give all the connections the same weight with a connection probability dependent on the distance

$$\text{Prob}(w_{ij} = 1) = P(|x(i) - x(j)|) \quad (1.15)$$

where  $x(i)$  and  $x(j)$  denote the location of postsynaptic and presynaptic neurons, and  $P$  is a function into the interval  $[0, 1]$  (Gerstner et al. 2014).

## 1.4 The Olfactory Bulb

This section will give an overview of the structure of the olfactory bulb along with how it is modeled. These will be discussed in order to provide context for the selected simulation study (Li and Cleland 2017). This section will assume that the reader is unfamiliar with the biological mechanics of the olfactory bulb and will go into more detail on that topic.

### 1.4.1 Neural Circuitry and Oscillatory Dynamics

The Olfactory Bulb (OB) is a key neural structure for olfactory information processing in both vertebrates and insects, representing the initial phase of processing after the olfactory receptors (Dayan and Abbott 2001). The following description of the neuroanatomy of the olfactory bulb is taken and summarized from Nagayama et al. (2014). Olfactory receptor neurons detect odor molecules and project their axons to the OB, where these axons end in structures called glomeruli (see Figure 1.2). Within these glomeruli, they form synapses with mitral and tufted cells, as well as with local interneurons such as periglomerular (PG) cells, external tufted cells, and superficial short-axon (sSA) cells. The mitral and tufted cells are instrumental in generating the output of the OB, projecting to the primary olfactory cortex. They also connect with inhibitory granule cells within the OB, which serve to modulate the activity of the mitral and tufted cells. Granule cells, specifically, are axon-less interneurons that extend their dendrites apically into the external plexiform layer (EPL), where they form reciprocal synapses with the secondary dendrites of mitral and tufted cells. The complex network within the OB, encompassing several layers including the EPL, internal plexiform layer (IPL), and granule cell layer (GCL), is essential for the translation of odor stimuli into the brain's rapid gamma-band oscillations (30 - 80 Hz). These oscillations are marked by synchronized spikes that are highly coherent across the OB (Kay and Lazzara 2010) and are fundamental to the OB's function, playing a critical role in the sensory processing of smells and reflecting the oscillatory nature of its neural dynamics (Li and Cleland 2017; Nagayama et al. 2014).

These oscillations may often be identified by means of a power spectral density plot (PSD). A PSD plot for spectral analysis visually represents the distribution of power into frequency components of a signal, showing how much of the signal's power is contained within specific frequency bands. One such PSD plot may be seen as the third plot in Figure 2.6 which shows frequencies in the gamma-band (30 - 80 Hz).

The study of the OB's mechanisms is based on the hypothesis that OB field oscillations play a central role in the neural processing of olfactory information. Specifically, they are believed to facilitate the transformation of odor stimuli into a secondary, spike-based

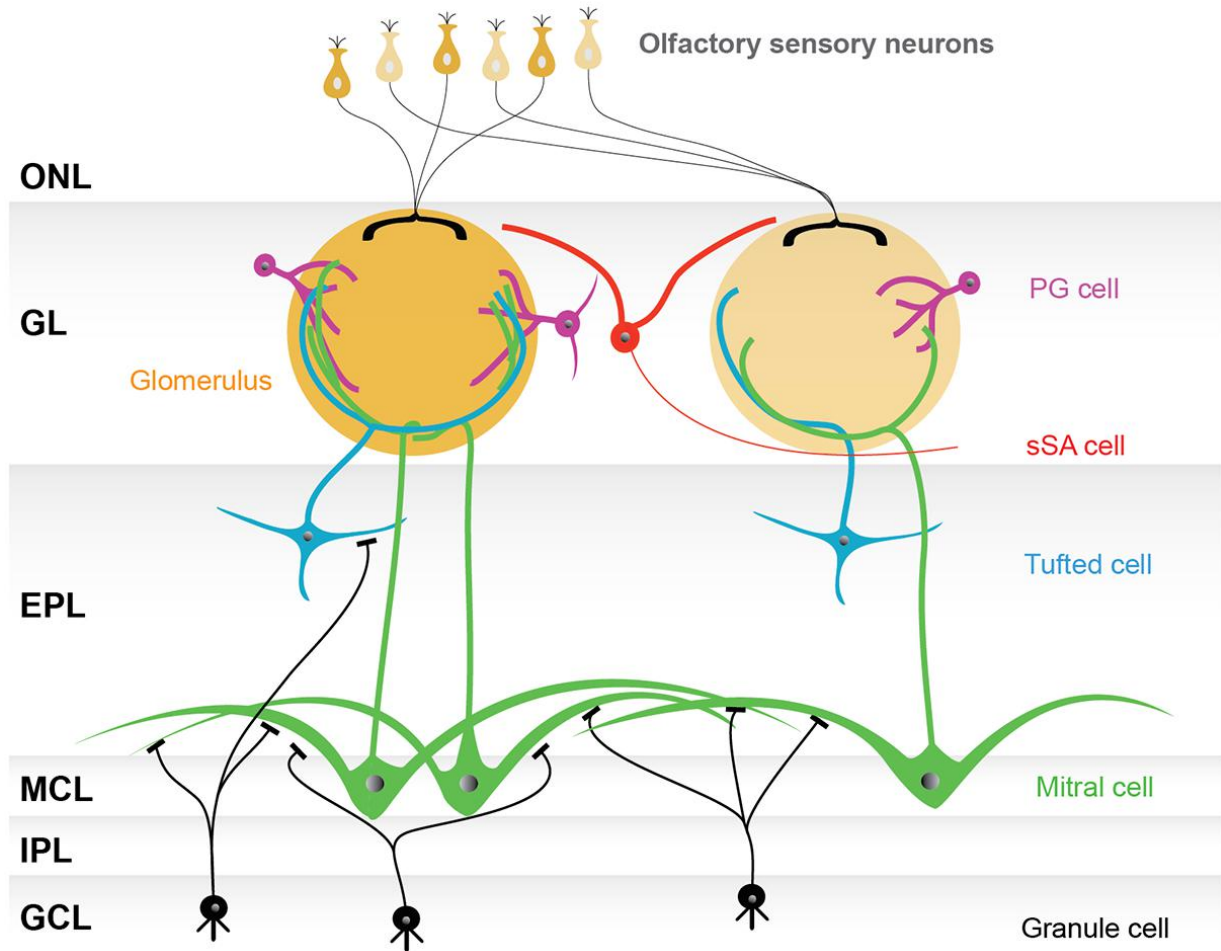


Figure 1.2: Basic model of the Olfactory Bulb Network. Olfactory receptor neuron axons converge onto spherical neuropil structures called glomeruli, where they synapse with output mitral/tufted cells as well as local interneurons like periglomerular cells. Mitral/tufted cells project out of the bulb while also receiving inhibitory synapses from granule cells in the external plexiform layer. This reciprocal mitral-granule cell interaction generates synchronized gamma oscillations critical for olfactory processing. Adapted from Nagayama et al. (2014).

electrical representation of information, which is then relayed to other regions of the brain for further processing and integration (Li and Cleland 2017). To fully grasp the nature of the secondary representation of olfactory information, it is important to understand the mechanisms through which OB field oscillations are generated.

The oscillations are generated through a fast negative feedback loop, a process that is mediated by the orchestrated interaction of specific neuronal types (Li and Cleland 2017). The primary constituents of this feedback loop are the Mitral Cells (MCs) and the Granule Cells (GCs).

The MCs serve as the principal output neurons (Li and Cleland 2017; Dayan and Abbott 2001). They are characterized as mitral and projecting tufted cells, playing a crucial role in transmitting olfactory information by sending projections to the primary olfactory cortex (Li and Cleland 2017; Dayan and Abbott 2001).

The GCs function as inhibitory interneurons, inhibiting MC activity (Li and Cleland 2017; Dayan and Abbott 2001). Their inhibitory action is essential for the modulation and fine-tuning of the olfactory bulb's output.

The interaction between Mitral Cells and Granule Cells is mediated through dendritic synapses located within the External Plexiform Layer (EPL) (see 1.2). The interaction between these two cell types must be considered in any model of the OB in order to capture the oscillation generation mechanism.

### 1.4.2 Network Architectures Simulating the Olfactory Bulb

The precise mechanism underlying gamma oscillogenesis in the olfactory bulb remains unclear. Li and Cleland (2017) describe several dynamical architectures.

First they describe the Pyramidal Interneuron Network Gamma (PING). The PING architecture is inspired by the anatomical dominance of excitatory-inhibitory reciprocal synapses in the EPL. However, it fails to incorporate cellular resonance properties like the intrinsic subthreshold oscillations found in MCs.

Next, they describe the Interneuron Network Gamma (ING) architecture. ING relies on the inhibitory interactions among granule cells, as suggested by EEG studies, and GABAergic input onto granule cells. Nonetheless, this theory has been largely discounted.

They also mention an architectural approach that posits that OB network oscillations are driven by the intrinsic subthreshold oscillations/dynamics in MCs. This approach successfully captures the dynamic behavior of STOs in MCs and resolves some limitations of the PING architecture. However, it requires a higher frequency of STOs in MCs than

experimentally observed and does not align with the sparse spiking behavior of granule cells.

Lastly, the Pyramidal Resonance Interneuron Network Gamma (PRING) presents a hybrid model, featuring an inhibitor-coupled intrinsic cellular oscillator. In this model, the intrinsic STOs of MCs are transiently coupled during afferent activation into a coherent oscillatory network, paced by GC-mediated inhibitory synaptic inputs that periodically reset the slower MC STOs. This model exhibits characteristics of the PING architecture while also acknowledging the role of STOs in MCs. The study by Li and Cleland (2017) focuses on modeling the PRING architecture, extending the architecture from one of their previous studies (Li and Cleland 2013).

## 1.5 Olfactory Bulb Simulation Study

The following section presents the simulation study which will be replicated in this paper. The study by Li and Cleland (2017) models the olfactory bulb using their proposed PRING architecture. They focused on the dynamics of gamma oscillations in response to odor stimulation. They discussed various experimental setups created by modifying the parameters of their model. A summary of their key findings will first be presented followed by their model specifications.

### 1.5.1 Key Findings

First, Li and Cleland (2017) demonstrated that odor stimulation elicits broadly coherent gamma oscillations throughout the olfactory bulb network, a finding corroborated using the default network configuration and simulated odorant stimuli.

The importance of Mitral cell STOs in network synchronization was also assessed by experimentally removing STOs from Mitral cells. This was achieved by substituting the persistent sodium current, highlighting the role of STOs in the coherence of network oscillations.

Next, an investigation into the interaction between network time constants and intrinsic STO frequencies revealed that, during sensory activation, faster network time constants supersede the intrinsic STO frequencies of Mitral cells. This was tested by modulating the intrinsic STO frequency of Mitral cells through adjustments to the slow potassium current  $I_{KS}$  activation variable's time constant and the conductance densities of  $I_{KS}$  and  $I_{NaP}$ .



Following that, the study also explored the impact of inhibitory synaptic weights on the strength and coherence of olfactory bulb gamma oscillations. This was tested by varying the synaptic weight from Granule Cells to Mitral Cells ( $W_{GC \rightarrow MC}$ ), indicating that optimal weights are crucial for robust oscillations.

Afterwards, the robustness of oscillatory dynamics to changes in excitatory synaptic weights was evaluated by adjusting the synaptic weight from Mitral Cells to Granule Cells ( $W_{MC \rightarrow GC}$ ). The dynamics were found to be tolerant even with increases up to eight times the default value. Sixth, the researchers assessed the robustness of gamma oscillations to variations in the intensity of afferent input, demonstrating that oscillations remain stable despite heterogeneity in excitation across the Mitral cell population induced by simulated odorant stimuli.

The role of glomerular-layer inhibition in enabling gamma oscillations was also investigated by modulating the synaptic weight from Periglomerular Cells to Mitral Cells ( $W_{PGC \rightarrow MC}$ ). Findings suggest that glomerular-layer inhibition facilitates oscillations by constraining Mitral cell excitation and reducing firing rate heterogeneity.

Lastly, the effect of network size on gamma oscillations was examined by incrementally increasing the number of Granule Cells while maintaining constant numbers of Mitral and Periglomerular cells. The results indicate that gamma oscillations are maintained across various network sizes, underscoring their robustness.

## 1.5.2 Computational Details

The following section describes and summarizes the simulation of the olfactory bulb’s network architecture. The network is composed of 25 Mitral Cells (MCs) as principal output neurons, 25 Periglomerular Cells (PGCs) for input modulation, and 100 Granule Cells (GCs) for inhibitory functions.

This setup is structured into units, each consisting of an MC and a PGC, representing the columns of the olfactory bulb that are linked to specific olfactory receptors (see [1.3](#)).

To simulate the spatial organization of the olfactory bulb and to address edge issues, the neurons are arranged in a two-dimensional grid with a size of 1mm x 1mm, and a toroidal configuration is applied. Neurons are labeled by their grid location, using column and row numbers for precise identification and analysis.

Hodgkin-Huxley-based models are used to model the MCs, PGCs, and GCs. The schematic representation of the network’s connectivity and structure can be found in figure [1.3](#), which illustrates the dendrodendritic synaptic connections among the MCs, PGCs,

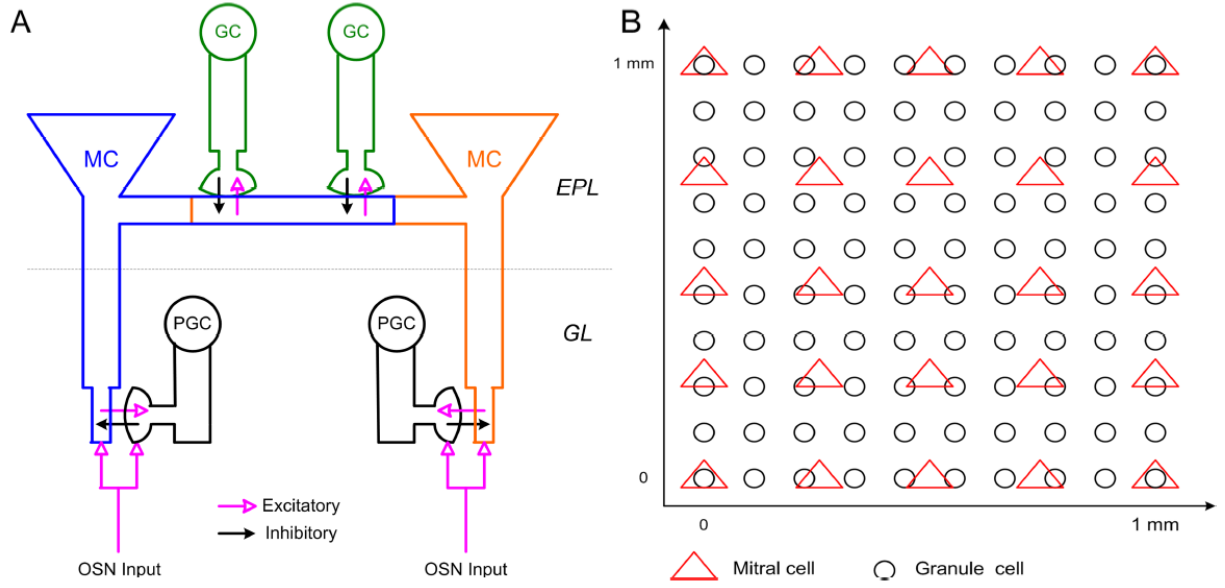


Figure 1.3: Schematic representation of OB network connectivity and model structure. **A:** Schematic representation of dendrodendritic synaptic connectivity among MCs, PGCs, and GCs. Reciprocal dendrodendritic synaptic connections exist between the MC tuft and PGC spines, and between the MC lateral dendrite and GC spines. GL: glomerular layer; EPL: external plexiform layer. **B:** Spatial localization of MCs and GCs across the two-dimensional toroidal surface of the model OB (1 mm × 1 mm). Adapted from Li and Cleland (2017).

and GCs, as well as the spatial localization of the MCs and GCs across the model's two-dimensional surface. These were computationally simulated using [NEURON \(2024\)](#). For further details on the differential equations and constants used within the model, readers are referred to the researchers' previous study (Li and Cleland [2013](#)).

To simulate the input from odorants to olfactory sensory neurons (OSNs), a sigmoidal function is utilized, expressed as

$$I_{\text{OSN}} = u_0 + 0.5(u_s - u_0) \left[ \tanh \left( \frac{3(t - t_{\text{ORN}})}{r} - 3 \right) + 1 \right] \quad (1.16)$$

where  $u_0$  represents the pre-odor value (simulating the pure air input),  $u_s$  denotes the steady-state value post odor excitation,  $r$  is the parameter determining the transition rate from  $u_0$  to  $u_s$ , and  $t_{\text{ORN}}$  signifies the time at which the odor onset occurs.

# Chapter 2

## Simulation Replication

Replication and reproduction of results is an essential aspect of academic research. With the reproducibility crisis in various fields (Baker 2016; Peng 2015; Hutson 2018), reproducing the results of studies has become a more important task. Although computational research may appear more reproducible due to the dependence on code, code accompanying published research is often pseudocode, incomplete, or completely unavailable (Hutson 2018). Even when the code is completely available, tools and algorithms that involve randomness may also face issues with reproducibility (Hutson 2018). Such is the case with computational neuroscience research. As such, replicating and reproducing a simulation study proves to be a valuable task for verifying computational tools and methods.

The following chapter describes the process of replicating the main default simulation in an olfactory bulb simulation study (Li and Cleland 2017). The chapter is intended for future students and researchers who wish to follow the given workflow. The reader is assumed to already be familiar with NEURON and R. Detailed descriptions of each step taken towards replicating the paper will be presented along with corresponding code and their outputs.

This chapter will assume that the operating system in use is either macOS or Linux with NEURON 8.2.4, Visual Studio Code 1.87, and Python 3.10.12 already installed. For those using Windows, it is preferable to use the Windows Subsystem for Linux which can be installed from the Microsoft Store. This is because some NEURON simulations may run into errors when being run directly on Windows.

The following steps were performed on a laptop equipped with an Intel(R) Core(TM) i7-7500U CPU at 2.70GHz, capable of boosting up to 2.90 GHz,. There are 16.0 GB of installed RAM. A 2TB Samsung SSD 870 EVO is used for storage, running on a 64-bit

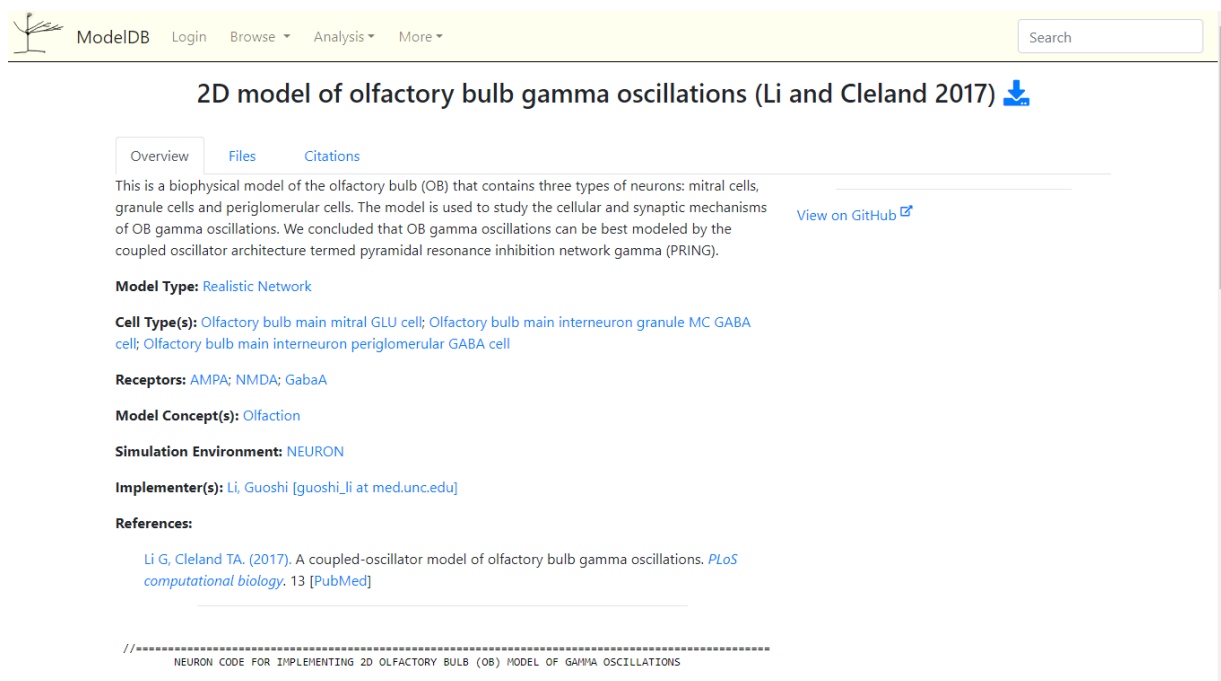


Figure 2.1: ModelDB Overview Tab. This is the initial tab displayed upon visiting a model page on ModelDB.

Windows 10 Home Single Language edition (version 22H2). A full simulation of the OB network model (Li and Cleland 2017) from the study took approximately 2 hours on this device.

## 2.1 Downloading the Model

The NEURON model for the following section can be found in ModelDB (McDougal et al. 2017) at <https://modeldb.science/232097>. Upon visiting the webpage, the *Overview* tab should be present as shown in Figure 2.1. The files for the model may be viewed under the *Files* tab as shown in Figure 2.2. Each file may be viewed within the webpage and downloaded individually as needed.

To download all the files, click the download icon beside the model's title. This will prompt the download of a zip file named `232097.zip` which contains all the model's associated files. Save it in the desired folder or check the downloads folder. Unzip the downloaded file which should contain the folders `_MACOSX` and `OBGAMMA` as shown in Figure 2.3. The

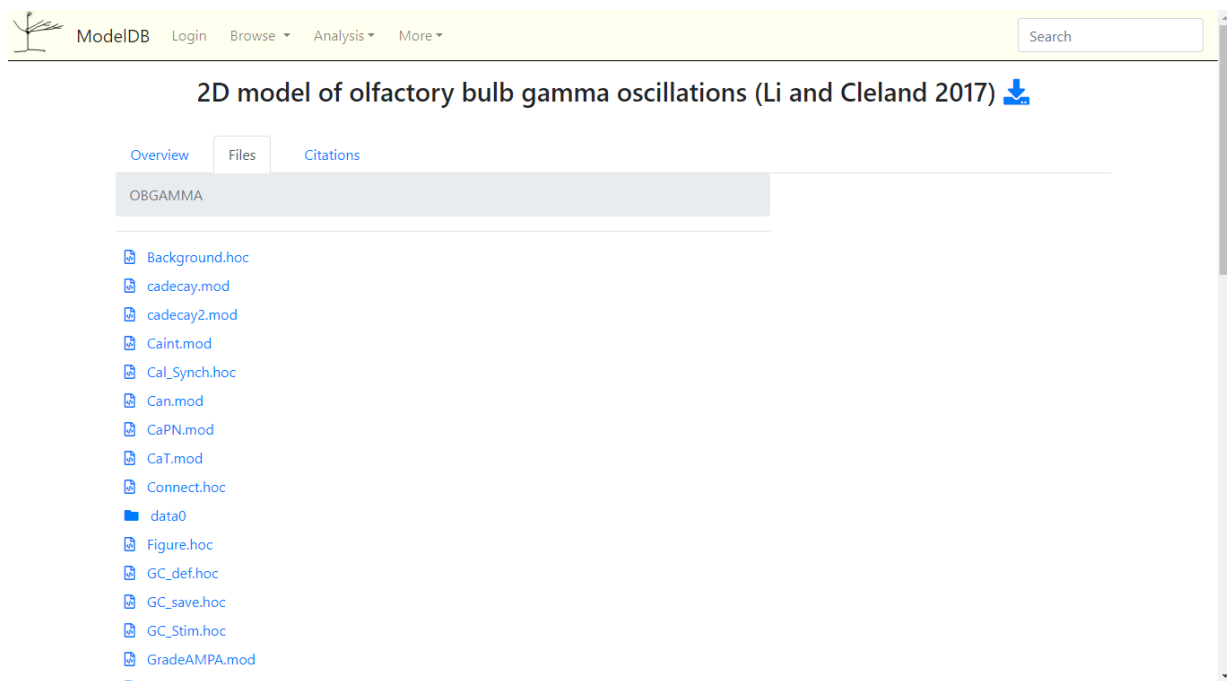


Figure 2.2: ModelDB Files Tab. Model files may be individually viewed here before downloading.

| Name       | Type               | Size   |
|------------|--------------------|--------|
| _MACOSX    | File folder        |        |
| OBGAMMA    | File folder        |        |
| 232097.zip | WinRAR ZIP archive | 101 KB |

Figure 2.3: Folders from Unzipped File. The folders `_MACOSX` and `OBGAMMA` should be present.

relevant files for the model will be found in the `OBGAMMA` folder. For those using macOS, the `OBGAMMA` folder within the `_MACOSX` folder should be used. Lastly, in the `OBGAMMA` folder, create two new empty folders named `connection` and `input`. These folders will be needed later on when running the simulation. Otherwise, the program will not execute properly.

## 2.2 Running the Simulation in Python

The following section will detail how to run the olfactory bulb neuron network simulation in Python. It is expected that the necessary files from the previous section have already been downloaded. It is also expected that the necessary folders, `connection` and `input`, have been created. A more detailed step-by-step guide may be found in Appendix A.

To run the simulation using Python, first open the `OBGAMMA` folder in Visual Studio Code. Create a new Python virtual environment. Using `pip`, install both the `NEURON` and `ipykernel` modules. Afterwards, type the command `'nrnivmodl'` in the terminal to compile the `NEURON` files.

Inside a new jupyter notebook, the following code will load and execute a demo simulation:

```
from neuron import h
h.load_file("mosinit.hoc")
h.demo_run()
```

A successful execution will populate the `data0` folder with data files following formats such as `Gd_i_j`, `Ms_i_j`, `Vgb_i_j`, `Vms_i_j`, etc. The full simulation may be run by replacing `h.demo_run()` with `h.default_run()`. The data obtained from the full simulation will be used to reproduce the plots from the study (Li and Cleland 2017) in R.

## 2.3 Reproducing Plots in R

The following will focus on reproducing the plots found in Figure 2 of the study (Li and Cleland 2017) in R. All code used to generate the plots may be found in Appendix B. It is assumed that a full simulation of 3 seconds with a time delta of 0.2 ms was run and that the data is accessible from the `data0` folder in `OBGAMMA`. All plots only display data from the final 1 second of simulation since the simulated odorant stimulus was only introduced during this final second. The figures presented here were assessed to be visually identical to the original figures. Assessing whether the figures are numerically identical is unfortunately not possible since the study (Li and Cleland 2017) did not provide the numerical data used to generate the original figures.

### 2.3.1 Figure 2A

Figure 2A from the study presents two plots. The first plot, replicated in Figure 2.4, displays the steady-state input intensities of the simulated odorant 1.16 for the olfactory sensory neurons (OSN). The second plot, replicated in Figure 2.5, displays the firing rate of each of the 25 mitral cells during the stimulus. The x-axis label has been changed from "Glom #" to "MC #" to clarify that each bar represents a mitral cell.

The data for Figure 2.4 was readily available in the `Odor` file. The data for Figure 2.5 was obtained by aggregating across the spike timing data for each mitral cell. The spike timing files for mitral cells follow the format `Ms_i_j` where  $i, j = 0, \dots, 4$ .



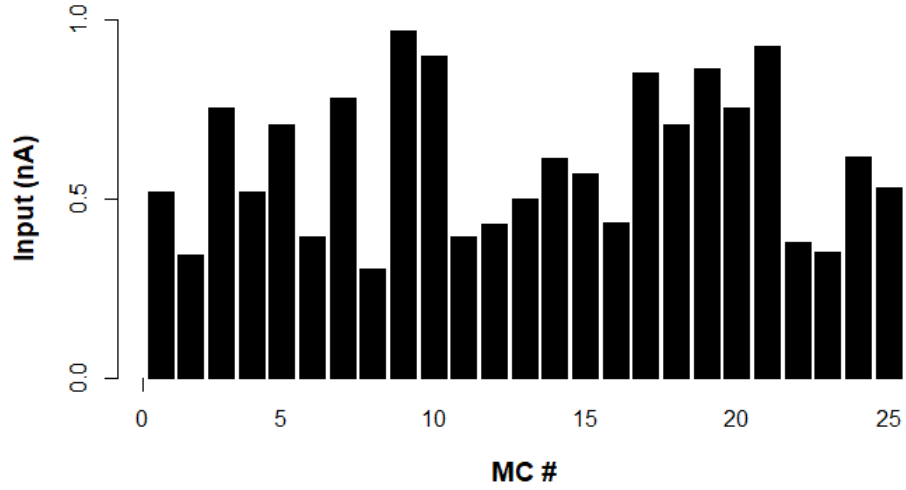


Figure 2.4: OSN Input. Simulated steady-state input intensities to the olfactory sensory neurons (OSNs) corresponding to the presented odorant stimulus pattern.

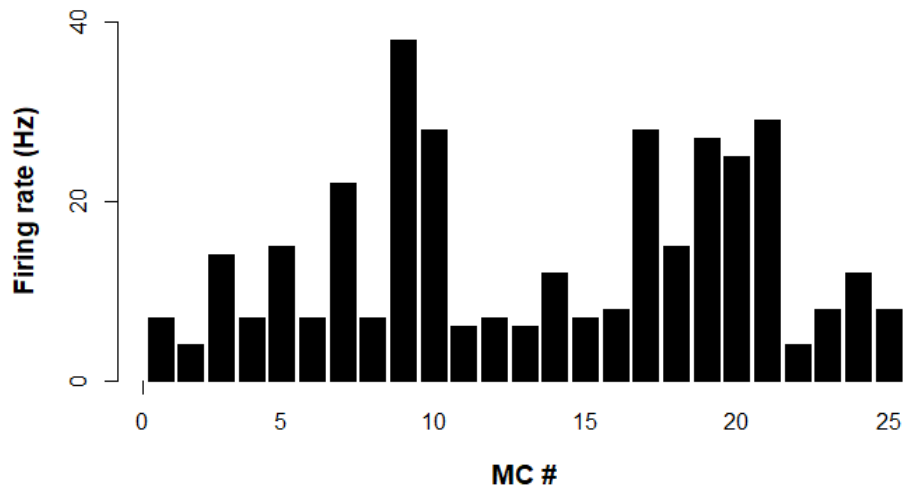


Figure 2.5: OSN Firing Rates. Firing rates of the 25 mitral cells in response to the odorant stimulus pattern shown in Figure 2.4.

### 2.3.2 Figure 2B

Figure 2B from the study presents three plots: the simulated Local Field Potential (sLFP) along with its corresponding Auto-Correlation Function and Power Spectral Density plots. These have been replicated in Figure 2.6. The sLFP was obtained by filtering the mean membrane potentials across all Mitral Cells. The data for the mean somatic membrane potentials for MCs were readily available in the `Vam` file. Filtering was carried out numerically using a band-pass filter (10 – 100 Hz) using the `signal` library in R. A bandpass filter in signal processing allows frequencies within a certain range to pass through while attenuating frequencies outside of that range, effectively isolating a specific band of frequencies from a broader spectrum. The auto-correlation function was obtained by applying the `ccf` function on the sLFP. The power spectrum of the signal was obtained by a fast Fourier transform (FFT) of the filtered sLFP using the `fft` function. More details on the method may be found in a previous study by Li and Cleland (2013).

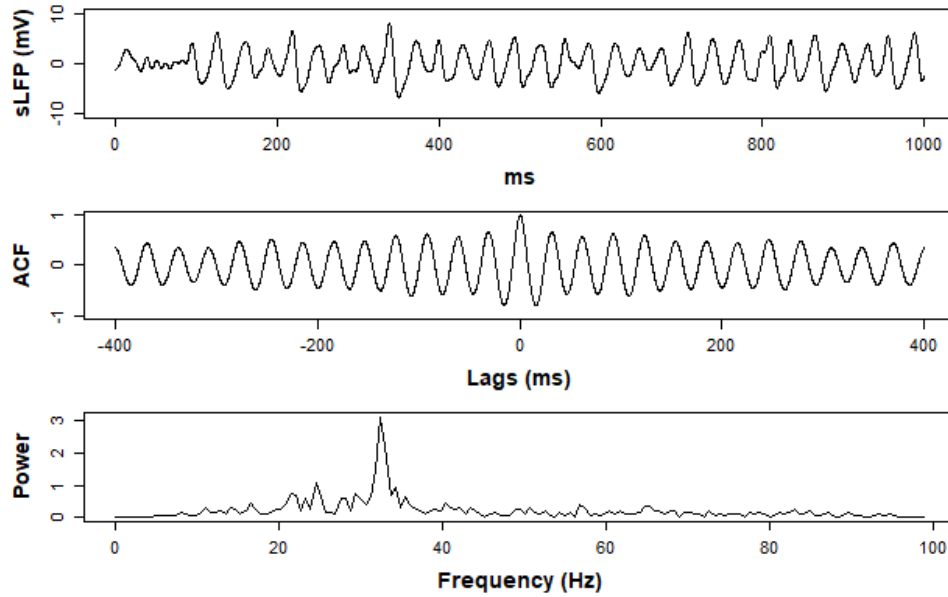


Figure 2.6: Analysis of the simulated local field potential (sLFP) signal derived from mean mitral cell membrane potentials. Top: Filtered sLFP time series in the gamma frequency band (10-100 Hz). Middle: Autocorrelation function of the sLFP revealing oscillatory dynamics. Bottom: Power spectral density of the sLFP showing a prominent gamma oscillation peak around 30 Hz.

### 2.3.3 Figure 2C

Figure 2C from the study presents the recorded voltages of four MCs. This was replicated in Figure 2.7. The MC with index  $i = 0, j = 1$  is presented against the MC with index  $i = 4, j = 1$  in the top plot. These voltage responses exhibit sub-threshold oscillations along with sparse spikes (top). In the bottom plot, The MC with index  $i = 3, 4$  is presented against the MC with index  $i = 2, j = 4$ . This pair exhibits more frequent spiking with ongoing sub-threshold oscillations.

The MC voltage data was readily available from the files with format `Vms_i_j` with the indexes  $i, j = 0, \dots, 4$ .

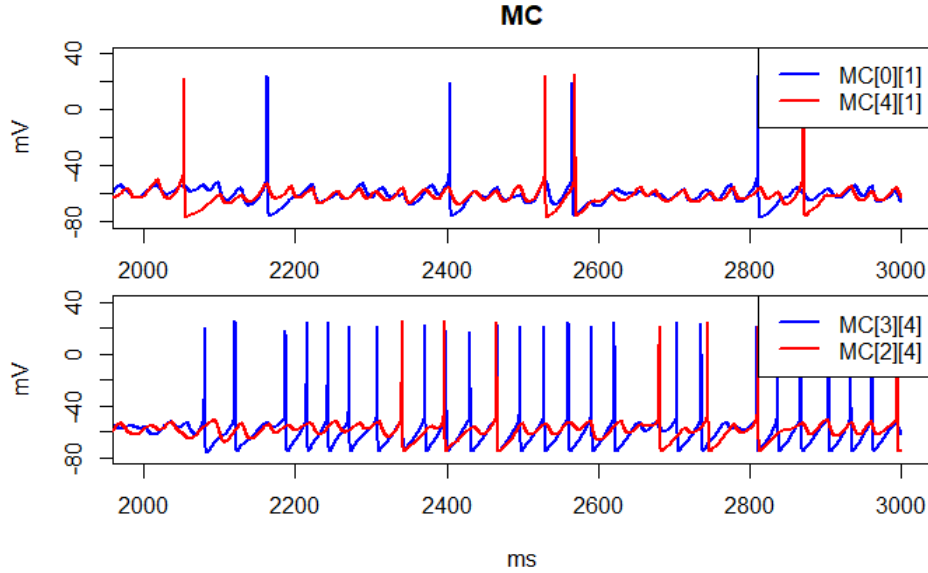


Figure 2.7: Comparison of membrane potentials from four different mitral cells during the odor response, exhibiting subthreshold oscillatory activity and sparse spiking. Top: Two mitral cells, (0, 1) and (4, 1), exhibiting sub-threshold oscillations along with sparse spikes. Bottom: Two other mitral cells, (3,4) and (2,4), display more frequent spiking with ongoing sub-threshold oscillations.

### 2.3.4 Figures 2D1, 2E1, 2F1

Figures 2D1, 2E1, and 2F1 from the study present some voltage responses from each type of cell: MC, GC, and PGC. Figure 2.8 presents a select pair of voltages with one experiencing mixed sub-threshold oscillations and the other showing dense activity. Figures 2.9 and 2.10 present the typical voltage responses for these two types of cells after the stimulus.

The data for these plots were obtained similarly as in Section 2.3.3. The GC voltage data was obtained from files with the format  $V_{gb\_i\_j}$  with  $i, j = 0, \dots, 9$ . The PGC voltage data was obtained from files with the format  $V_{pb\_i\_j}$  with  $i, j = 0, \dots, 4$ .

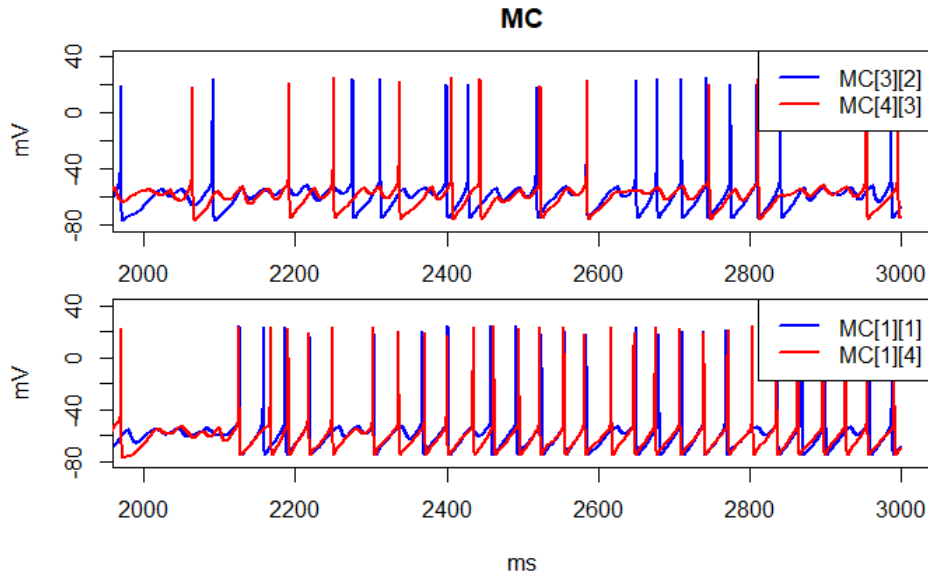


Figure 2.8: Voltage responses of two MCs exhibiting mixed STOs and spikes (top) and another two MCs exhibiting dense spiking activity (bottom) during odor presentation

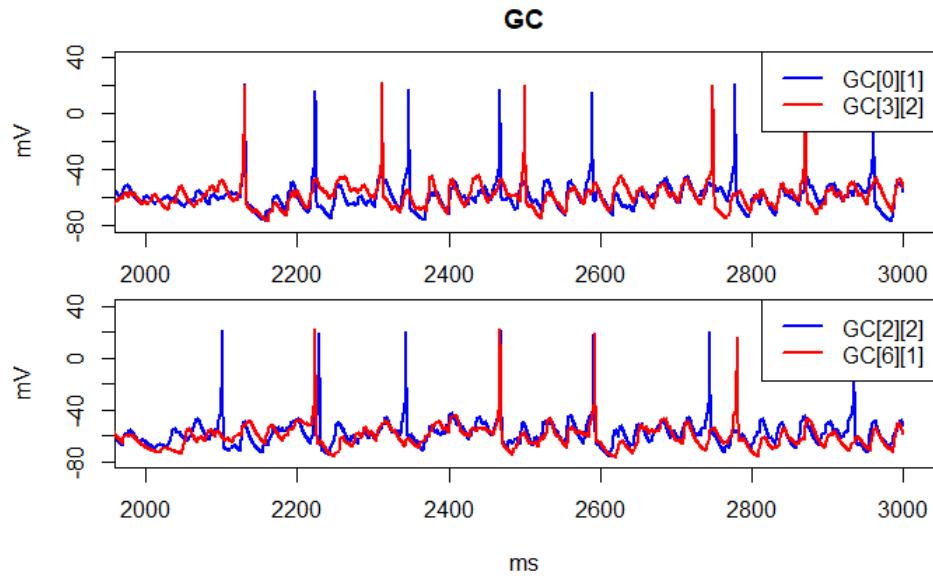


Figure 2.9: Voltage responses of two typical pairs of GCs during odor presentation.

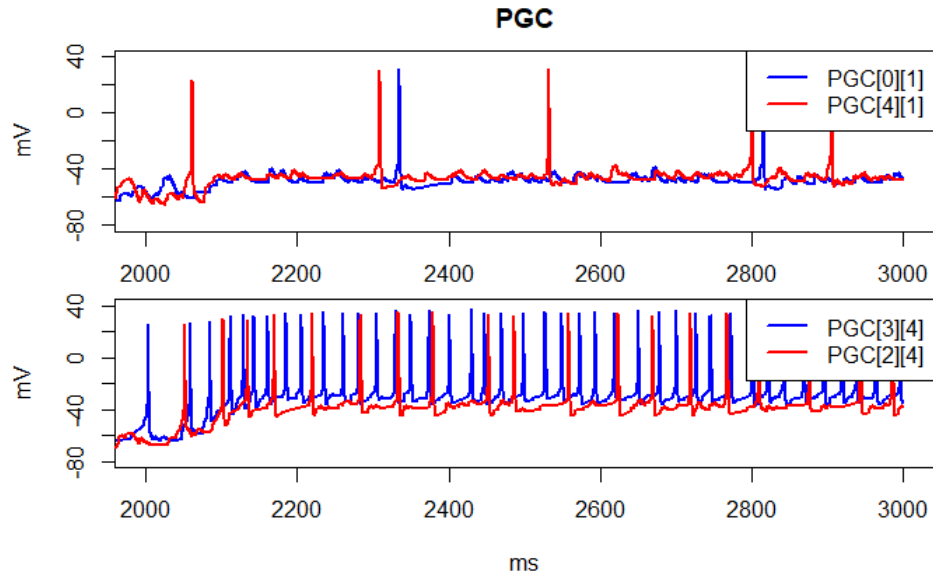


Figure 2.10: Voltage responses of two typical pairs of PGCs during odor presentation.

### 2.3.5 Figures 2D2, 2E2, 2F2

Figures 2D2, 2E2, and 2F2 from the study present the raster plots of all the cells of each cell type. The replicated raster plot for MCs, GCs, and PGCs can be seen in Figures 2.11, 2.12, and 2.13 respectively. These were slightly modified to only present the spikes that occurred during odor stimulation as the figures from the study includes some spikes prior to the stimulus. Raster plots are visual representations of spike timings of neurons over a period of time, with each row corresponding to a different neuron and each mark indicating the timing of a spike. The plots for GC, PGC, and MC show the discrete firing events across a network of neurons, with varying degrees of synchrony and firing rate.

The data used to generate the plots were obtained from files with the format **Ms\_i\_j** with  $i, j = 0, \dots, 4$  for MCs, **Gs\_i\_j** with  $i, j = 0, \dots, 9$  for GCs, and **Ps\_i\_j** with  $i, j = 0, \dots, 4$  for PGCs. For each raster plot, all the files for the corresponding cell type were loaded then plotted using the **rect** function.

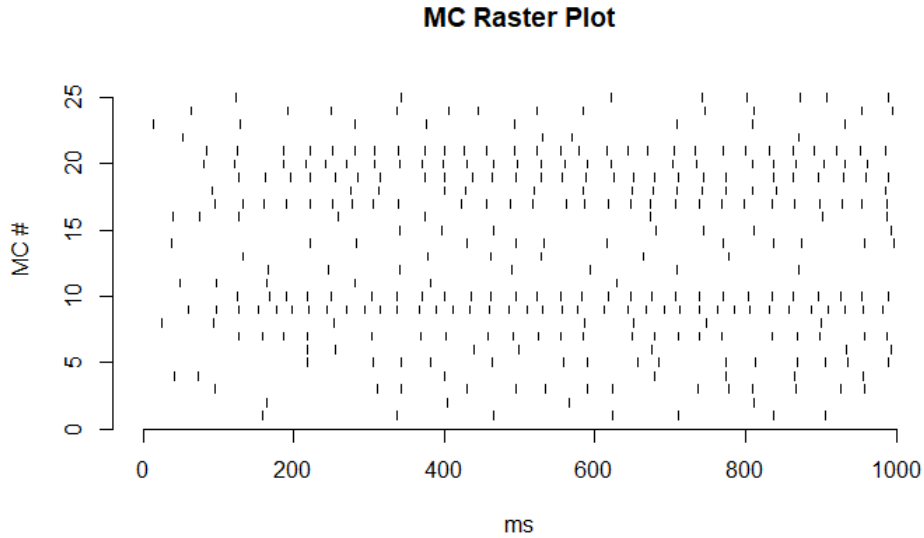


Figure 2.11: Raster plot representing the spike timings of mitral cells (MCs) in response to odor stimulation over a 1000 ms duration. Each row corresponds to a distinct MC, and each dot represents a spike. This plot emphasizes the varied response of MCs during the stimulus period.

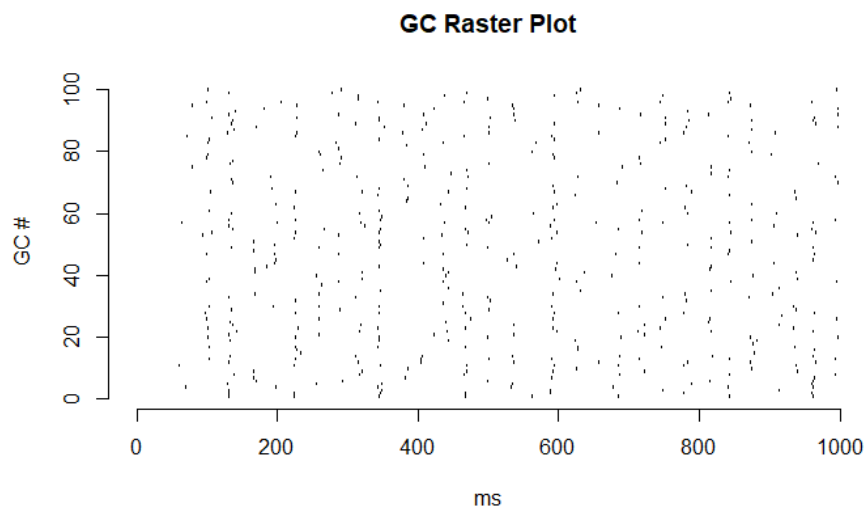


Figure 2.12: Raster plot representing the spike timings of granule cells (GCs) in response to odor stimulation over a 1000 ms duration.

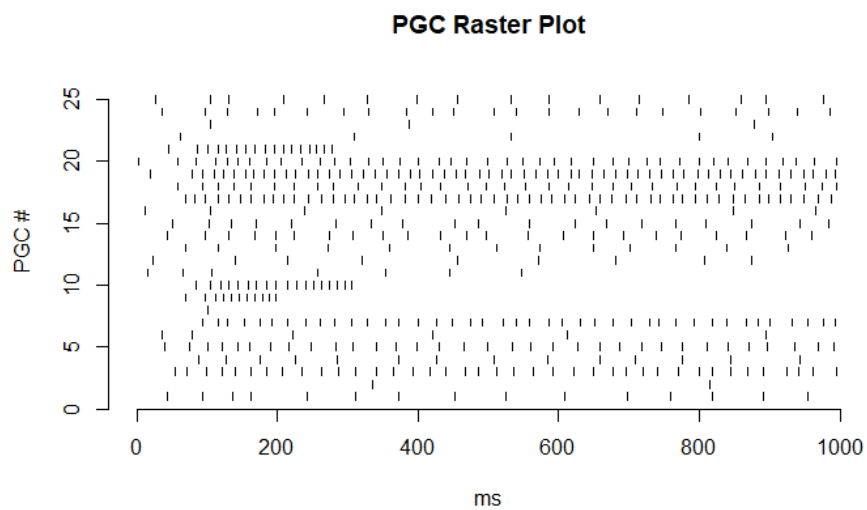


Figure 2.13: Raster plot representing the spike timings of periglomerular cells (PGCs) in response to odor stimulation over a 1000 ms duration.

### 2.3.6 Figures 2D3, 2E3, 2F3

Figures 2D3, 2E3, and 2F3 from the study present the population activity of the MCs, GCs, and PGCs respectively. The corresponding power spectrums are also displayed. These are replicated in Figures 2.14, 2.15, and 2.16 respectively. The population activity represents the firing intensity among cells of the specified type.

This population activity is an aggregated count of the number of spikes across all cells of the specified type in the duration of 1 second. The data for the number of spikes is obtained from the spike timings loaded for the raster plots in section 2.3.5. The aggregated count is computed in the same way a histogram with a bin width of 5 ms is computed. Instead of plotting the aggregated counts as a bar plot, they are instead plotted as a line plot. The aggregated binned counts may then be treated as a time series, and a Fourier transform may be applied to obtain the power plots. These were choices made by Li and Cleland (2017) in order to represent and analyze the neuronal population activity of MCs, GCs, and PGCs.

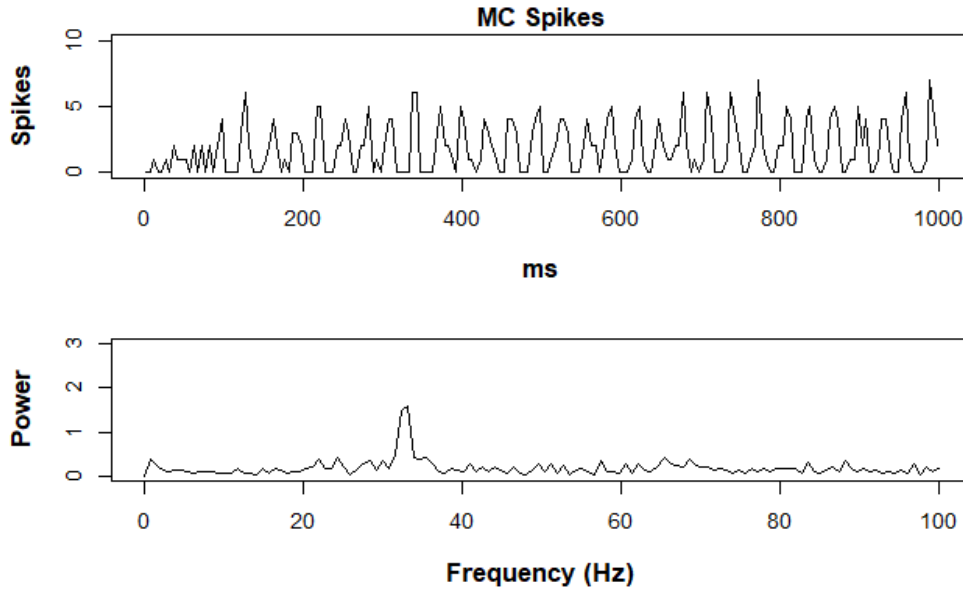


Figure 2.14: Population activity of mitral cells during the 1 second odor stimulus, represented as the aggregate spike count across all mitral cells in 5 ms time bins (top). The corresponding power spectrum (bottom) shows a prominent gamma oscillation peak around 30 Hz.



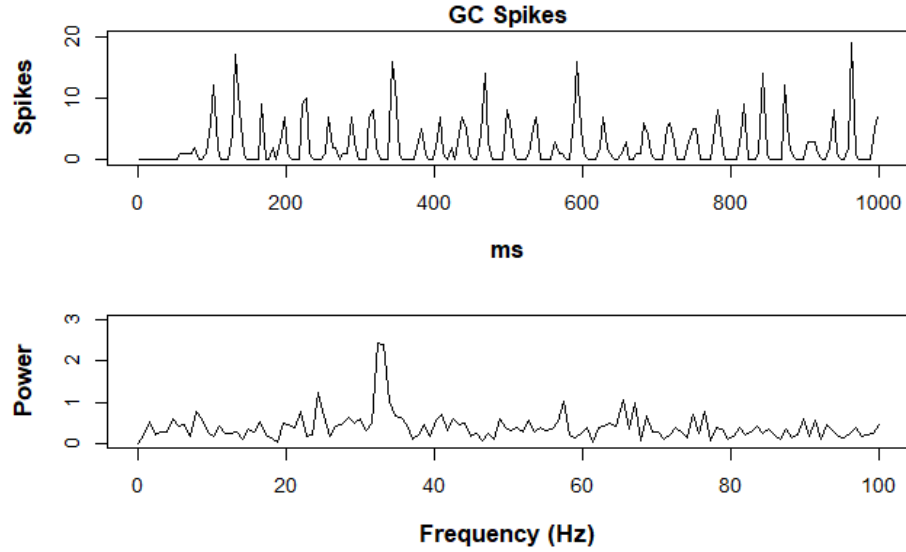


Figure 2.15: Population activity of granule cells during the 1 second odor stimulus, represented as the aggregate spike count across all mitral cells in 5 ms time bins (top). The corresponding power spectrum (bottom) shows a prominent gamma oscillation peak around 30 Hz similar to Figure 2.14.

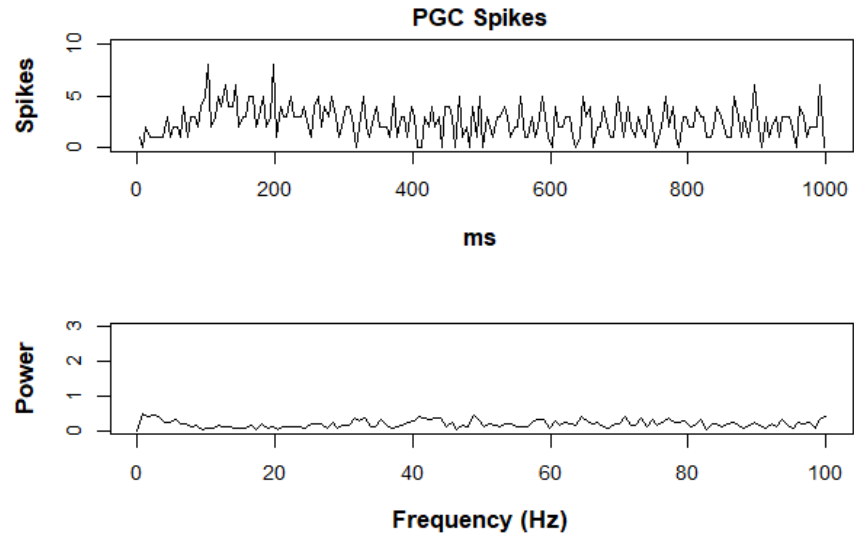


Figure 2.16: Population activity of periglomerular cells during the 1 second odor stimulus, represented as the aggregate spike count across all mitral cells in 5 ms time bins (top). The power spectrum (bottom) lacks a distinct peak.

# Chapter 3

## Simulation Modifications

The following chapter will describe and discuss the results of different modifications to the default simulation. We will discuss the results of changing the initial simulation seed with the default parameters. We will also discuss the results of modifying the connection probability between mitral cells and granule cells. The connection probability will be modified to create a sparse and full connectivity scenario.

### 3.1 Random Seeds

The results in the following section were obtained by modifying the seed variables in the Parameter.hoc file, namely `seedU`, `seedN`, and `NSSEED`. Two simulations were executed with different seeds. All three seed variables were set to the same values for each simulation. The specific seed values used for the new simulations were 37 and 73. Each simulation ran for approximately 2 hours, similar to the default simulation.

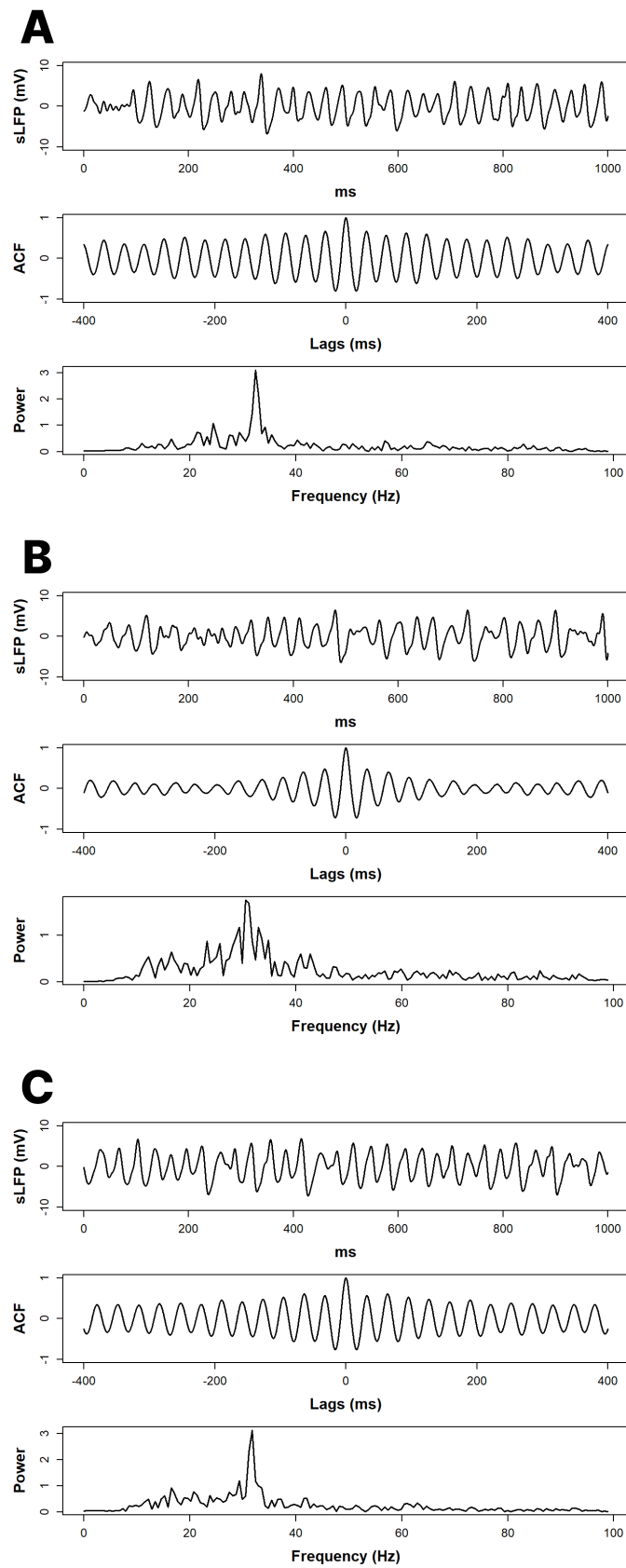


Figure 3.1: LFP, ACF, and PSD plots show consistent behavior with slight variations across seeds. **A:** LFP, ACF, and PSD plots for the default seed LFP. **B:** LFP, ACF, and PSD plots for random seed 1. **C:** LFP, ACF, and PSD plots for random seed 2.

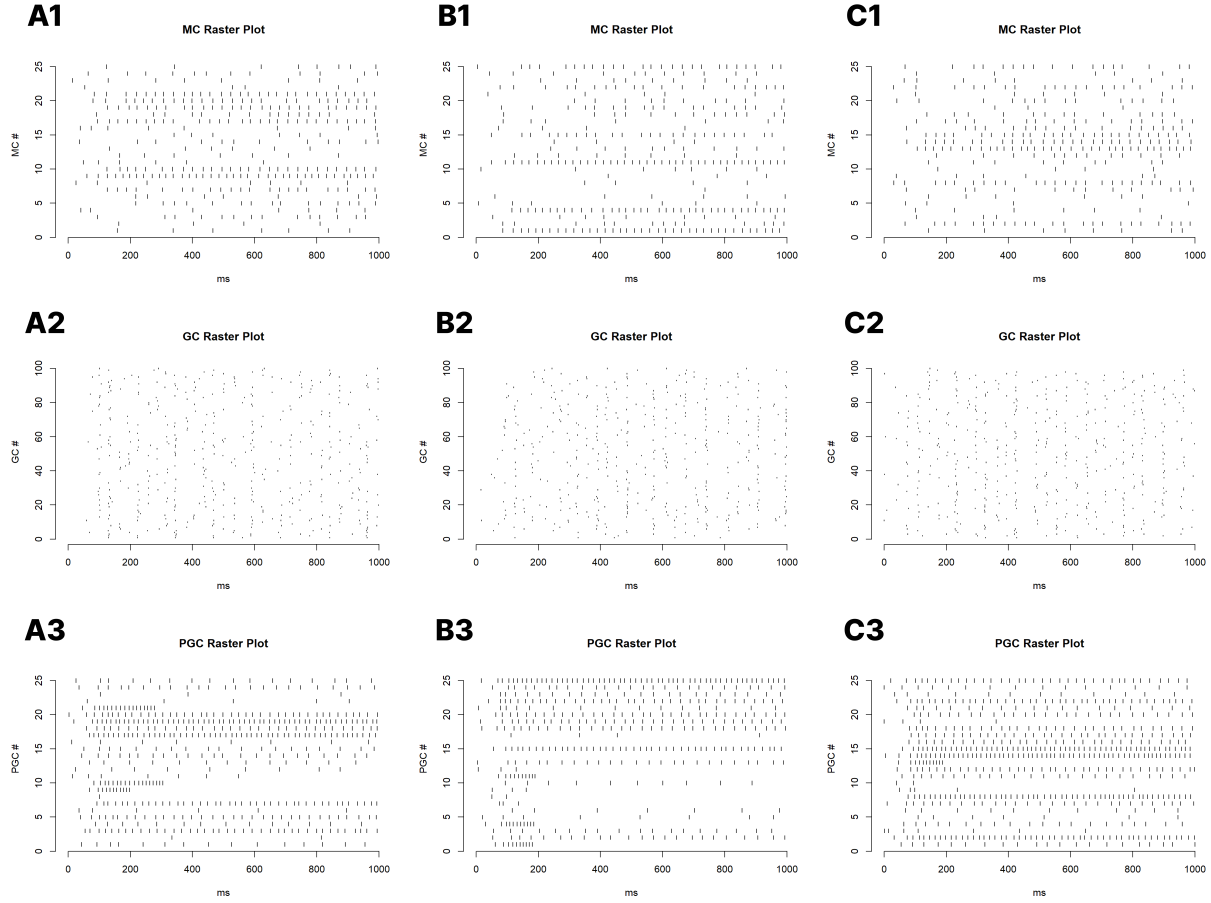


Figure 3.2: Raster plots for MCs, GCs, and PGCs showing similar firing patterns across all seeds. Notably, firing patterns are not bound to specific cells and vary across simulation seeds. **A:** Raster plots for the default seed **B:** Raster plots for random seed 1. **C:** Raster plots for random seed 2.

Figure 3.1 presents the simulated local field potentials (sLFPs), autocorrelation functions (ACFs), and power spectral densities (PSDs) of the simulation results from the default seed (A) and two other seeds (B and C). It can be observed that the plots from the two random seeds in Figure 3.1B and 3.1C are fairly consistent with the default seed's plots shown in Figure 3.1A. Slight deviations in each plot are likely attributable to the random initialization of each simulation.

Across the three simulations, there are notable consistencies in the sLFP morphology, displaying consistent oscillatory activity. This is supported by the periodic nature of the

ACFs, with decaying amplitude as lags increase, suggesting a stable oscillatory behavior in the network’s activity over time. The prominent peak observed in the PSDs corresponds to a dominant gamma frequency component at around 30 Hz, which is consistent with the expected behavior of the olfactory bulb. The consistency of these properties across different seeds suggests that the network is stable and less susceptible to the effects of random initialization. While random seeding introduces variability at the microscopic level, the macroscopic properties, such as dominant frequency and rhythmicity, are robust features of the model.

Figure 3.2 presents the raster plots of the MCs, GCs, and PGCs across all simulation seeds. A, B, and C denote the raster plots for the default seed, random seed 1, and random seed 2 respectively. The raster plots for each respective cell across each simulation demonstrate similar firing patterns.

The MCs display evenly-spaced spikes within a specific cell, with some cells firing more frequently than others. The GCs display some coherence, with several GCs spiking simultaneously at different times. Noticeable examples of this may be found at around 600ms in A2, 1000ms in B2, and 400ms in C2. The PGCs exhibit patterns similar to the MCs, except that there are some cells that taper off and stop firing altogether as the simulation progresses. It is also worth noting that firing patterns across each cell type are not bound to specific cells and vary across seeds.

## 3.2 Sparse and Full Connectivity

The results in the following section were obtained by modifying the connection probability variable  $P_c$  in the `Parameter.hoc` file. The default simulation has  $P_c$  set to 0.3. A sparse connectivity scenario was created by setting  $P_c$  to 0.1. Similarly, a full connectivity scenario was created by setting  $P_c$  to 1.0. The sparse connectivity scenario ran for approximately 1.5 hours, while the full connectivity scenario ran for around 4 hours.

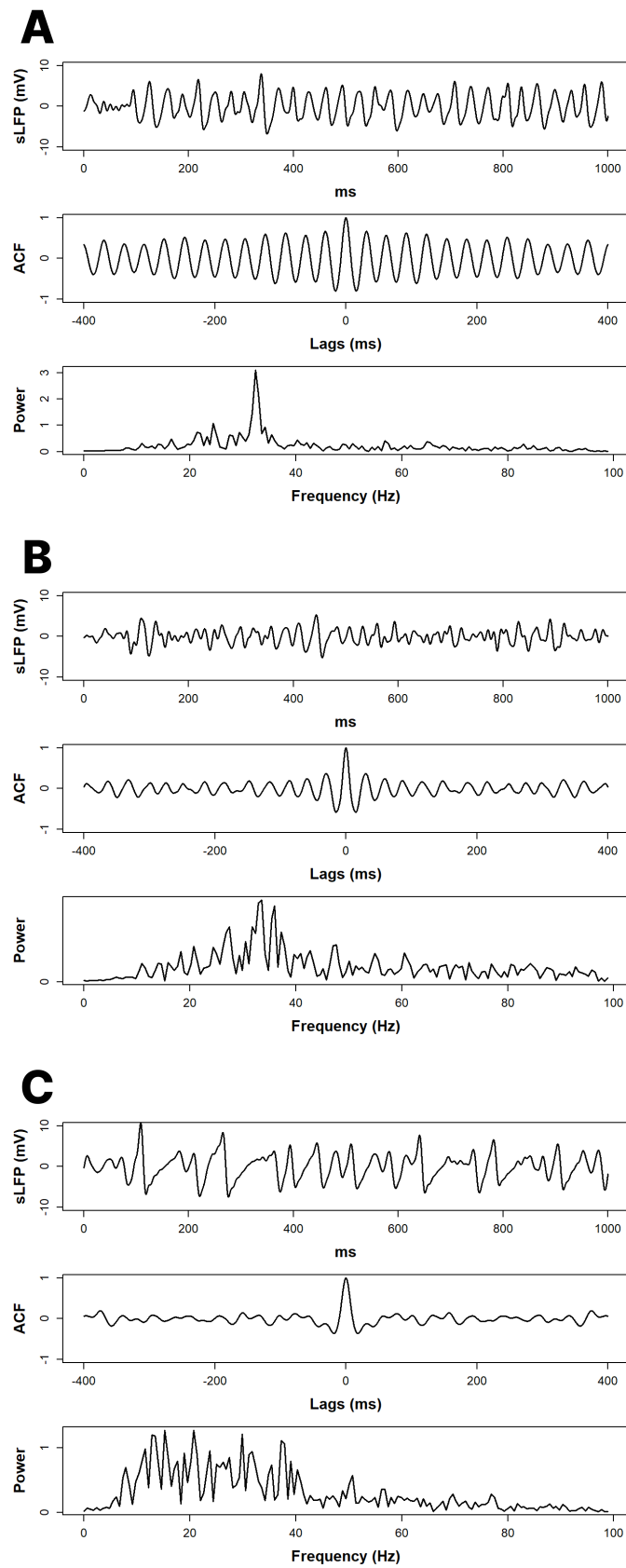


Figure 3.3: LFP, ACF, and PSD plots show different rhythmic behavior across the default, sparse, and full network connectivity. **A:** LFP, ACF, and PSD plots for the default seed LFP. **B:** LFP, ACF, and PSD plots for sparse connectivity. **C:** LFP, ACF, and PSD plots for full connectivity.

Figure 3.3 presents the simulated local field potentials (sLFPs), autocorrelation functions (ACFs), and power spectral densities (PSDs) of the simulations results from the default connectivity (A), sparse connectivity simulation (B), and the full connectivity simulation (C). Unlike the previous modification (Figure 3.1), where the default connectivity was compared with other random seeds, modifying the connectivity produces noticeable differences in the sLFP oscillatory behavior.

Compared to the default connectivity scenario (A), the sparse connectivity scenario (B) presented with less regular oscillations, indicative of diminished synchronous activity, which is a plausible outcome of reduced synaptic connections. The full connectivity scenario (C) displayed highly irregular sLFP oscillations with variable amplitude, pointing to the potential for over-saturation in synaptic connections leading to erratic network behavior.

Supporting the sLFP findings, the ACF of the sparse network (B) demonstrated a notable decline in rhythmicity, with a noticeable peak at zero lag but rapidly decaying correlations, signifying less predictability in neuronal firing. In contrast, the ACF from the fully connected network (C) revealed a pronounced peak at zero lag with an absence of any sustained rhythmic pattern, suggestive of a rapid degradation of periodicity in the network’s signaling.

The PSD of the default network (A) displays a distinct peak within the gamma frequency band at 30 Hz. The sparse network (B) also presents a peak at around 30 Hz. However, in contrast to the default network, the sparse network presented a PSD with multiple peaks of lower amplitudes, with more activity around the 10-30 Hz range. The fully connected network (C) depicts even more activity in the 10-40 Hz range compared to the sparse network (B). Altogether, these results may suggest that a network with the default connectivity is more conducive to the generation of gamma rhythms.

Figure 3.4 presents the raster plots of the MCs, GCs, and PGCs of the default network (A), the sparse network (B), and the fully-connected network (C). The MC raster plots (A1, B1, C1) exhibit notable differences in firing patterns across connectivity conditions. In comparison to the default network configuration (A1), the sparse connectivity (B1) exhibits increased firing rates across MCs. In contrast, full connectivity (C1) exhibits decreased firing rates across all MCs.

There are also notable differences in network activity in the GC raster plots (A2, B2, C2). Compared to the default network (A2), the sparse network (B2) exhibits correspondingly sparse GC activity. On the other hand, the fully connected network (C2) exhibits coherent and synchronous bursts of firing activity. This is apparent in the seemingly vertical lines formed at various times in the GC raster plot (C2). The times that these vertical lines appear are at approximately 115ms, 271ms, 400ms, 517ms, 646ms, 787ms, and 909ms.

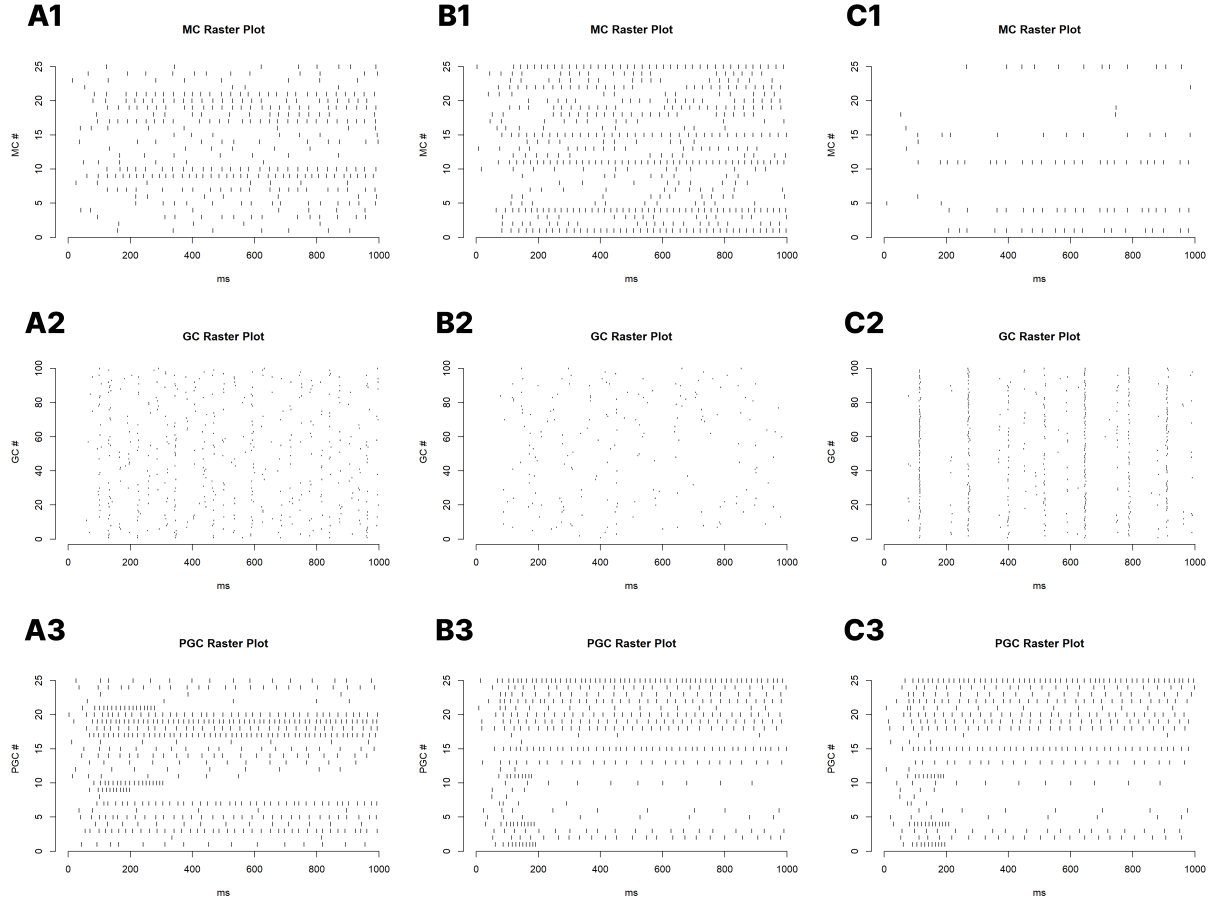


Figure 3.4: Raster plots for MCs and GCs show different firing patterns across connectivity schemes exhibiting their excitatory-inhibitory relationship. Meanwhile, PGCs maintain similar firing patterns. **A:** Raster plots for the default simulation **B:** Raster plots for sparse connectivity. **C:** Raster plots for full connectivity.

The difference between these times are 156ms, 129ms, 117ms, 129ms, 141ms, and 122ms. Based on these numbers, the synchronous firing of GCs that these vertical lines represent occurs, on average, approximately every 132.33 ms.

The PGC raster plots (A3, B3, C3) display consistent behavior across the different network configurations. These exhibit the same behavior as in the random seed modification (Figure 3.2) with some cells firing at constant rates and other cells tapering off as the simulation progresses. This may indicate that PGC behavior is independent of network



configuration and robust across random seeds.

The differences in MC and GC activity are likely due to the varying excitatory-inhibitory relationship of MCs and GCs. Fewer MC to GC connections lead to increased MC activity (B1) and reduced GC activity (B2) due to reduced inhibitory action of GCs. On the other hand, as more MCs are connected to GCs, the decreased MC activity (C1) highlights the increased inhibitory GC activity (C2). It is interesting to note that this strengthened excitatory-inhibitory relationship arising from increased MC to GC connections corresponds to increased synchrony and coherence as seen in C2.

# Chapter 4

## Conclusion

In this paper, we discussed a set of tools and a workflow for replicating and modifying a spiking NEURON (NEURON 2024) network simulation. Chapter 1 gave an overview of essential concepts underlying computational neuroscience including spike trains and neuron modeling. Chapter 1 also discussed the biology, modeling, and simulation of the olfactory bulb based on a study by Li and Cleland (2017). Chapter 2 described the technical process of replicating the primary results of the olfactory bulb study (Li and Cleland 2017). This included downloading the model, running the simulation in Python, and reproducing plots in R. Chapter 3 discussed modifications and results to the olfactory bulb model. This included trying different initial seed values and modifying the network connectivity. The results showed that network behavior is consistent across different seeds and that mitral cell and granule cell activity is dependent on network connectivity.

Overall, we have fulfilled the objective of exploring and documenting a workflow for replicating and extending simulations created for NEURON (NEURON 2024). Computational tools such as NEURON itself along with Python were used to run simulations, while R was used as an analytical tool to create informative visualizations such as ACF, PSD, and raster plots. These tools were used to reproduce and further extend an existing olfactory bulb model. Utilizing these tools, further investigations may be performed on the model or used as a workflow for other models available on the ModelDB database (McDougal et al. 2017).

# Bibliography

- Baker, Monya. 2016. “Reproducibility crisis.” *nature* 533 (26): 353–66.
- Buehlmann, Andres, and Gustavo Deco. 2008. “The Neuronal Basis of Attention: Rate versus Synchronization Modulation.” *Journal of Neuroscience* 28 (30): 7679–7686. ISSN: 0270-6474. <https://doi.org/10.1523/JNEUROSCI.5640-07.2008>. eprint: <https://www.jneurosci.org/content/28/30/7679.full.pdf>. <https://www.jneurosci.org/content/28/30/7679>.
- Calvin, W. H., and C. F. Stevens. 1967. “Synaptic Noise as a Source of Variability in the Interval between Action Potentials.” *Science* 155 (3764): 842–844. <https://doi.org/10.1126/science.155.3764.842>. eprint: <https://www.science.org/doi/pdf/10.1126/science.155.3764.842>. <https://www.science.org/doi/abs/10.1126/science.155.3764.842>.
- . 1968. “Synaptic noise and other sources of randomness in motoneuron interspike intervals.” PMID: 5709873, *Journal of Neurophysiology* 31 (4): 574–587. <https://doi.org/10.1152/jn.1968.31.4.574>. eprint: <https://doi.org/10.1152/jn.1968.31.4.574>. <https://doi.org/10.1152/jn.1968.31.4.574>.
- Dayan, P., and L. F. Abbott. 2001. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press.
- Einevoll, Gaute T., Christoph Kayser, Nikos K. Logothetis, and Stefano Panzeri. 2013. “Modelling and analysis of local field potentials for studying the function of cortical circuits.” *Nature Reviews Neuroscience* 14, no. 11 (November): 770–785. ISSN: 1471-0048. <https://doi.org/10.1038/nrn3599>. <https://doi.org/10.1038/nrn3599>.
- Gerstner, Wulfram, Werner M. Kistler, Richard Naud, and Liam Paninski. 2014. *Neuronal Dynamics: From single neurons to networks and models of cognition and beyond*. Cambridge: Cambridge University Press.

- Hagen, Espen, David Dahmen, Maria L. Stavrinou, Henrik Lindén, Tom Tetzlaff, Sacha J. van Albada, Sonja Grün, Markus Diesmann, and Gaute T. Einevoll. 2016. “Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks.” *Cerebral Cortex* 26, no. 12 (December): 4461–4496. ISSN: 1047-3211. <https://doi.org/10.1093/cercor/bhw237>. eprint: <https://academic.oup.com/cercor/article-pdf/26/12/4461/8657457/bhw237.pdf>. <https://doi.org/10.1093/cercor/bhw237>.
- Hebb, D. O. 1949. *The Organization of Behavior: A Neuropsychological Approach*. John Wiley & Sons.
- Hodgkin, Alan L, and Andrew F Huxley. 1952. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” *The Journal of physiology* 117 (4): 500–544.
- Hutson, Matthew. 2018. *Artificial intelligence faces reproducibility crisis*.
- Kass, Robert E., Shun-Ichi Amari, Kensuke Arai, Emery N. Brown, Casey O. Diekman, Markus Diesmann, Brent Doiron, et al. 2018. “Computational Neuroscience: Mathematical and Statistical Perspectives” [in eng]. PMC6454918, *Annual Review of Statistics and Its Application* 5 (March): 183–214. ISSN: 2326-8298, 2326-831X. <https://doi.org/10.1146/annurev-statistics-041715-033733>.
- Katz, B. 1966. *Nerve, Muscle and Synapse*. McGraw-Hill.
- Kay, Leslie M., and Philip Lazzara. 2010. “How Global Are Olfactory Bulb Oscillations?” PMID: 20660428, *Journal of Neurophysiology* 104 (3): 1768–1773. <https://doi.org/10.1152/jn.00478.2010>. eprint: <https://doi.org/10.1152/jn.00478.2010>. <https://doi.org/10.1152/jn.00478.2010>.
- Lapique, L. 1907. “Recherches quantitatives sur l’excitation électrique des nerfs traitée comme une polarisation.” *J. Physiol. Pathol. Gen* 9:620–635.
- Li, G, and TA Cleland. 2013. “A Two-Layer Biophysical Model of Cholinergic Neuromodulation in Olfactory Bulb.” *Journal of Neuroscience* 33 (7): 3037–3058. ISSN: 0270-6474. <https://doi.org/10.1523/JNEUROSCI.2831-12.2013>. eprint: <https://www.jneurosci.org/content/33/7/3037.full.pdf>. <https://www.jneurosci.org/content/33/7/3037>.
- . 2017. “A coupled-oscillator model of olfactory bulb gamma oscillations.” *PLoS Comput Biol* 13, no. 11 (November 15, 2017): e1005760. <https://doi.org/10.1371/journal.pcbi.1005760>.
- McCulloch, W. S., and W. Pitts. 1943. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” *Bulletin of Mathematical Biophysics* 5:115–133.

- McDougal, Robert A., Thomas M. Morse, Ted Carnevale, Luis Marengo, Rixin Wang, Michele Migliore, Perry L. Miller, Gordon M. Shepherd, and Michael L. Hines. 2017. “Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience.” *Journal of Computational Neuroscience* 42 (1): 1–10.
- Moore, G. P., D. H. Perkel, and J. P. Segundo. 1966. “Statistical Analysis and Functional Interpretation of Neuronal Spike Data.” PMID: 5323104, *Annual Review of Physiology* 28 (1): 493–522. <https://doi.org/10.1146/annurev.ph.28.030166.002425>. eprint: <https://doi.org/10.1146/annurev.ph.28.030166.002425>. <https://doi.org/10.1146/annurev.ph.28.030166.002425>.
- Nagayama, Shin, Ryota Homma, and Fumiaki Imamura. 2014. “Neuronal organization of olfactory bulb circuits.” *Frontiers in Neural Circuits* 8. ISSN: 1662-5110. <https://doi.org/10.3389/fncir.2014.00098>. <https://www.frontiersin.org/articles/10.3389/fncir.2014.00098>.
- NEURON. 2024. *What is NEURON?* [https://www.neuron.yale.edu/neuron/what\\_is\\_neuron](https://www.neuron.yale.edu/neuron/what_is_neuron). Accessed: 2024-03-11.
- Panzeri, Stefano, Riccardo Senatore, Marcelo A Montemurro, and Rasmus S Petersen. 2007. “Correcting for the sampling bias problem in spike train information measures.” *Journal of neurophysiology* 98 (3): 1064–1072.
- Peng, Roger. 2015. “The reproducibility crisis in science: A statistical counterattack.” *Significance* 12 (3): 30–32.
- Rolls, ET, and G Deco. 2010. *The Noisy Brain: Stochastic Dynamics as a Principle of Brain Function*. Oxford University Press.
- Rosenblatt, F. 1958. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” PMID: 13602029, *Psychol Rev* 65 (6): 386–408. <https://doi.org/10.1037/h0042519>.
- Theunissen, F., and J. P. Miller. 1995. “Temporal encoding in nervous systems: A rigorous definition.” *Journal of Computational Neuroscience* 2:149–162. <https://doi.org/10.1007/BF00961885>.

# Appendix

# Appendix A

## Running Neuron Simulation in Python

To run the simulation using Python, first open the **OBGAMMA** folder in Visual Studio Code. This can be done by selecting the *"Open Folder..."* option under the *File* tab in the upper-left corner of Visual Studio Code as shown in Figure A.1. Afterwards, start a new terminal by selecting the *"New Terminal"* option under the *Terminal* tab in the upper-left corner of Visual Studio Code as shown in Figure A.2. A terminal should appear at the bottom of the Visual Studio Code window similar to what is shown in Figure A.3.

Next, we will create a Python virtual environment in the newly opened terminal by typing the command `python3 -m venv env` (without the backticks). This command makes use of python's `venv` module to create an environment folder named `env` within the **OBGAMMA** folder. Verify that the `env` folder now exists in the folder to determine if the command was successful. Afterwards, we will activate the environment by typing the command `source env/bin/activate`. The current line of the terminal should now begin with the text `env` similar to what is shown in Figure A.4. This indicates that the environment has been activated.

After creating and activating the environment, we will now install the **NEURON** module for Python. In the terminal, type the command `pip3 install NEURON`. This will install the **NEURON** module. Something similar to what is shown in Figure A.5 should be seen in the terminal. Afterward, we will also install the interactive Python kernel module since we will make use of Python notebooks later on in the process. Install this by typing `pip3 install ipykernel` in the terminal. The terminal should show something similar to what was shown in Figure A.5 after installing the **NEURON** module.

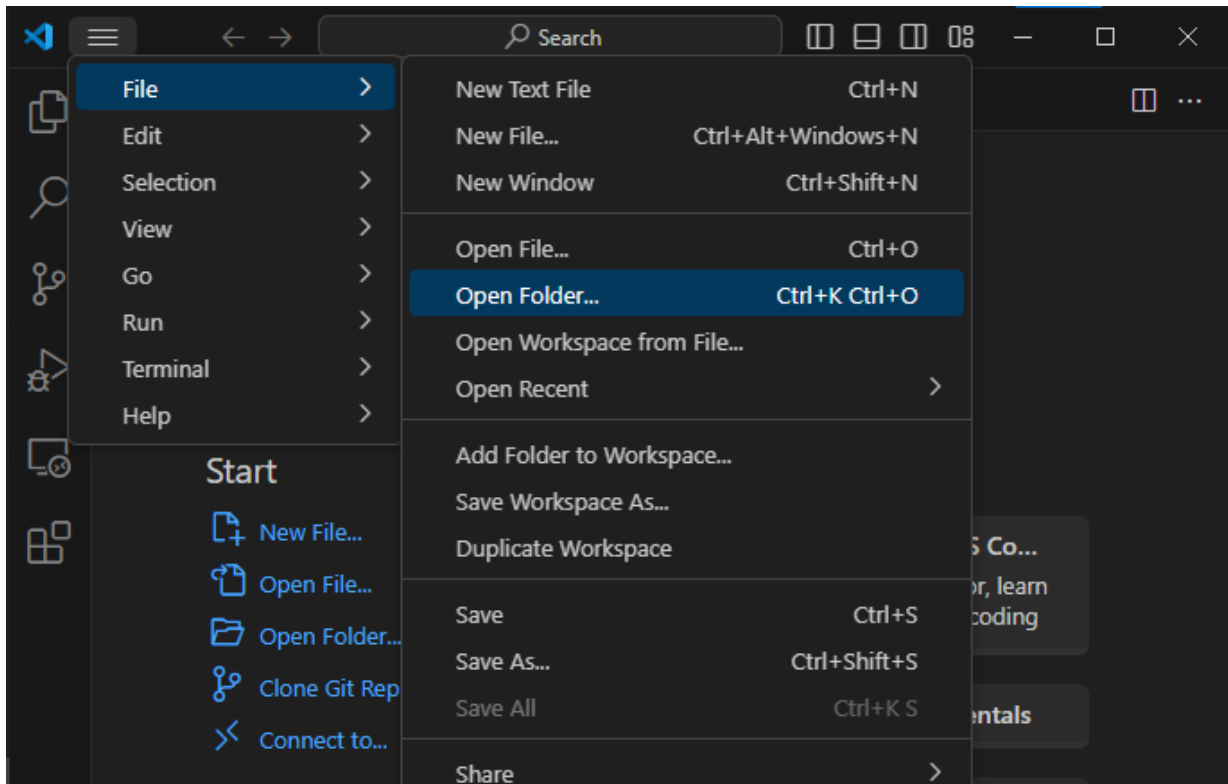


Figure A.1: Open Folder Option in VS Code



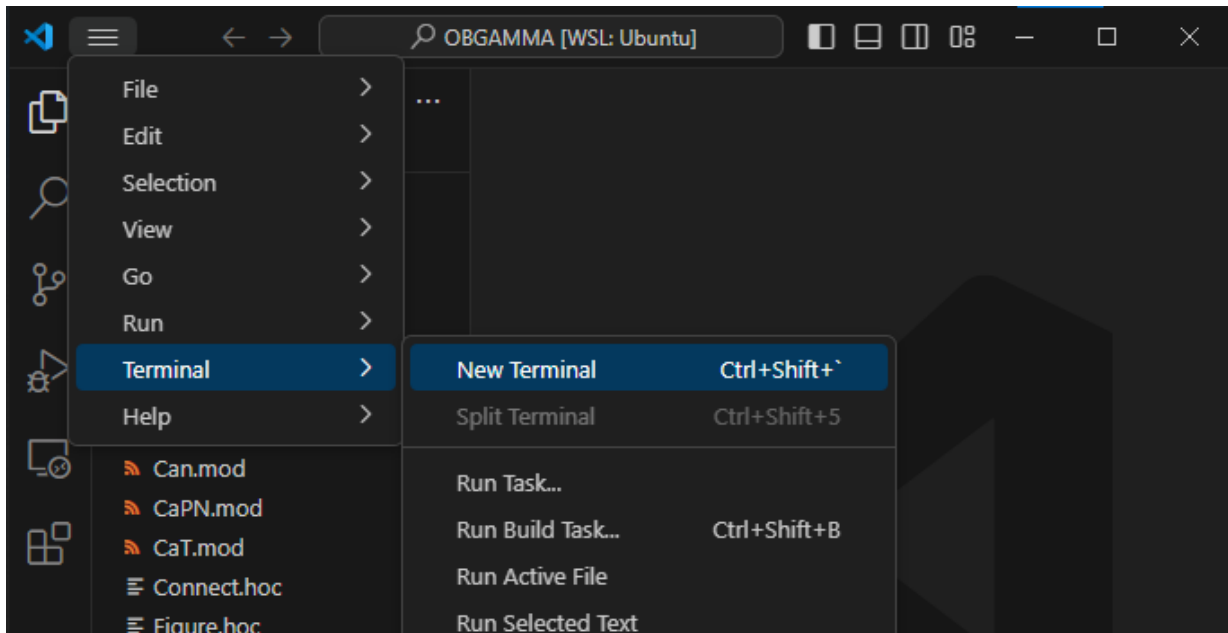


Figure A.2: New Terminal Option in VS Code

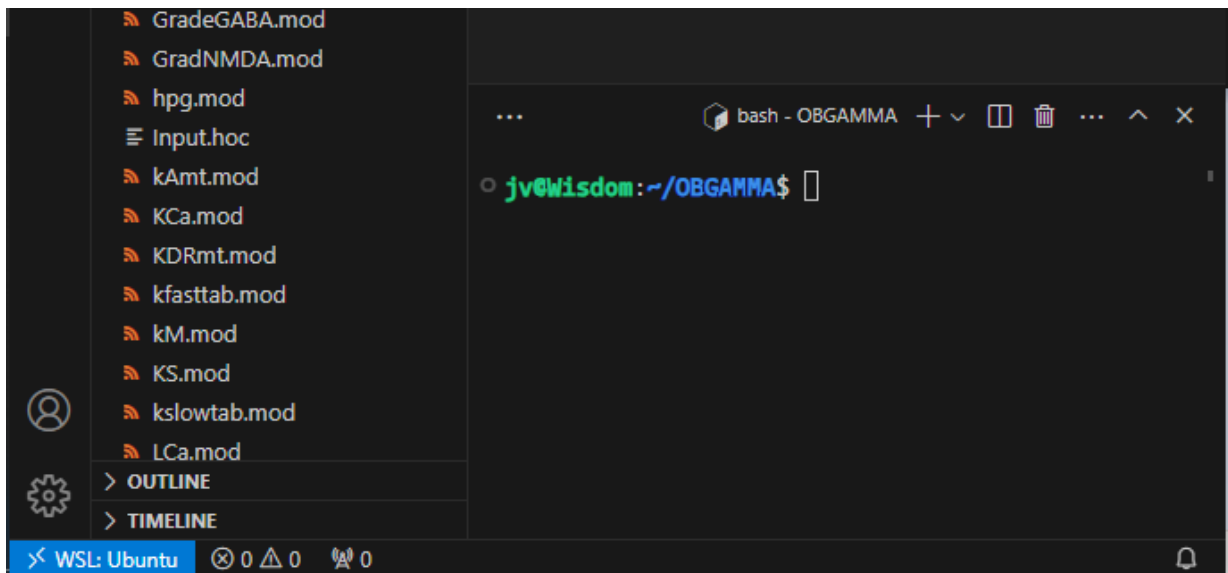
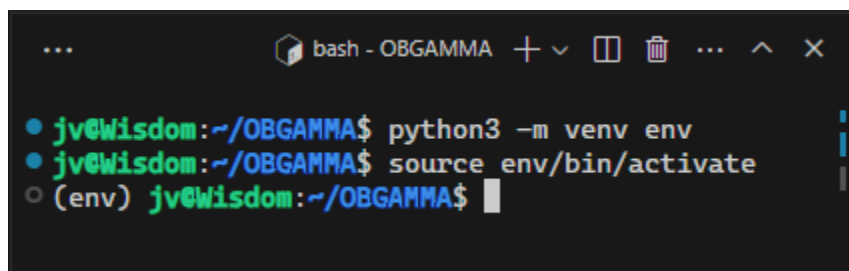
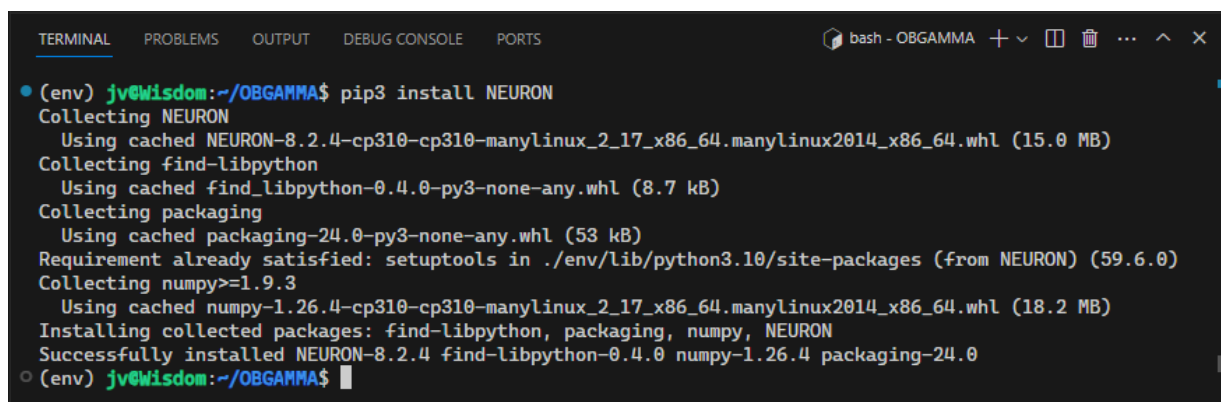


Figure A.3: Terminal in VS Code




```
... bash - OBGAMMA + v [icon] [icon] ... ^ x
• jv@Wisdom:~/OBGAMMA$ python3 -m venv env
• jv@Wisdom:~/OBGAMMA$ source env/bin/activate
○ (env) jv@Wisdom:~/OBGAMMA$
```

Figure A.4: Activated Environment in VS Code Terminal



```
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS bash - OBGAMMA + v [icon] [icon] ... ^ x
• (env) jv@Wisdom:~/OBGAMMA$ pip3 install NEURON
Collecting NEURON
  Using cached NEURON-8.2.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (15.0 MB)
Collecting find-libpython
  Using cached find_libpython-0.4.0-py3-none-any.whl (8.7 kB)
Collecting packaging
  Using cached packaging-24.0-py3-none-any.whl (53 kB)
Requirement already satisfied: setuptools in ./env/lib/python3.10/site-packages (from NEURON) (59.6.0)
Collecting numpy>=1.9.3
  Using cached numpy-1.26.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.2 MB)
Installing collected packages: find-libpython, packaging, numpy, NEURON
Successfully installed NEURON-8.2.4 find-libpython-0.4.0 numpy-1.26.4 packaging-24.0
○ (env) jv@Wisdom:~/OBGAMMA$
```

Figure A.5: Installing the NEURON Python Module



```
(env) jv@wisdom:~/OBGAMMA$ nrnivmodl
/home/jv/OBGAMMA
Mod files: "./CaPN.mod" "./CaT.mod" "./Caint.mod" "./Can.mod" "./GradNMDA.mod" "./GradeAMPA.mod" "./Grad
eGABA.mod" "./KCa.mod" "./KDRmt.mod" "./KS.mod" "./LCa.mod" "./NaP.mod" "./Naxn.mod" "./Nicotin.mod" "./
OdorInput.mod" "./SineInput.mod" "./cadecay.mod" "./cadecay2.mod" "./hpg.mod" "./kAmt.mod" "./kM.mod" ".
/kfasttab.mod" "./kslowtab.mod" "./nafast.mod" "./nmdanet.mod"

Creating 'x86_64' directory for .o files.

-> Compiling mod_func.cpp
-> NMODL ../CaPN.mod
-> NMODL ./CaT.mod
```

Figure A.6: Compiling NEURON files

With the NEURON module now installed, we can compile the `.mod` files in the OBGAMMA folder. We can do this by typing the command `nrnivmodl` in the terminal as shown in Figure A.6. This should create a new folder in the OBGAMMA folder named `x86_64`.

After compiling the NEURON files, we will create a Python notebook. We can create a new file by selecting the *"New File..."* option under the File tab as shown in Figure A.7. After selecting that, some options will appear. Select *Jupyter Notebook* with the `.ipynb` extension as shown in Figure A.8. This will open a new Jupyter Python Notebook.

Having created the Python notebook, we must now select which kernel to use. This step will ensure that the notebook makes use of the environment that was created earlier. Near the upper-right corner of Visual Studio Code, we will find the *Select Kernel* option. Clicking it will present a few options. Select the one labeled `env` showing the path `env/bin/python` as shown in Figure A.9. The *Select Kernel* text should now be replaced by the text `env (Python 3.10.12)` as shown in Figure A.10.

With the kernel selected, we may attempt to run the demo simulation included in the model's files. First, we must initialize the model. In the first code cell of the Python notebook, type the following code:

```
from neuron import h
h.load_file("mosinit.hoc")
```

Then, we can execute the code by clicking the triangular play button to the left of the code cell as seen in Figure A.11. If there are no issues, the code cell should output various numbers including the average number of GC inputs per MC and vice versa as shown in Figure A.12.

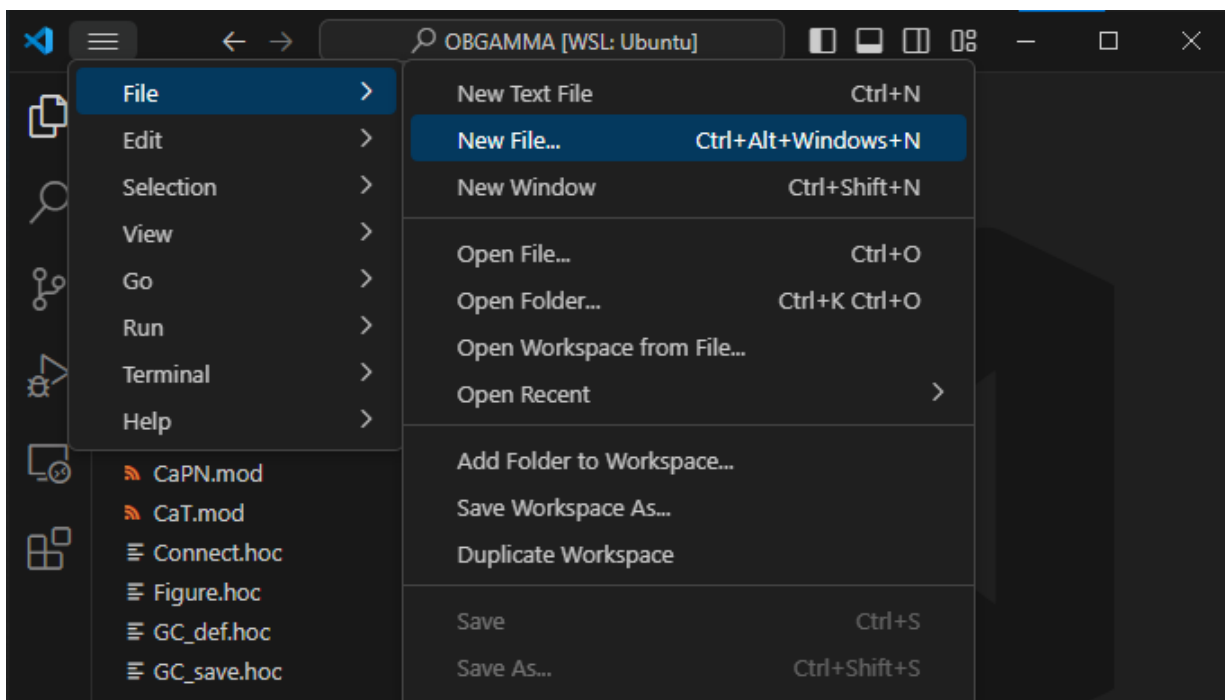


Figure A.7: New File Option in VS Code

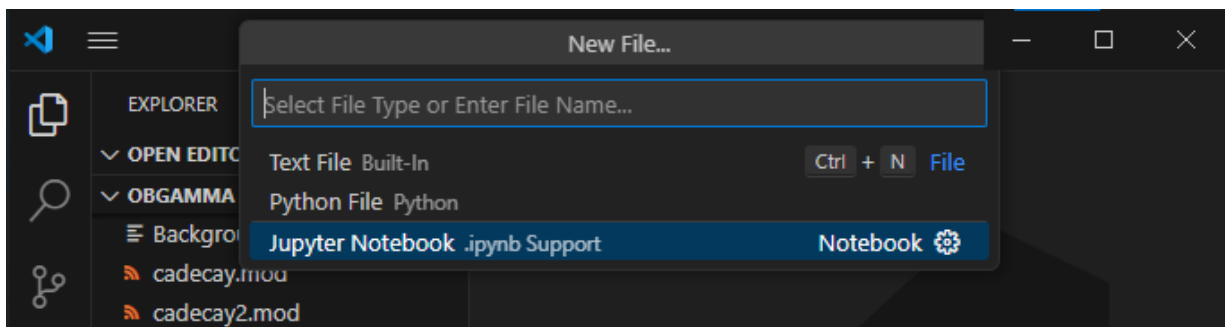


Figure A.8: New Jupyter Notebook Option in VS Code

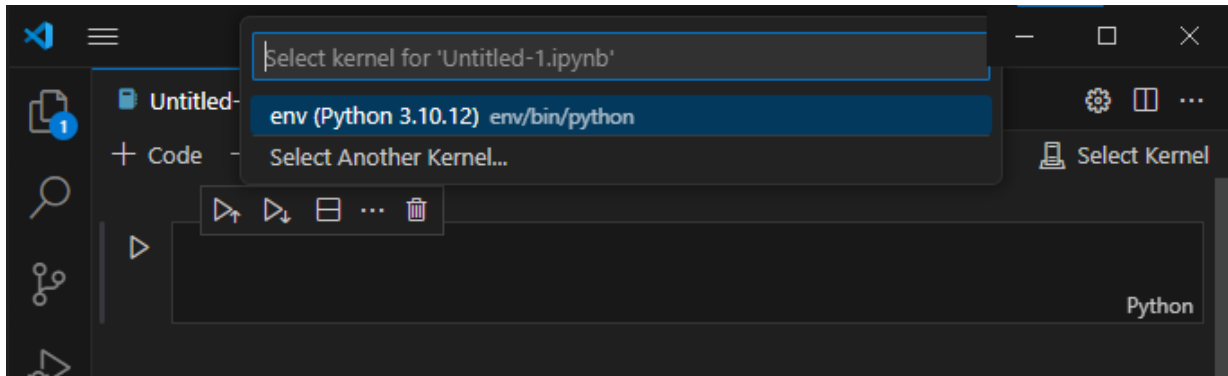


Figure A.9: Selecting the env Kernel in VS Code

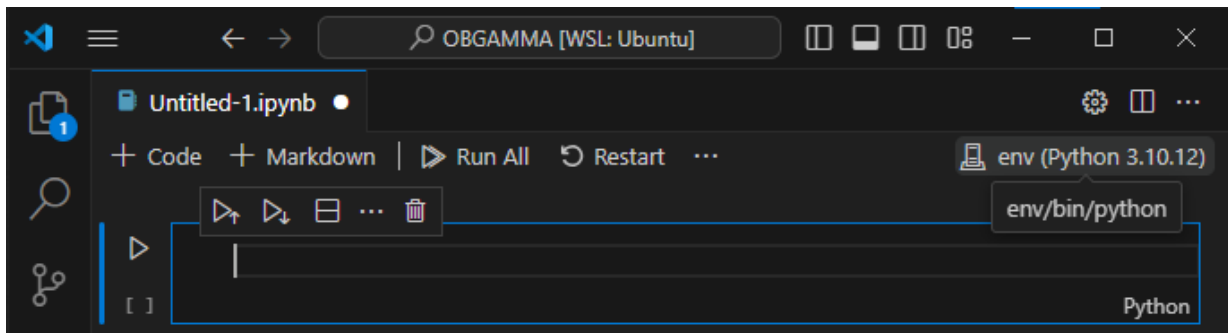


Figure A.10: Select Kernel Text Replaced by env (Python 3.10.12)

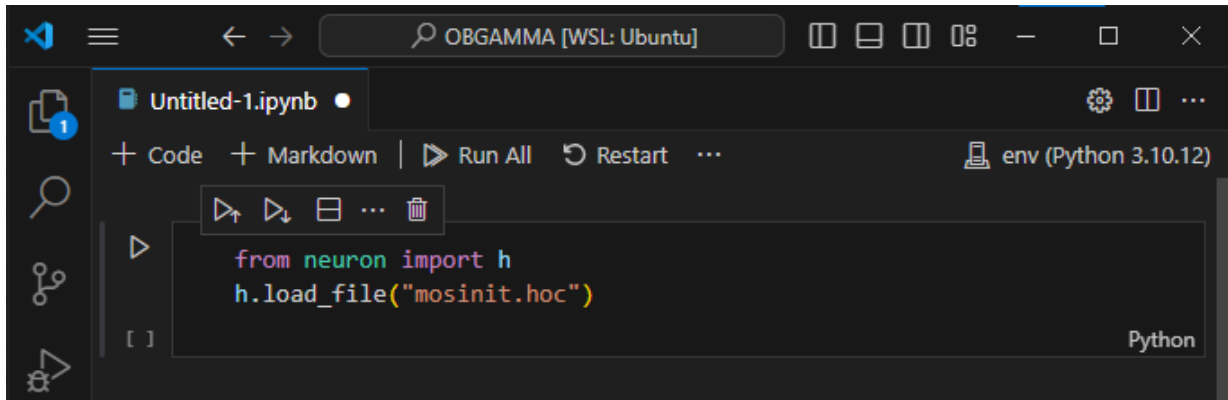
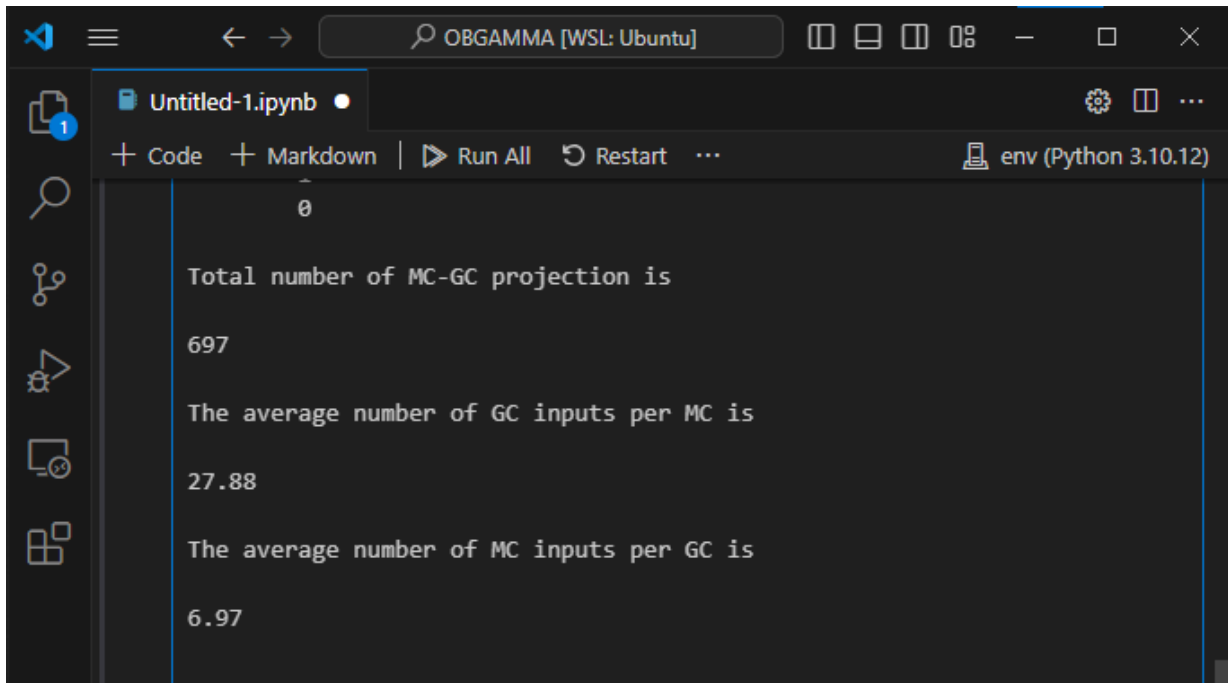


Figure A.11: Code Cell with Execute Button

After successful initialization, create a new code cell. To run the demo simulation, type `h.demo_run()` which will run for a few minutes. If successful, the code block should output the text `data0` as seen in Figure A.13. To further verify, we may check if the `data0` folder under OBGAMMA is populated with various data files with formats including `Gd_i-j`, `Ms_i-j`, `Vgb_i-j`, `Vms_i-j`, etc. If such files do not exist, then the simulation did not run properly.

To run the full simulation, we may instead use the code `h.run()` instead of `h.demo_run()`. The full simulation may take many hours to finish running. When the simulation is completed, we may begin reproducing the plots in R based on the data in the `data0` folder.



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar indicates the environment is 'OBGAMMA [WSL: Ubuntu]'. The notebook file is named 'Untitled-1.ipynb'. The toolbar includes buttons for '+ Code', '+ Markdown', 'Run All', 'Restart', and a menu icon. The environment is set to 'env (Python 3.10.12)'. The code cell contains the following text:

```
0

Total number of MC-GC projection is

697

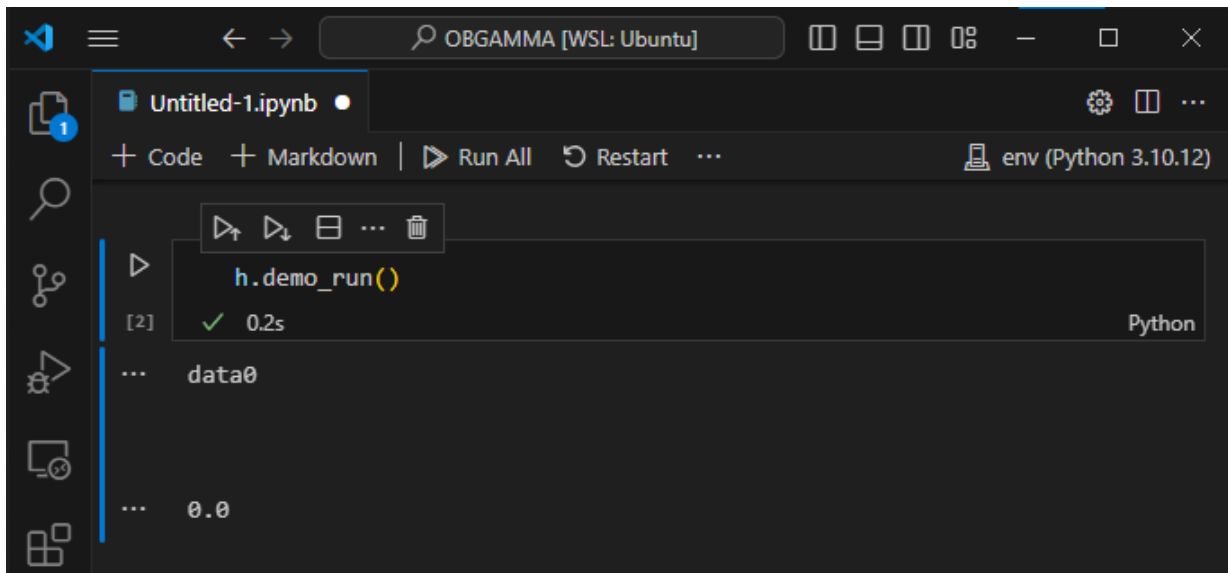
The average number of GC inputs per MC is

27.88

The average number of MC inputs per GC is

6.97
```

Figure A.12: Successful Code Initialization Output



The screenshot shows the same Jupyter Notebook interface. The code cell contains the following text:

```
h.demo_run()
```

The output of the cell is displayed below the code, showing a green checkmark, the execution time '0.2s', and the variable 'data0' with its value '0.0'.

```
[2] ✓ 0.2s Python
... data0
... 0.0
```

Figure A.13: Successful Demo Simulation

# Appendix B

## R Codes for Plot Reproduction

All the code in the following section assumes that the time record has been loaded. This may be done with the following code:

```
1 ### Read the recorded times ###
2 tt <- scan("tt", quiet=TRUE)
```

### B.1 Figure 2A

```
1 Odor <- scan("Odor", quiet=TRUE)
2 names(Odor) <- ifelse((1:25 %% 5) == 0, 1:25, rep(NA, 25))
3
4 type <- "MC"
5 n <- c("MC" = 5, "GC" = 10, "PGC" = 5)[type]
6 prefix <- c("MC" = "Ms", "GC" = "Gs", "PGC" = "Ps")[type]
7 combs <- expand.grid(j=seq(0, n-1), i=seq(0, n-1))
8 spike_files <- sprintf("%s_%d_%d", prefix, combs$i, combs$j)
9 spike_times <- lapply(spike_files,
10                       function(fname) {
11                           tryCatch(
12                               read.table(fname)$V1,
13                               error=function(e) numeric(0))}
14                       )
15 x_values <- sapply(spike_times, function(x) length(x[x >= 2000]))
```



```

16 names(x_values) <- ifelse((1:25 %% 5) == 0, 1:25, rep(NA, 25))
17
18 # par(mfrow=c(2, 1), mai=c(2,1,0.15,0.1))
19 barplot(0dor, col="black",
20         ylab="", yaxt="n", ylim=c(0, 1))
21 axis(1, at=0)
22 axis(2, at=seq(0, 1, 0.5))
23 title(xlab = "Glom #", ylab = "Input (nA)", cex.lab = 1.2, font.lab = 2)
24
25 barplot(x_values, col="black",
26         ylab="", yaxt="n", ylim=c(0, 40))
27 axis(1, at=0)
28 axis(2, at=seq(0, 40, 20))
29 title(xlab = "Glom #", ylab = "Firing rate (Hz)", cex.lab = 1.2, font.lab = 2)

```

## B.2 Figure 2B

```

1 library(signal)
2
3 nextpow2 <- function(i) {
4   for (ii in 0:31) {
5     if (i < 2^ii) {
6       return(ii)
7     }
8   }
9   return(NULL)
10 }
11 # Set parameters
12 FILORDER <- 1000
13 DT <- 0.2 # sampling time in ms
14 Fs <- 1 / DT * 1000 # sampling frequency in Hz
15 Fc <- c(10, 100) # Cut-off frequencies in Hz
16 Wc <- Fc / (Fs / 2) # Normalize the frequencies
17
18 # Extract the desired portion of the signal
19 T1 <- 2000
20 T2 <- 3000

```

```

21 n1 <- T1 / DT + 1
22 n2 <- T2 / DT
23 maxlags <- 2000
24
25 t <- tt[n1:n2]
26 y <- Vam[n1:n2] - mean(Vam[n1:n2]) # Detrend the data by subtracting the mean
27
28 ### Compute sLFP ###
29 # Create the FIR filter and apply it using firlfilt
30 h <- signal::fir1(FILORDER, Wc, type="pass")
31 x <- signal::firlfilt(h, y)
32
33 ### Compute Auto-Correlation ###
34 ac_results <- ccf(x, x, lag.max = maxlags, type="correlation", plot=FALSE)
35 lags <- ac_results$lag
36 auto_correlations <- ac_results$acf
37
38
39 ### Compute Power ###
40 Fmax <- 100
41 L <- length(y)
42 NFFT <- 2^nextpow2(L) # Next power of 2 from length of y
43 pad_y <- c(y, rep(0, NFFT-L))
44 Y <- fft(pad_y)/L;
45 YY <- 2*abs(Y[1:(NFFT/2)])
46
47 f <- (Fs/2) * seq(0, 1, length.out=NFFT/2)
48 m <- Fmax/(0.5*Fs)*(0.5*NFFT);
49 m <- floor(m)
50
51
52 pad_x <- c(x, rep(0, NFFT-L))
53 X <- fft(pad_x)/L;
54 XX <- 2*abs(X[1:(NFFT/2)])
55
56
57 # Plot the data
58 par(mfrow=c(3, 1), cex.axis = 1.2, mai=c(0.6,0.7,0.1,0.1))

```

```

59 plot(t-2000, x, type = "l", lwd = 1.75, ylim = c(-10, 10),
60      xlab = "", xaxt="n",
61      ylab = "", yaxt="n")
62 axis(1, at = seq(0, 1000, by = 200)) # Customize the x-axis ticks as needed
63 axis(2, at = seq(-10, 10, by = 10))   # Customize the y-axis ticks as needed
64 title(xlab = "ms", ylab = "sLFP (mV)", cex.lab = 1.5, font.lab = 2)
65
66 plot(lags*DT, auto_correlations, type = 'l', lwd = 1.75,
67      xlab = "", xaxt="n", xlim=c(-400, 400),
68      ylab = "", yaxt="n", ylim=c(-1, 1))
69 axis(1, at=seq(-400, 400, 200))
70 axis(2, at=seq(-1, 1, 1))
71 title(xlab = "Lags (ms)", ylab = "ACF", cex.lab = 1.5, font.lab = 2)
72
73 plot(f[1:m], XX[1:m], type='l', lwd=1.75, xlab="", xaxt="n", ylab="", yaxt="n")
74 axis(1, at = seq(0, 100, by = 20))
75 axis(2, at = seq(0, 3, by = 1))
76 title(xlab = "Frequency (Hz)", ylab = "Power", cex.lab = 1.5, font.lab = 2)

```

## B.3 Figure 2C

The following code is also reused for D1, E1, and F1.

```

1 plot_pair <- function(cell1, cell2, type) {
2   prefix <- c("MC" = "Vms", "GC" = "Vgb", "PGC" = "Vpb")[type]
3   vs1_name <- sprintf("%s_%d_%d", prefix, cell1[1], cell1[2])
4   vs2_name <- sprintf("%s_%d_%d", prefix, cell2[1], cell2[2])
5   vs1 <- read.table(vs1_name)$V1
6   vs2 <- read.table(vs2_name)$V1
7   # Assuming tt has been loaded
8   lwd <- 2
9   plot(tt, vs1, type="l", xlim=c(2000, 3000), ylim=c(-80, 40), col="blue", lwd=lwd)
10  lines(tt, vs2, type="l", xlim=c(2000, 3000), ylim=c(-80, 40), col="red", lwd=lwd)
11  legend(x = "topright", # Position
12        legend = c(
13          sprintf("%s [%d] [%d]", type, cell1[1], cell1[2]),
14          sprintf("%s [%d] [%d]", type, cell2[1], cell2[2])

```

```

15         ), # Legend texts
16         lty = c(1, 1), # Line types
17         col = c("blue", "red"), # Line colors
18         lwd = lwd)
19     }
20
21     compare_pairs <- function(pair1, pair2, type) {
22         par(mfrow=c(2, 1), mai=c(0.4,0.4,0.4,0.4))
23         plot_pair(pair1[[1]], pair1[[2]], type)
24         title(type)
25         plot_pair(pair2[[1]], pair2[[2]], type)
26     }
27

```

To create the plot for C, the `compare_pairs` function from the previous code was called as follows:

```

1  compare_pairs(
2      list(c(0, 1), c(4, 1)),
3      list(c(3, 4), c(2, 4)),
4      type="MC"
5  )

```

## B.4 Figures 2D1, 2E1, 2F1

```

1  ### Figure 2D1 ###
2  compare_pairs(
3      list(c(3, 2), c(4, 3)),
4      list(c(1, 1), c(1, 4)),
5      type="MC"
6  )

1  ### Figure 2E1 ###
2  compare_pairs(
3      list(c(0, 1), c(3, 2)),
4      list(c(2, 2), c(6, 1)),
5      type="GC"
6  )

```

```

1  ### Figure 2F1 ###
2  compare_pairs(
3      list(c(0, 1), c(4, 1)),
4      list(c(3, 4), c(2, 4)),
5      type="PGC"
6  )

```

## B.5 Figures 2D2, 2E2, 2F2

The following code defines a custom function for creating a raster plot

```

1  plot_raster <- function(type) {
2      n <- c("MC" = 5, "GC" = 10, "PGC" = 5)[type]
3      prefix <- c("MC" = "Ms", "GC" = "Gs", "PGC" = "Ps")[type]
4      combs <- expand.grid(j=seq(0, n-1), i=seq(0, n-1))
5      spike_files <- sprintf("%s_%d_%d", prefix, combs$i, combs$j)
6      spike_times <- lapply(spike_files,
7                           function(fname) {
8                               tryCatch(
9                                   read.table(fname)$V1,
10                                   error=function(e) numeric(0))}
11                           )
12      x <- lapply(spike_times, function(x) x[x >= 2000] - 2000)
13      xlab <- "ms"
14      ylab <- sprintf("%s #", type)
15      main <- sprintf("%s Raster Plot", type)
16      xlim <- c(0, 1000)
17      nbTrains <- length(x)
18      acquisitionDuration <- max(xlim)
19      plot(c(0, acquisitionDuration), c(0, nbTrains + 1), type = "n",
20           xlab = xlab, ylab = ylab,
21           xlim = xlim, ylim = c(1, nbTrains + 1), bty = "n", main=main)
22      invisible(sapply(1:nbTrains,
23                      function(idx) {
24                          if (length(x[[idx]]) > 0)
25                              rect(x[[idx]] - 0.000001, idx - 0.3,
26                                  x[[idx]] + 0.000001, idx+0.3)

```

```

27         )))
28     }

1   ### Figure 2D2 ###
2   plot_raster("MC")

1   ### Figure 2E2 ###
2   plot_raster("GC")

1   ### Figure 2F2 ###
2   plot_raster("PGC")

```

## B.6 Figures 2D3, 2E3, 2F3

```

1   plot_spikes <- function(type) {
2       nextpow2 <- function(i) {
3           for (ii in 0:31) {
4               if (i < 2^ii) {
5                   return(ii)
6               }
7           }
8           return(NULL)
9       }
10
11      n <- c("MC" = 5, "GC" = 10, "PGC" = 5)[type]
12      prefix <- c("MC" = "Ms", "GC" = "Gs", "PGC" = "Ps")[type]
13      ymax <- c("MC" = 10, "GC" = 20, "PGC" = 10)[type]
14
15      combs <- expand.grid(j=seq(0, n-1), i=seq(0, n-1))
16      spike_files <- sprintf("%s_%d_%d", prefix, combs$i, combs$j)
17      spike_times <- lapply(spike_files,
18                           function(fname) {
19                               tryCatch(
20                                   read.table(fname)$V1,
21                                   error=function(e) numeric(0))}
22                           )

```

```

23 x <- spike_times
24 flat_x <- Reduce(function(ls1, ls2) c(ls1, ls2), x)
25 filt_x <- flat_x[(2000 <= flat_x) & (flat_x <= 3000)] - 2000
26 hist_values <- hist(filt_x, breaks=seq(0, 1000, 5), plot=FALSE)
27 middle <- hist_values$mids
28 counts <- hist_values$counts
29
30
31 par(mfrow=c(2, 1), mai=c(0.9, 0.8, 0.3, 0.1))
32 # par(mar=c(3, 4, 5, 2))
33 plot(middle, counts, type='l', lwd=1.75,
34       xlab="", xaxt="n", xlim=c(0, 1000),
35       ylab="", yaxt="n", ylim=c(0, ymax))
36 axis(1, at=seq(0, 1000, 200))
37 axis(2, at=seq(0, ymax, ymax/2))
38 title(main=sprintf("%s Spikes", type),
39       xlab = "ms", ylab = "Spikes",
40       cex.lab = 1.2, font.lab = 2)
41
42
43 y <- counts - mean(counts)
44 L <- length(y)
45
46 NFFT <- 2^nextpow2(L)      # Next power of 2 from length of y
47 pad_y <- c(y, rep(0, NFFT-L))
48 Y <- fft(pad_y)/L
49 YY <- 2*abs(Y[1:(NFFT/2)])
50
51 # Plot the data
52
53 f <- seq(0, 100, length.out=NFFT/2)
54 # par(mar=c(5, 4, 0, 2))
55 plot(f, YY, type='l',
56       xlim=c(0, 100), ylim=c(0, 3),
57       lwd=1.75,
58       xlab="", xaxt="n",
59       ylab="", yaxt="n")
60 axis(1, at = seq(0, 100, by = 20))

```

```

61     axis(2, at = seq(0, 3, by = 1))
62     title(xlab = "Frequency (Hz)", ylab = "Power", cex.lab = 1.2, font.lab = 2)
63 }

1  ### Figure 2D3 ###
2  plot_spikes("MC")

1  ### Figure 2E3 ###
2  plot_spikes("GC")

1  ### Figure 2F3 ###
2  plot_spikes("PGC")

```