Thesis Dissertation

# OPERATIONAL MODE AND INFORMATION REPRESENTATION IN SIMPLE CELLS OF THE CAT PRIMARY VISUAL CORTEX

**Georgios Hadjiantonis**

**UNIVERSITY OF CYPRUS**

**COMPUTER SCIENCE DEPARTMENT**

December 2020

**UNIVERSITY OF CYPRUS**

**COMPUTER SCIENCE DEPARTMENT**

# Operational mode and information representation in simple cells of the cat primary visual cortex

**Georgios Hadjiantonis**

Supervisor

Dr. Chris Christodoulou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

December 2020

# Acknowledgments

I would like to express my gratitude to my supervisor, Professor Chris Christodoulou for his continued support and guidance throughout this thesis and for introducing me to the field of Computational Neuroscience.

I would like to thank our collaborator, Professor Guido Bugmann, for his invaluable ideas and comments, that shaped the course of this thesis.

I would also like to thank Professor Matteo Carandini for kindly providing us the experimental data, without which this work would not have been possible.

Finally, I am grateful to my family for their deepest support in my education and choices.

# Abstract

This thesis aims to gain a better understanding of the operational mode of neurons. More specifically, we focus on simple cells of the cat primary visual cortex. First, we apply an existing metric to determine single neurons' operational mode. We then develop two novel methods with which we try to infer the amplitude, duration and synchronicity of the individual presynaptic input from the postsynaptic membrane potential. Understanding the underlying input parameters and especially the level of synchrony is essential in determining the neural operational mode. The methods are tested on simulated data and later applied to the real data.

According to the first proposed method, we examine the distribution of local maxima during the pre-spiking period and local minima during the inhibitory period of the membrane potential. We show that an estimate of the amplitude can be extracted by fitting Gaussian bells on the histograms, suggesting that peaks of the histogram arise on discrete multiples of its magnitude.

For the second method, we measure the variability of a filtered version of the membrane potential, with the filter being tuned to events of a given duration. We then investigate how the variability is affected by the different parameters of presynaptic input, namely amplitude, duration and synchrony of individual events, using simulated data. Each parameter is examined separately through simulations of uniformly or randomly distributed alpha functions.

For a series of isolated peaks of the potential, the variability is highest for filter time window equal to the duration of the peaks. Thus, by increasing the duration of the individual peaks, we note slower increase of variability as a function of filter window size. Moreover, for randomly timed peaks the variability remains relatively at the same level after reaching a certain maximum value. There is also a clear correlation between the height of the peaks and variability. The effects of synchrony require further study.

Based on the observations on simulated data, we provide an estimate of the features of presynaptic excitatory and inhibitory input of the real neuron.

# Contents

# List of Figures

# Chapter 1

# Introduction

## Contents

## 1.1 Introduction

Neurons constitute the fundamental units of the brain and nervous system, propagating signals through the generation of characteristic electrical pulses, also called action potentials or spikes. Neural coding refers to the study of how information is encoded into sequences of spike trains to be processed by neurons and how it can be decoded back. The two most prevailing views are that information is encoded on the average firing rate, termed as rate code [2], or it is conveyed in the precise inter-spike intervals, termed as temporal code [40].

The neural operational mode drew a lot of attention and became the center of a debate among the neuroscience community. Based on the above encoding schemes arise the neural operational modes of temporal integration [8,37,38] and coincidence detection [1,41], which correspond to rate and temporal code, respectively. Among the contributions, that were made in order to determine the operational mode of a neuron is the *normalized pre-spike slope* (NPSS) [27]. The metric provides a measure of the contribution of temporal integration or coincidence detection to the operation of a neuron, by normalizing the slope of the membrane potential preceding each spike between a lower and an upper bound. The lower bound corresponds to the case of temporal integration, where as the upper to a perfect coincidence detection. A modified version of the NPSS was later introduced addressing the limitations of the initial definition, which did not consider any inhibitory input [47].

The major purpose of this thesis is to investigate the behavior of neurons from the cat primary visual cortex and determine their operational mode. A characteristic attribute of mammalian's area V1 of the visual cortex is its orientation selectivity. The behavior of V1 neurons is captured by the broadly accepted and used Hubel & Wiesel model [22]. The model distinguishes V1 cells into two types, namely, simple and complex, according to their response on different stimuli, in the form of rotating black and white stripes. It is based on the principle, that simple cells' receptive field consist of elongated *ON* and *OFF* type subfields. In case of stimulus, where light covers the *ON* area and dark the *OFF* area, the cell is excited, while in the opposite scenario the cell in inhibited. Thus, the rotation angle of the receptive fields determines their preferred orientation. Consecutively, complex cells receive input from simple cells. In contrast with simple cells, the receptive fields of complex cells are not divided into *ON*/*OFF* subfields, as they consist of multiple overlapping simple cells' receptive fields of the same orientation preference.

The study is focused on simple cells. We first apply the empirical procedure of the modified NPSS metric [47] and implement a computational model of the simple cell and the LGN receptive fields, according to the Hubel & Wiesel model with the push-pull mechanism. Additionally, we propose two methods of inferring characteristics of the presynaptic input from the postsynaptic membrane potential, in order to be used in the simulation. In particular, we analyze the distribution of the local extrema and measure the variability of the membrane potential.

## 1.2 Thesis outline

The thesis consists of five chapters. The second chapter makes a brief reference to the background and previous work on neural coding and operational mode. In addition, the basic concepts underlying the primary visual cortex are reported, as well as how such cells can be modeled. The third chapter includes details of the methodology, that was followed and the implementation of the work and a short presentation of the intracellular recordings, on which the study is based. The results are discussed on the fourth chapter. The last chapter is a summary of the conclusions and possible future work, around this thesis.

# Chapter 2

# Background

## Contents

## 2.1 Neural Code

Neurons in the brain process information by converting the receiving continuously varying spatiotemporal pattern/information into a train of spikes. Neural coding refers to the study of how information is representated and processed by neurons. The problem of neural code can be viewed from two opposite perspectives, considered as the opposite sides of the same coin. Neural encoding is the mapping from stimulus to the neuron's response, which involves the observation of how the neuron responds, to different stimuli and the definition/implementation of models, with the aim of understanding and predicting responses to other occasions. Neural decoding is the reverse mapping of neuron's response to external stimulus.

There has been a long-standing debate among the neuroscience community, about the mechanisms used by neurons to encode information into spike trains. The main focus of the controversy was whether neurons use rate or temporal coding.

Rate code is the earliest proposed mechanism, which states that all the information is conveyed on the average rate of firing (generally stochastic) [2]. This hypothesis was supported by the fact, that in many cases, neurons responded to the same stimulus with different spike trains, while the average firing rate showed relatively small fluctuations [6, 44]. Rate code is considered to be robust to noise/inter-spike interval variability, since temporal jitter might alter the relative timing of individual spikes, but not the overall rate.

On the other hand, in temporal code, information about the stimulus is conveyed in the precise inter-spike intervals. Temporal code is generally more complex and theoretically much more efficient than rate code [40].

## 2.2 Neural operational modes

### 2.2.1 Temporal integration or coincidence detection

The two most dominant modes of neural operation are temporal integration and coincidence detection. The first suggests that the neuron acts as an integrate and fire device, depending on the input rate (rate code) [8, 37, 38], where as the latter takes into account the synchronous arrival of the input (temporal code) [1,41]. The difference between these two modes, can be characterized by the duration of the interval over which the neuron summates the presynaptic input, in other words, the temporal accuracy, that the input is taken into account. In case the integration interval is short, in respect to the presynaptic interspike interval, the neuron can be described as a coincidence detector. On the other hand, if this interval is longer, the neuron acts as a temporal integrator, firing a response after a certain number of presynaptic spikes on average.

Functional implications of both mechanisms have been proposed in the past. Coincidence detection has been associated with Hebbian learning and memory formation in the brain [9, 20] and synaptic plasticity [31]. Temporal integration has been linked with different cognitive processes, namely decision making [32] and motion perception [17].

Several contributions were made, in order to determine/characterize the operational mode of a neuron. For instance, the *coincidence advantage* measure tried to assess the ability of synchronous and asynchronous presynaptic spikes (termed synchronous and asynchronous gain) to cause a postsynaptic spike [1]. The results suggested that the cortical neurons act as coincidence detectors, than temporal integrators. Other contributions followed the argument that in integration mode, the spike train should be regular at high firing rates and tried to explain the high firing irregularity of cortical neurons, using the term *coefficient of variation* (CV) [7, 11, 41]. *Integration time window* was another measure, based on the idea that a temporal integrator would have long integration time in respect to the response rate, while in contrast for coincidence detection that interval is short [26].

### 2.2.2 The normalized pre-spike slope (NPSS) measure

Another proposed measure to identify the operational mode of a neuron is the *normalized pre-spike slope* (NPSS) [27]. It is based on the normalized slope of the membrane potential during a predefined time window prior to the spike. The slope is linearly normalized for each spike between a lower bound, representing the case where the neuron acts as a perfect integrator and an upper bound as if the neuron acts as a perfect coincidence detector. One of the limitations of the initial definition of NPSS, is the fact that it relied on the assumption that there were no inhibitory inputs.

An adaptation of the NPSS was introduced in [47] taking into account both excitatory and inhibitory inputs. In addition to the theoretical estimation of the bounds, an empirical process was demonstrated applying the metric on experimental data. Spikes were identified after the potential reached a specified value, adopted from previous work on the same data [29]. The ISIs during the non-stimulated period are defined as time between the lowest point of the "valley" (a term introduced in [29]) and a specified time prior to the next spike. For the stimulated period, the same definition of ISIs was used, as in [30], from the lowest point of the valley until the last decaying point before the following spike. The pre-spike slopes were grouped according to the ISI and for each group the upper and lower bounds were set to the maximum and minimum slopes, respectively. In order to smooth the results two 3rd degree polynomial functions were fitted on the bounds and shifted, so that they encompass the majority of the values. However as noted by the authors [47], the adapted theoretical NPSS cannot correctly infer the operational mode of a neuron, as the theoretical upper bound calculation should be modified such that it becomes a function of the excitatory/inhibitory input ratio.

## 2.3 Neural mechanisms in the visual cortex

### 2.3.1 Primary visual cortex

The visual cortex can be described at several levels of functional organization. The primary visual cortex, also known as visual area 1 (V1) is considered to be the first level of cortical processing of visual information. V1 neurons receive their main visual input from the lateral geniculate nucleus of the thalamus (LGN) and send their main output to subsequent, higher visual areas and subcortical structures e.g., V2, V3, MT etc. [16]. V1 also receives input from other brain regions, including pulvinar, claustrum, nucleus paracentralis, raphe system, locus coeruleus and the nucleus basalis, that are thought to have modulatory role [39]. One example is the cholinergic input from the nucleus basalis, which is suggested to be associated with alertness and attention, by changing the excitability of V1 neurons [19].

Area V1 is found in the cortex of all mammalian species [28], however research has been mainly conducted in carnivores (cats and ferrets), rodents (mice) and primates (macaques and humans). In primates, V1 corresponds to Brodmann's area 17, also called striate cortex [45], while in cats, V1 includes both areas 17 and 18, since both receive direct input from the LGN [34]. Cat primary visual cortex drew a lot of attention, due to the fact that the majority of cells in layer 4, the cortical layer that receives the dominant LGN input, are highly selective for stimulus orientation, although the same is not true in many other species.

### 2.3.2 Orientation selectivity of neurons in the visual cortex

A remarkable attribute of V1 neurons is their orientation selectivity, which does not apply to the relay cells of LGN, that provide the most information to the cortex. This property was first studied by Hubel & Wiesel [22], who proposed a simple model, which still remains at the center of a long-standing controversy over the synaptic mechanisms that underlie the orientation selectivity. Their model explains orientation selectivity of the cat visual cortex, considering only the thalamic input to a simple cell in layer 4.

Based on the Hubel & Wiesel model V1 neurons are divided into two types: simple and complex.

**Simple cells**

Simple cells are characterized by the elongated, adjacent *ON* and *OFF* subfields of their receptive fields. The subfields are positioned alongside each other, with their long axes rotated in an angle, that determines the preferred orientation of the cell. An illustration the simple cell receptive field is shown in Fig. 2.1. Simple cells' receptive fields are derived from the receptive fields of geniculate relay cells. An *ON* subfield consists of several *ON*-center relay cells, whose receptive field centers are aligned along the same region (Fig. 2.1). Subsequent studies provided further support to the above statement [14, 36]. Other arrangements of the subregions are also possible, e.g., a central *OFF* region surrounded by *ON* regions, or one *ON* and one *OFF* region.

A simple cell is excited when light spots fall in the *ON* area and dark spots on the *OFF* area. In the case of the reverse pattern of stimulation (dark on *ON* and light on *OFF*), the neuron is inhibited. This is also referred to as the push-pull mechanism. Source of the inhibition is shown to be intra-cortical relay cells [21]. Moreover, when the receptive field is uniformly illuminated or darkened, simple cells respond noticeably less, because of the mutually antagonistic relationship of the subregions. It is also suggested that the intracortical inhibition is the dominant mechanism of suppression [21]. Estimation of the fraction of excitatory input and the strength of inhibitory input in a push–pull model of the inputs to V1 simple cells is described in [10]. The results indicate that the maximum strength of the inhibition is about 30% of the maximum strength of excitation and that the V1 cells' firing can be induced if the percentage of active excitatory LGN inputs, exceeds about 40%.

The ratio of the elongation (length:width) of a subfield has been reported to range from 1.7 to 13 [23]. Similar values were proposed by [18, 25]. Furthermore, as it is predicted/implied by the Hubel & Wiesel model, the shape of the subfields is correlated with the width of the orientation tuning curves; cells with longer receptive fields, showed sharper sensitivity to the optimally oriented stimulus. [18, 23]. At later study the elongation was found to be minimal, having mean ratio of 1.7 [35]. These authors mentioned that the methodology used in previous work to determine the length of the receptive fields might have led to overestimation of that ratio.

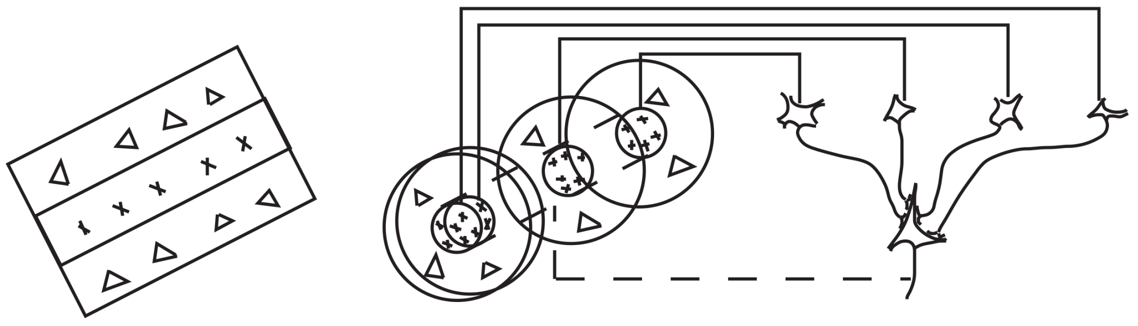


Figure 2.1: (Left) A receptive field of a simple cell in the cat visual cortex. A light covering the *ON* subregion (x) or dark the *OFF* region (triangles) excites the cell, whereas the reverse arrangement, light on *OFF* region or dark on the *ON* region inhibits the cell. (Right) The model of Hubel & Weisel [22] explaining the organization of simple receptive fields. The V1 simple cell (bottom right) receives input from relay cells (top right) whose receptive-field centers are located on the simple cell's central *ON* region. *OFF* relay cells whose receptive- field centers would be located on the simple cell's *OFF* regions are not shown. Adapted from [22]

.

### Complex cells

Complex cells are selective for orientation and spatial frequency, similar to the simple cells, although their selectivity cannot be directly predicted from the substructure of their receptive field [22]; there is no clear division of *ON*/*OFF* subregions. Thus, complex cells respond to a stimulus of the appropriate orientation regardless of position within the receptive field, e.g., in the case of a drifting visual grating, for simple cells the response is periodic, whereas in complex cells it is relatively continuous in time. Complex cells receive input from several simple cells and can be modeled by spatial pooling of simple cells, with overlapping receptive fields but different arrangements of *ON* and *OFF* regions [33].

For the purposes of this thesis, we are focused on the operational mode of simple cells, only.

### 2.3.3 Operational mode of Visual Cortical Neurons

Several evidence suggests that cortical neurons lower their threshold in response to rapid depolarizations, a form of sensitivity that increases according to the level of membrane potential [4, 13]. Further to that statement, findings support an adaptive coincidence-detection mechanism [5]. In the latter work, spike threshold showed an inverse nonlinear dependence on the rate of depolarization, which became steeper, with broader range of thresholds, when the cell was more depolarized. Based on these results, it was proposed that under high-input conditions the neuron adaptively enhances the sensitivity to synchronous inputs while simultaneously decreasing the sensitivity to temporally uncorrelated inputs (see Fig. 2 of [5]). In addition, spike activity and high-frequency fluctuations in membrane potential are both better tuned for stimulus orientation than the mean membrane potential.

# Chapter 3

# Methods

## Contents

## 3.1 Procedure of investigation

We first analysed intracellular recordings from simple cells of cats' visual cortex and applied the empirical procedure of the modified NPSS metric [47]. Taking into account the bursting behavior of the simple cells in certain stimuli, we made a selection of the spikes to be considered for the NPSS metric. Secondly, the neuron, the LGN receptive fields and stimulus were simulated with the prospect to apply the same metric to the simulated data. Knowing the underlying mechanisms of the simulation model, we would be able to validate the results and specify the operation of the real neuron. A critical point was the determination of the parameters of the simulation, namely amplitude, duration and degree of synchronization of the presynaptic input. Information about these parameters was derived after the examination of the postsynaptic potential of intracellular recordings. More specifically, we investigated the distribution of local maxima and local minima of the postsynaptic potential and tried to interpret the variability of the potential, after a series of simpler and gradually increased complexity simulations.

## 3.2 Data

The data used consisted of intracellular recordings from simple and complex neurons of area 17 of a cats' visual cortex, previously described in [12]. Estimation of membrane resistance and time constants of the same data based on off-line fitting of the cells' responses to current pulses of different amplitude was demonstrated in [3]. Neurons were stimulated by monocularly presented drifting sine-wave gratings. Optimal spatial frequency was determined from computer-generated spatial frequency tuning curves. Grating size, position and temporal frequency (1, 2, or 4 Hz) were adjusted to be optimal. The stimuli consisted of 12 different orientations (0-330° in 30° steps) and an additional blank screen presentation, 13 stimuli in total. The duration of each orientation was 4 sec. Sample-rate of the recordings was 4.0984e+03 Hz (time-step of 0.244 msec). Junction potentials according to the authors were measured to be 10 mV and were taken into account (added to the reported membrane potential values).

We focused on simple cells; cell 61 was chosen as a representative. As shown in Fig. 3.1 and the orientation tuning curves Fig. 4 of [12], cell 61 has a clear preference of orientation, eliciting no response to stimuli of non-preferred orientations and displaying greater activity for one direction of motion (270°) over the opposite (90°). The cell was stimulated by sinusoidal grating of frequency 4 Hz. A sine wave of the same frequency matches the elicited bursts in the response for preferred orientation and direction, as illustrated in Fig. 3.1.

Spike times in membrane potential traces were found by high-pass filtering the potentials and finding the crossings of the result with a threshold. For the purposes of the following experiments the spikes were removed, by replacing them with a straight line connecting the points before and after the spikes. According to [12], the spikes predominantly begin 1msec before and end 5msec after the peak in the membrane potential (mean duration 6.5msec).

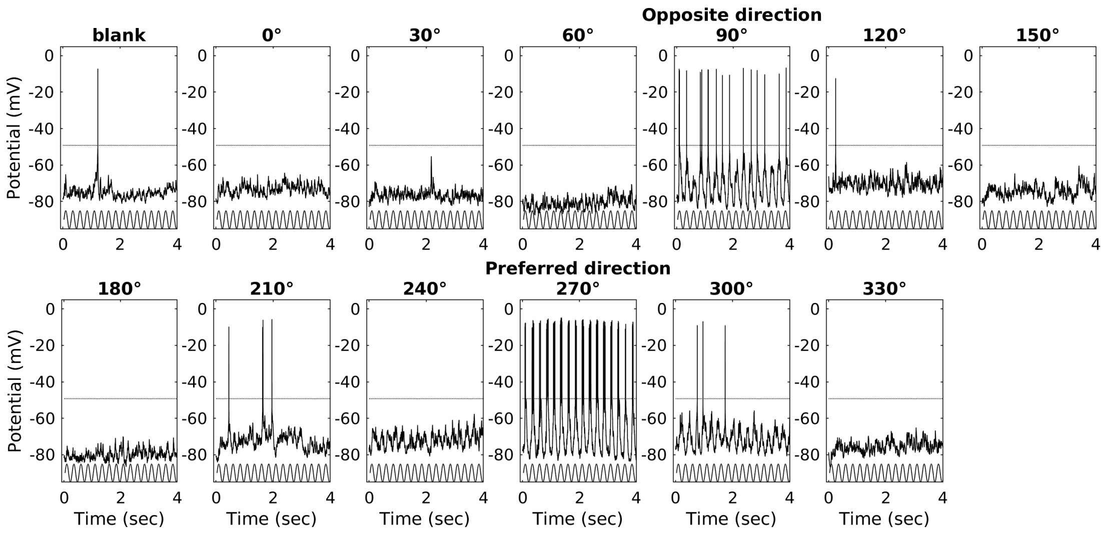


Figure 3.1: Responses of cell 61 for each stimulus direction. The cell shows intense activity for the preferred (270°) and opposite direction (90°), while nearly no response for the other directions. Below the membrane potential, a sine wave is shown with the same temporal frequency (4 Hz) as the stimulus.

## 3.3 Application of NPSS on intracellular recordings

In order to examine the operational mode of the given neuron based on its activity, we used the NPSS metric. Instead of applying NPSS on every spike, we chose the first spike of each burst and burst-group. Bursts were defined as the spikes having interspike interval at most 0.007 sec (colored red in Fig. 3.2) and burst-groups as the bursts with maximum interval time of 0.05 sec (colored green in Fig. 3.2).

Inter-burst interval ("interspike-interval") was defined as the time between the last spike of the previous burst (the time after that spike at $V_{th}$) and the first spike of the next burst (at $V_{th}$). The histogram (Fig. 3.4) shows two clusters of time-intervals, the shorter (more frequent) between the bursts belonging to a burst-group and the longer intervals between the burst-groups.

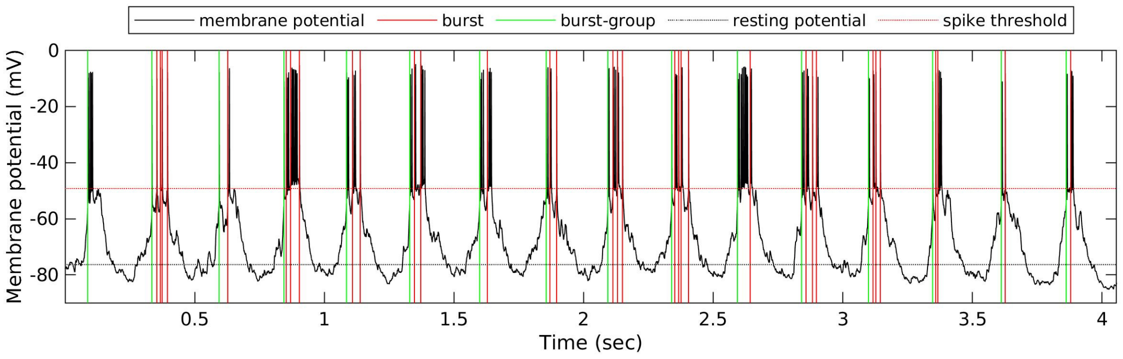
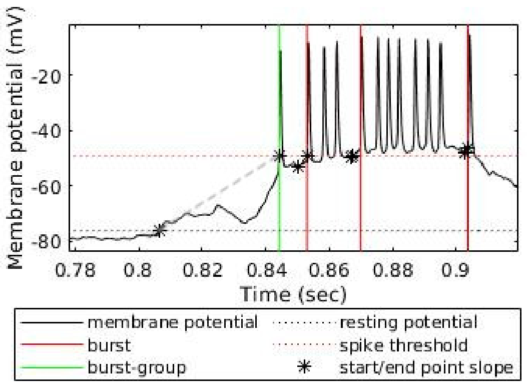


Figure 3.2: Membrane potential of cell 61 for preferred orientation (270°).

The NPSS measure requires to define a coincidence window, that corresponds to the maximum time-difference between a set of spikes, which is considered synchronous. The slope is later calculated from the beginning of the window to its end. In our case, this window had variable duration, as the start-point was set at the time when potential reaches $V_{rest}$ for the last time and end-point as the time the potential reaches $V_{th}$ before the spike. In case of burst-groups, when the potential stays close or even above the $V_{th}$, the start-point was set at the time of minimum potential value between the spikes and the end-point at $V_{th}$, if the start-point is below $V_{th}$, otherwise at the exact time of the spike. An example is illustrated at Fig. 3.3.



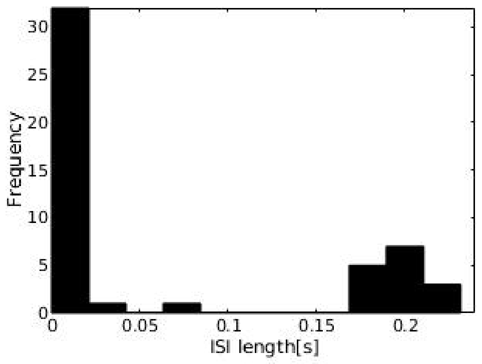


Figure 3.3: Close-up of the membrane potential of cell 61 for preferred orientation (270°). Stars indicate the start- and end-points used for the slope calculation (dashed grey line).

Figure 3.4: Histogram of ISI lengths of cell 61 for preferred orientation (270°).

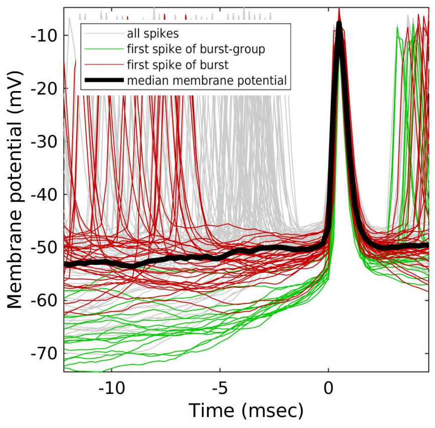
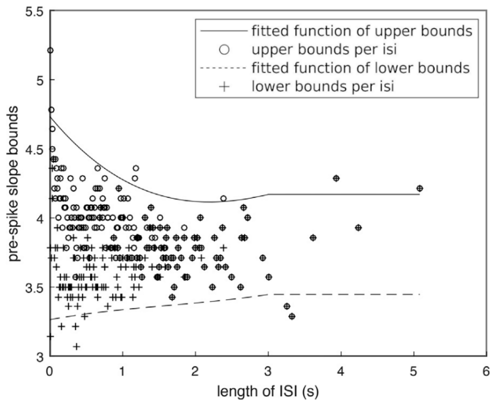


Figure 3.5: Spikes of cell 61 for preferred orientation.

To verify the correctness of the implementation, we reproduced Fig. 3 of [47] using the same data. The original figure is shown in Fig. 3.6a and its reproduction in Fig. 3.6b. We used a coincidence window of 7 msec, as described in the paper. The differences compared with the original figure lie in the different methodology used to calculate the interspike intervals (ISIs). According to the authors, for the silent period (non-stimulated period of the recording) the start of the ISI was defined as the lowest point of a valley, as in [29] and the point at 0.01005 sec prior to the next spike as the end. In addition, for the stimulated period, the ISI definition differed slightly i.e., the end of ISI was the last decaying point before the spike. In contrast, we empirically defined the ISIs, as the last time the membrane potential had value above -50 mV until the time it reached -30 mV. Moreover, for the original figure, the ISIs longer than 3 sec were grouped together; that is the reason why the fitted functions remain constant after that duration.



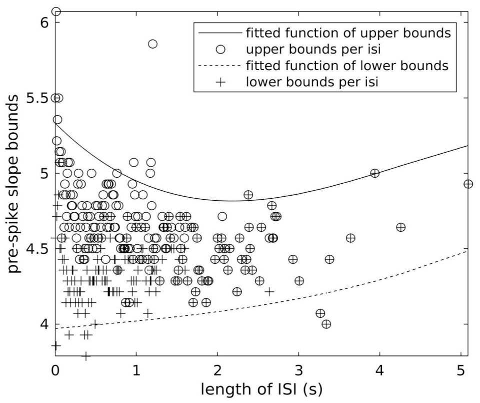


Figure 3.6: Upper and lower bounds for each group range of ISIs, indicated by circle and cross, respectively. The solid and dashed lines represent the fitted functions for each case. (Left) Original Fig. 3 of [47]. (Right) Our reproduction of the same figure.

The implementation of the NPSS and its application on the intracellular recordings is in A.1 and A.2.

## 3.4 Simulation of V1 neuron

The simulation consists of a V1 simple cell, modeled by a Leaky Integrate-and-Fire (LIF) neuron [43], receiving presynaptic input from LGN cells of centre-*ON* or centre-*OFF* type. Presynaptic spike trains were generated by inhomogeneous Poisson processes, whose rate was modulated by the stimulus and position of the LGN receptive fields at the current time.

### 3.4.1 Neuron model

The V1 simple cell was defined as a LIF model (Eq. 3.1) using the BRIAN simulator [42], with membrane time constant $\tau_m$, spike threshold $V_{th}$, refractory period $\tau_{ref}$, reset potential $V_{reset}$ and membrane resistance $R_m$.

$$\begin{gathered} \tau_m \frac{dV}{dt} = -[V(t) - V_{rest}] + R_m I(t) \\ V(t) = V_{reset}, \text{ during refractory period} \end{gathered} \tag{3.1}$$

The synaptic input current to the neuron was defined as the summation of $\alpha$-function shaped postsynaptic currents (PSC), described by

$$I(t) = \sum h_s(t - t_i) \tag{3.2}$$

where $t_i$ is the time of presynaptic action potentials and $h_s(t)$ is the PSC kernel

$$h_s(t) = \begin{cases} we\tau_s^{-1} t \exp(-t/\tau_s), & t > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

where $w$ is the synaptic weight and $\tau_s$ the synaptic time constant. $h_s(t)$ peaks at value determined by $w$ at time equal to $\tau_s$. Excitatory and inhibitory synapses may have different weight and time constant.

The implementation/definition of the neuron and synapses is in A.9

### 3.4.2 Stimulus and receptive fields

The neuron was stimulated by sinusoidal grating varying both in space and time. The stimulus is described by

$$s(x,y,t) = Acos(Kxcos\Theta + Kysin\Theta - \Phi)cos(\omega t) \tag{3.4}$$

where $x$ and $y$ are the coordinates on the grid; $t$ is the time; $K$ and $\omega$ are the spatial and temporal frequency; $\Theta$ and $\Phi$ are the orientation and spatial phase of the grating; $A$ is the contrast amplitude. Fig. 3.7 is an illustration of the stimulus, showing the effect of each parameter.

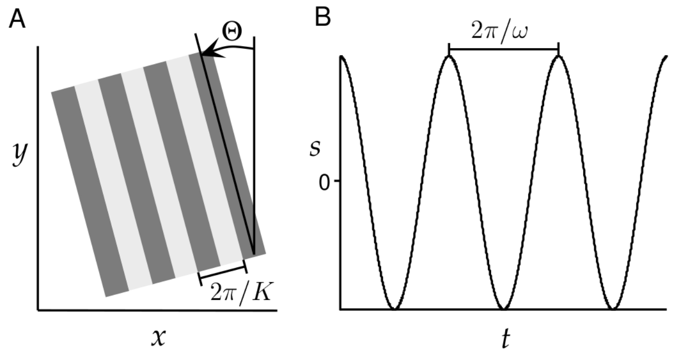
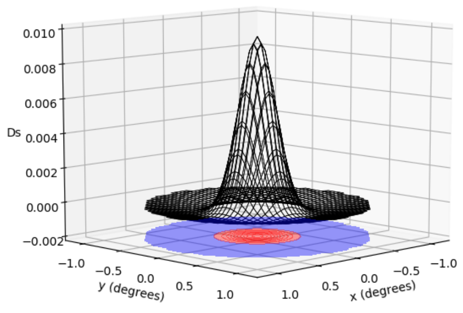


Figure 3.7: (A) A square-wave grating analogous to the sinusoidal grating of Eq. 3.4, where the lighter stripes are areas of $s > 0$ and darker stripes of $s < 0$. $K$ determines the spatial frequency of the wave and $\Theta$, its orientation. $\Phi$ is the spatial phase, which shifts the pattern in the direction perpendicular to the direction of the stripes. (B) The light-dark intensity at any point of the spatial grating oscillates sinusoidally in time with period $2\pi/\omega$. Adapted from Fig. 2.8 of [15].

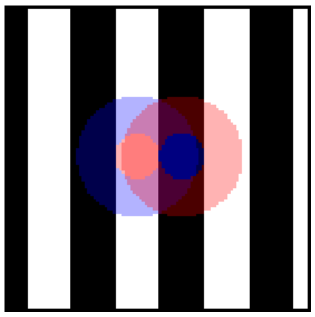
The spatial structure of the receptive fields of the LGN cells were captured by the difference of two Gaussians (DoG), adapted from the spatiotemporal LGN model of [46].

$$D_s = \pm\left(\frac{K_{center}}{2\pi\sigma^2_{center}}\exp\left(-\frac{x^2+y^2}{2\sigma^2_{center}}\right) - \frac{K_{surround}}{2\pi\sigma^2_{surround}}\exp\left(-\frac{x^2+y^2}{2\sigma^2_{surround}}\right)\right) \tag{3.5}$$

and parameter values $\sigma_{center}$ = 10.6', $\sigma_{surround}$ = 31.8' and $K_{center}/K_{surround}$ = 17/16. The sign $\pm$ determines the type of the receptive field, center-$ON$ (+) and center-$OFF$ (-). The result was then normalized, so that the sum of absolute values is one, in order to control the firing rate. Fig. 3.8a illustrates a center-*ON* spatial receptive field. The implementation of the receptive field's kernel is in A.11.



(a)



(b)

Figure 3.8: (a) A center-*ON* spatial receptive field modeled as a DoG. The positive and negative areas are colored red and blue, respectively. (b) Illustration of the stimulus and receptive fields. On the left side there is one center-*ON* receptive field (with red/positive center and blue/negative surround area) and on the right one center-*OFF* receptive field (with blue/negative center and red/positive surround area). The black(=-1) and white(=1) stripes represent the stimulus used in the simulations.

The firing rate of each LGN cell at each time is equal to the sum of the element-wise product of the receptive field kernel with the stimulus, negative firing rate set to zero. The optimal stimulus for a center-*ON* receptive field, in order to achieve the maximum firing rate is white (=1) in the center and black(=-1) in the surrounding area. This position was never achieved because of the stimulus' shape, i.e., stripes instead of circles. As shown in Fig. 3.8b, the spatial frequency of the stimulus was set so that the width of each black and white stripe is equal to the diameter of the center of each receptive field and the surrounding area falls into the opposite colored stripe. This has as a result the maximum possible firing rate. The calculation of the firing rate in the simulation is in A.13.

For the purposes of the experiments, the receptive fields were placed in two rows; a row with excitatory center-*ON* and inhibitory center-*OFF* receptive fields and a row of the opposite combination, inhibitory center-*ON* and excitatory center-*OFF*. The optimal case of the stimulus for excitation is when there is light on the first row (activation of excitatory center-*ON*) and dark on the second row (activation of excitatory center-*OFF*), thus the V1 cell receives only excitatory input. On the other hand, the optimal case for inhibition is when the first row is in the dark (activation of inhibitory center-*OFF*) and light on the second row (activation of inhibitory center-*ON*).

## 3.5 Determination of presynaptic input parameters

We tried to infer the following parameters of presynaptic input, from the postsynaptic potential, in order to optimize the simulation's parameters:

1. Amplitude of individual presynaptic events
2. Duration of individual presynaptic events
3. Level of synchronization

For the purpose of determining the amplitude of the presynaptic events, we examined the distribution of the local maxima/minima of the postsynaptic potential. We also introduced a measure of the variability of the postsynaptic potential and used the method for specifying the above parameters, which were required by the simulation.

Considering the fact that at the preferred orientation the neuron receives presynaptic input mostly only excitatory or inhibitory, we separated these two cases, by dividing the postsynaptic membrane potential into three different regions based on neuron's resting potential:

1. **Pre-spiking period:** the period starting from the time the membrane potential is above the resting potential until the first spike of each burst (colored green in Fig. 3.9)
2. **Excitatory period:** when the membrane potential is above the resting potential (colored blue in Fig. 3.9). Note that the pre-spiking period in included in the excitatory. Spikes are not considered in the calculations.
3. **Inhibitory period:** when the membrane potential is below the resting potential (colored orange in Fig. 3.9).

Because the spikes of the postsynaptic neuron affect its potential, keeping the potential high and "hiding" the effects of the presynaptic input, attention was focused on the pre-spiking and inhibitory periods.

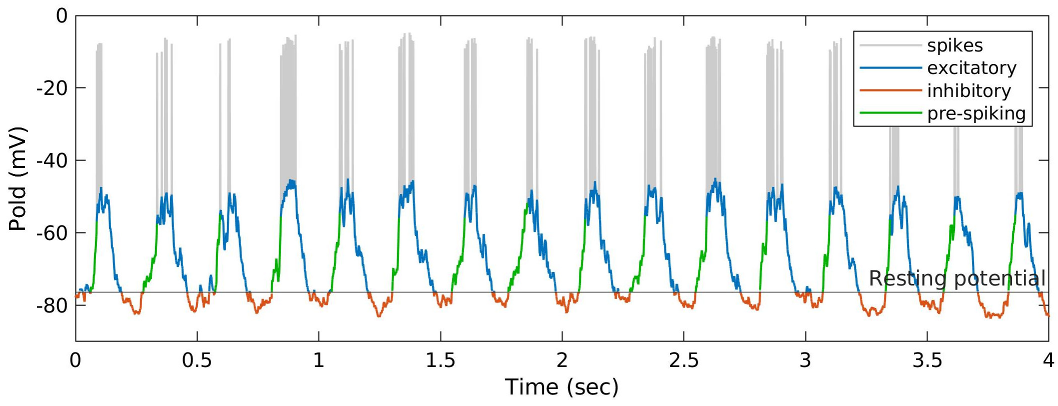


Figure 3.9: The different periods on intracellular recordings (cell 61 - preferred orientation). Note that the pre-spiking period (green line) in included in the excitatory period (blue line).

### 3.5.1 Distribution of local extrema of postsynaptic potential

We examined the distribution of local maxima for the pre-spiking period and local minima for the inhibitory period, with the hypothesis that the height of presynaptic inputs would be reflected on the resulted histogram as peaks on discrete multiples of the inputs' height. In other words, the histogram captures the frequency of the values of local maxima/minima of the membrane potential. In the ideal case of perfect synchrony of the presynaptic input, the local maxima/minima of the postsynaptic potential will be equal to the number of the presynaptic events at the current time multiplied by their amplitude. Thus, it is more likely to have peaks in the histogram every certain values - multiples of the amplitude of presynaptic events. Intermediate values on the histogram are obtained from overlapping presynaptic potentials, whose timing differs to a small degree.

To justify the results, Gaussian bells (Eq. 3.6) were fitted into the histogram, with individual width ($w_i$) and amplitude ($h_i$) and distance from initial point ($a$) in multiples of $b$. Parameter $b$ is the predicted amplitude of the presynaptic inputs. The number of the fitted Gaussian bells (parameter $k$) is empirically set. During the fitting process lower and upper bounds were set, in order to prevent overfitting (very sharp) or any Gaussian bell to cover the rest of them (very wide).

$$\sum_{i=1}^{k} h_i \exp(-(x-(a-i*b)^2/w_i)) \tag{3.6}$$

The method was tested on simplified numerical experiments and then applied on the intracellular recordings. The implementation of the method is in A.4.

### 3.5.2 Measure of variability

#### Definition

For the sake of simplicity, we use the terms *Pold*, as the cell's membrane potential and *Pnew*, as the difference of every value of *Pold*, from the average of its values preceding and succeeding at a specific interval. The variability of the membrane potential is defined as the standard deviation of *Pnew* over different intervals. A pseudocode of the measure of variability is shown below.

**Algorithm 1:** Pseudocode of the measure of variability

```
output: Variability of the membrane potential

$Pold \leftarrow membrane\ potential$
for each interval in range of intervals do
    $Pnew \leftarrow [\,]$
    for $i \leftarrow 0$ to $LENGTH(Pold) - 1$ do
        $Pnew[i] \leftarrow Pold[i] - AVERAGE(Pold[i + \textbf{interval}], Pold[i - \textbf{interval}])$
    end
    $variability[interval] \leftarrow STANDARD\ DEVIATION(pnew)$
end
```

Oscillatory structures and fluctuations of the membrane potential, such as peaks, are identified by calculating *Pnew*, which for short intervals acts as an edge detector. For instance, maximum values of *Pnew* occur when the central value of *Pold* is located on a peak, so it differs the most from the average of its next and previous values. The intensity of these oscillations and their temporal relationship between one another are captured by the standard deviation of *Pnew* and by varying the duration of intervals.

The membrane potential's periods of interest were the pre-spiking and inhibitory, where the effects of the presynaptic input are not affected by the postsynaptic neuron responses. For each different period, only *Pnew* values, that use data of the same type of period were taken into account. This had as a result, the number of *Pnew* points used in the calculation of standard deviation to decrease, as the duration of the interval increased. This effect was mostly noticeable for the pre-spiking period, which has much shorter duration compared to the inhibitory. For higher confidence of the results, a lower limit to the number of *Pnew* values was set at 100.

### Methodology - Application of the measure

For the purpose of understanding the role of input's parameters on variability, we conducted a series of gradually increased complexity numerical experiments. Using simulations, we examined separately the effects of each parameter, namely amplitude, duration and synchronization of the individual presynaptic events, which were represented as alpha functions. We chose to use alpha functions, because they were considered more realistic for our purposes and we had control over the parameters of interest. For the timing of the alpha functions we used uniform and Poisson distributions.

We first examined the effects of amplitude, by varying the height of the alpha functions. The duration of the individual events was tested by controlling the width of each alpha function based on their shape parameter. At last, for the synchronization we used different rate and multiple number of independent Poisson processes.

Using the acquired knowledge, we tried to approach and interpret the variability of the real intracellular recordings. In order to validate our conclusions, the observations will later be tested on the simulated neuron.

The implementation of the measure and its application on numerical experiments and intracellular recordings is in A.5 and A.6.

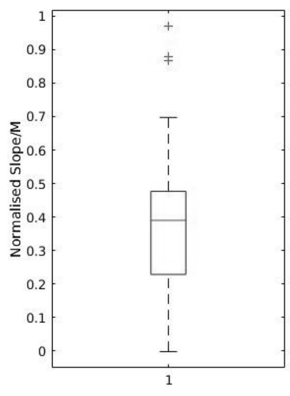
# Chapter 4

# Results and Discussion

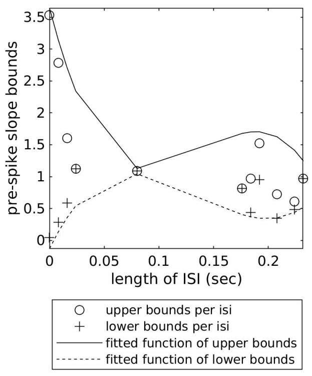
## Contents

## 4.1 Application of NPSS on intracellular recordings

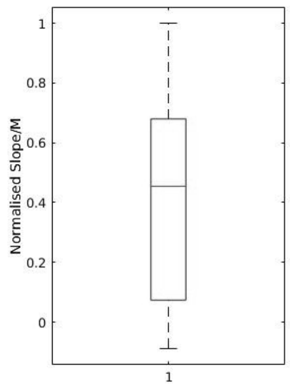
The NPSS metric was first applied to the membrane potential of cell 61, in order to determine its operational mode. Fig. 4.1 shows the results of NPSS on three cases, for the first spike of all bursts, of each burst-group only and of each burst belonging to a burst-group. Because of the smaller number of spikes in the case of burst-groups, the bin-size was set to 0.025 sec, instead of 0.008 sec like in the other two cases. Considering all the spikes, the results indicate that the neuron acts more as a temporal integrator, as the maximum NPSS value (excluding the outliers) is at 0.69 and the median at 0.39 (Fig. 4.1a). Taking into account only the first spike of each burst-group, NPSS values are more evenly distributed, with maximum number at 1 and median at 0.45 (Fig. 4.1c). For the case of the NPSS applied to the first spike of each burst belonging to the burst-group, the NPSS values are slightly lower than the case of all spikes (Fig. 4.1e).



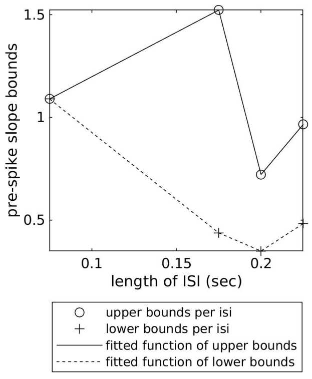
(a) NPSS values for the first spike of each burst.



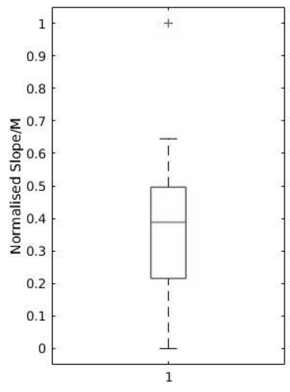
(b) Estimated functions of upper and lower bounds for the first spike of each burst.



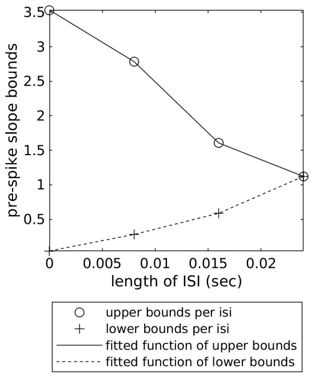
(c) NPSS values for the first spike of each **burst-group only**.



(d) Estimated functions of upper and lower bounds for the first spike of each **burst-group only**.



(e) NPSS values for the first spike of each **burst belonging to a burst-group**.



(f) Estimated functions of upper and lower bounds for the first spike of each **burst belonging to a burst-group**.

Figure 4.1: NPSS on cell 61 on preferred orientation and direction.

## 4.2 Simulation of V1 neuron

The simulation of V1 neuron was implemented with the prospect to apply the NPSS metric to the simulated data and further examine neuron's behavior. Fig. 4.2 shows the membrane potential of the simulated neuron, stimulated by different orientations. Blue and red lines below the membrane potential show the time of presynaptic excitatory and inhibitory input. Preferred orientation was set at 0° (and opposite 180°). The results are only preliminary, as the simulation parameters had not yet been optimized. Nevertheless, there is clear preference to the orientation, since the presynaptic excitatory and inhibitory input are aligned. For non-preferred orientations the neuron is excited and inhibited at the same time, because of the fact that the light of the stimulus covers both *ON* and *OFF* receptive subfields, which contradict each other. As a result, the membrane potential stays at lower levels, never reaching the firing threshold. This also shows that the push-pull mechanism can be characterized by a certain level of synchrony on its own. The LGN relay cells, which are the main source of input, begin to fire at relatively the same time during the preferred orientation, since they all fall in the same brightness-region (light/dark stripe) of stimulus.

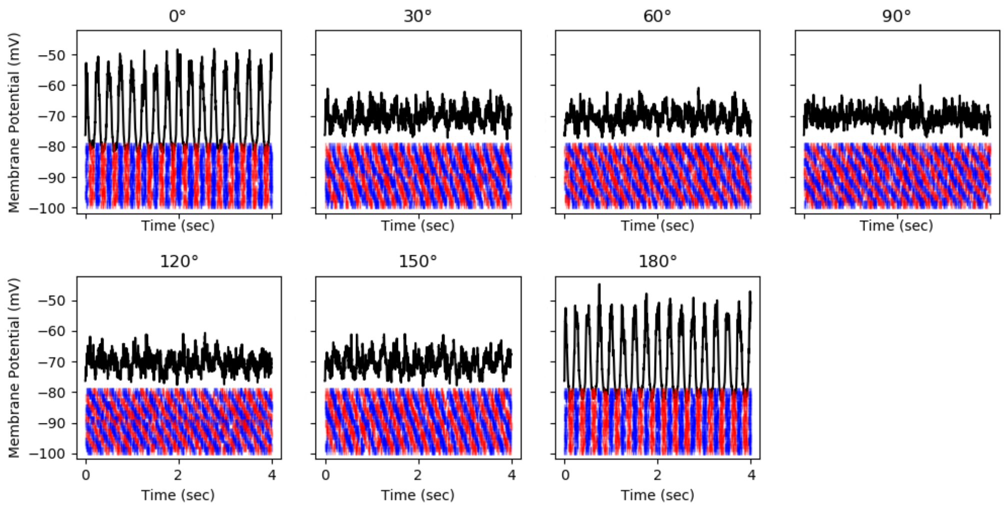


Figure 4.2: Membrane potential of the simulated V1 neuron for different orientations. The time of presynaptic excitatory and inhibitory input, each one corresponding to a receptive field, is shown beneath the membrane potential by blue and red lines, respectively. The preferred orientation of the neuron is 0° (and opposite 180°). Note that simulation parameters have not been optimized.

## 4.3 Determination of presynaptic input parameters

### 4.3.1 Test and validation of the proposed methods

#### Distribution of local extrema

One approach, that we tested in order to determine the amplitude of the individual presynaptic events is the study of the distribution of the local extrema. An example is in Fig. 4.3. We used a single Poisson process of alpha functions. The rate was set at 200 Hz, height of alpha functions was 4 mV and shape parameter 0.002 sec. The histogram of the local maxima along with the fitted curve and the predicted bounds is shown below. The size of the bins was adjusted at 0.6 mV, so that the pattern/peaks of the histogram were more discernible.

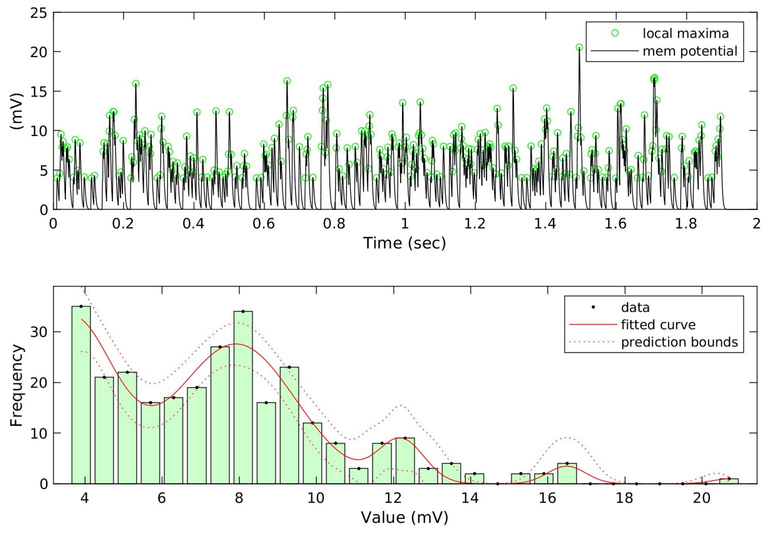


Figure 4.3: (Top) A single Poisson process of alpha functions of rate 200 Hz, height 4 mV and shape parameter 0.002 sec. (Bottom) The histogram (bin size 0.6 mV) of the local maxima, the fitted curves of Gaussian bells (red line) and the prediction bounds (dotted red lines).

We empirically fitted five Gaussian bells (Eq. 3.6), because of the five distinct peaks on the histogram. The fitted curve is illustrated by a solid red line. The key element, which provides us with the predicted amplitude, is that each Gaussian bell is positioned with equal distance between them (parameter *b* from Eq. 3.6). Equivalently, every Gaussian bell has distance from an initial point (parameter $a$) in multiples of the same value (parameter $b$). The fitted value corresponding to parameter $a$ was -0.65 mV (95% confidence bounds -1.6 – 0.29 mV) and to $b$ was 4.28 mV (95% confidence bounds 3.86 – 4.69 mV), which managed to approach the height of the alpha functions (4 mV). It is worth noting that the starting point of the coefficients affected the resulted coefficient confidence-bounds. The adjusted R-squared value for the fitting was 0.91. Another remark is that the results were more accurate when the alpha functions had shorter duration, which reduced the probability of intermediate values between peaks and made the histogram clearer.

**Measure of variability**

**Effect of amplitude of individual events on variability.**

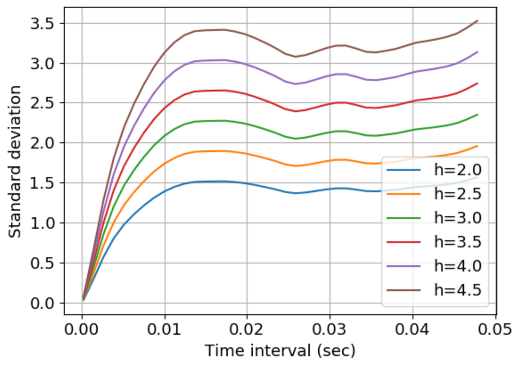


Figure 4.4: Effect of the amplitude of individual alpha functions on variability of the resulting potential. Numerical experiment, using a single Poisson process of alpha functions, with constant rate (100 Hz), shape parameter of alpha functions (0.003 sec), but variable amplitude (2.0 – 4.5 mV).

We examined the effect of amplitude of individual presynaptic events on variability, through simpler simulations. We used a single Poisson process of alpha functions, whose rate and duration remained constant and increased the amplitude of each alpha function. The results are shown on Fig. 4.4. As the height (h) of alpha functions increases, we notice an increase of the variability, as well; the standard deviation is shifted upwards. Similar results were obtained using different values for rate and duration of the alpha functions. The changes of the potential become stronger, as a consequence of higher alpha functions. Thus the difference of each point from the average of the points before and after each interval (value referred to as *Pnew*) also increases. This has as a result the rise of the standard deviation of *Pnew*.

An interesting observation is that the variability of potential seems to be linear, while changing the height of EPSPs.

**Effect of duration of individual presynaptic events on variability**

In order to understand how the duration of the individual events affect variability, we first examined the variability of uniformly distributed alpha functions and changed their duration by increasing the shape parameter (alpha) of the functions. The shape parameter equals the time required to reach the peak of the alpha function. Note that alpha functions are not symmetrical, so the shape parameter does not represent directly the duration of the function.

The initial experiment was conducted using low (25 Hz) and high (100 Hz) rate. The results are illustrated in Fig. 4.5. Before proceeding with the interpretation of the effects of duration of alpha functions, it is worth mentioning the impact of rate on the variability. On both plots, we notice that the variability follows a periodical pattern; this is more pronounced when the rate is high at 100 Hz (Fig. 4.5 (right)). This behavior is linked with the time between each alpha function. The variability reaches its highest point for the first time, at the interval, equal to the half of distance between the peaks of the alpha functions. For instance, for the case of rate at 100 Hz, there is an alpha function every 0.01 sec. Therefore the variability has a maximum at the interval of 0.005 sec. At that interval, we compare the value of the peak of every alpha function, with the points located at the valleys, resulting to the maximum/minimum possible values of *Pnew* and consequently to the peak of the standard deviation. As the interval increases until the point that is equal to the distance between the alpha functions, we end up comparing corresponding values of the potential, from different alpha functions. In other words, for a rate of 100 Hz, 0.01 sec before and after a peak of any alpha function, correspond to the peaks of the alpha functions preceding and succeeding that interval. This implies the decrease of *Pnew* values and of the standard deviation. The pattern repeats, as the intervals are increased further, reaching subsequent alpha functions with equivalent combinations.

Regarding the variability during high rate of input, we notice an upward trend on the minimum values. Because of high rate the alpha functions are densely positioned, the potential remains relatively at high levels. The initial increase of the potential is responsible for the increasing minimums, as the interval grows. In contrast, excluding that short period, the minimum values are constant.

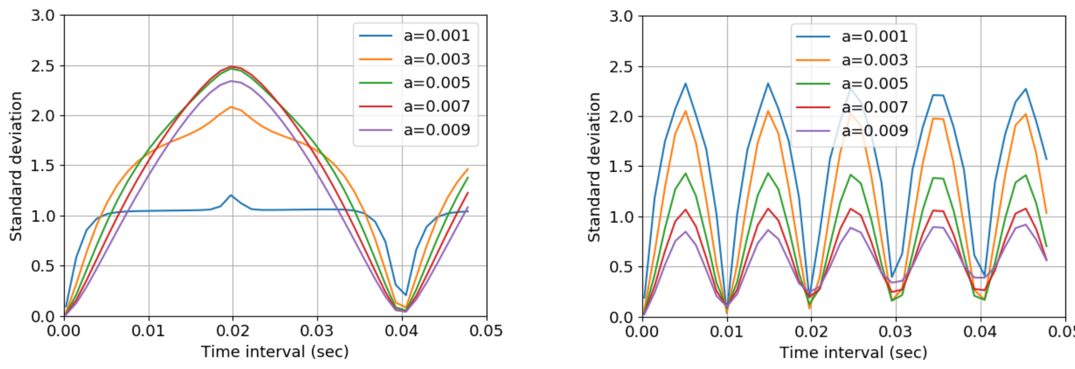


Figure 4.5: Effect of the duration of individual alpha functions, uniformly distributed, on the variability of the resulting potential. Numerical experiment, using alpha functions, with constant rate of 25 Hz (left) and 100 Hz (right), amplitude (4 mV), but variable shape parameter (a) - time to reach peak (0.001 – .009 sec).

Concerning the effects of the duration of each input, in both cases, we notice that for shorter duration of events, the variability rises more steeply. Moreover, on the one hand, when the rate is low, the variability reaches higher values, as the duration increases (for a=0.001 – 0.007 sec) and the alpha functions overlap together. However for the longest duration (a=0.009 sec) variability is a bit lower compared to the previous value (a=0.007 sec). This is also the case of the rate at 100 Hz. This is due to the overlapping alpha functions, which keep the valleys of final potential at relatively high (do not reach 0 mV).

Comparing the variability between low and high rate, the duration of the alpha functions controls the growth rate of the variability, where as the input rate controls the intervals of the extrema of the variability (maxima and minima), acting as a bound-interval up to which the variability can increase.

The same experiment was repeated, but instead of the timing of alpha functions to be uniform, we use a single Poisson process of constant rate (100 Hz). The results are presented in Fig. 4.6. Instead of the periodic pattern shown in the previous experiment, the variability after reaching a maximum value remains at similar levels. It is clearly seen again that the duration of the alpha functions affects the rate, at which the variability increases for the first intervals; longer duration of the alpha functions implies slower/gradual increase of variability.

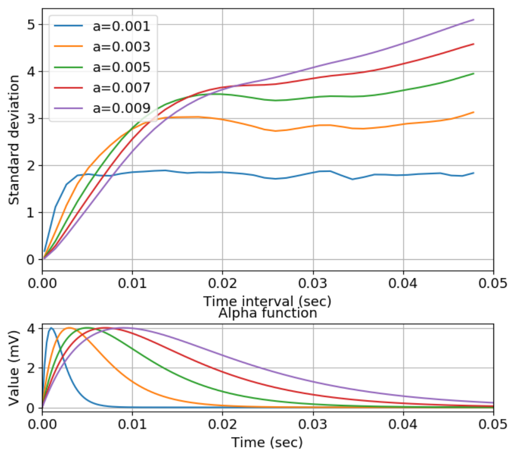


Figure 4.6: Effect of the duration of individual alpha functions, following the Poisson distribution, on the variability of the resulting potential. Numerical experiment, using alpha functions, with constant rate (100 Hz), amplitude (4 mV), but variable shape parameter (a) - time to reach peak (0.001 – 0.009 sec). The parameters used are the same as on Fig. 4.5. (Top) The variability of the resulting potential. (Bottom) The individual alpha functions.

#### Effect of input synchronization on variability

We further examined the role of input rate. The plots of Fig. 4.7 arise from alpha functions of constant duration and amplitude, uniformly distributed (on the left) and based on a single Poisson process (on the right) for various rates (30 – 500 Hz). In general, the variability of uniformly distributed alpha functions was explained in the previous section. Nevertheless, it is mentioned again as a comparison with the corresponding Poisson processes. The periodic nature of the variability, when timing between input is uniform, is not visible after the introduction of randomness (use of a single Poisson process). Due to the randomness, there is not a single interval to match the distances between every alpha function in the Poisson process. In other words, for every interval there are still high *Pnew* values.

The oscillations of the variability shown, especially at higher rates (300 Hz and 500 Hz) are due to random structures in each case. To verify that, the experiment was repeated with different random seed. It seems that the major effect of the rate is the increase of the variability, since there are more overlapping alpha functions, from which bigger structures emerge.

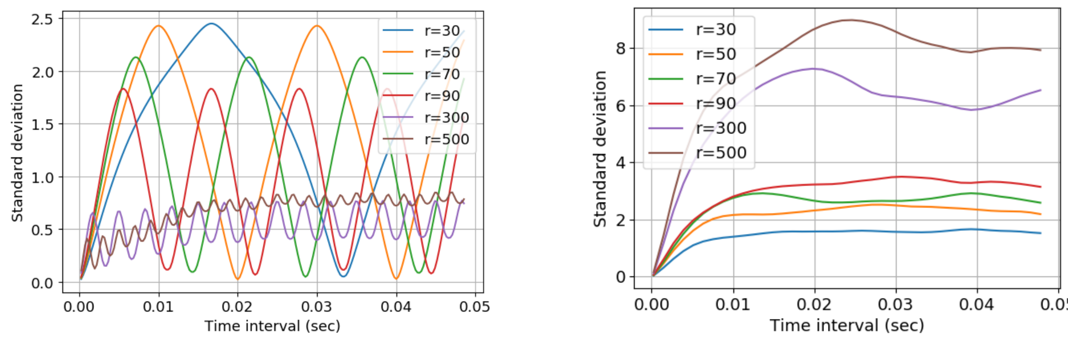


Figure 4.7: Effect of the rate of alpha functions on the variability of the resulting potential, following the uniform distribution (left) and Poisson distribution (right). Numerical experiment, using alpha functions, with constant shape parameter (0.003 sec), amplitude (4 mV), but variable rate (30 – 500 Hz).

The effect of the number of inputs on the variability was tested by using multiple Poisson processes, whose accumulated value remained constant, i.e., the total value was divided by the number of Poisson processes. As illustrated on Fig. 4.8, the number of inputs has no effect on the variability, since the sum of independent Poisson processes is equivalent with a single Poisson process, if the total rate remains the same. Any variation between the results, are due to the randomness.

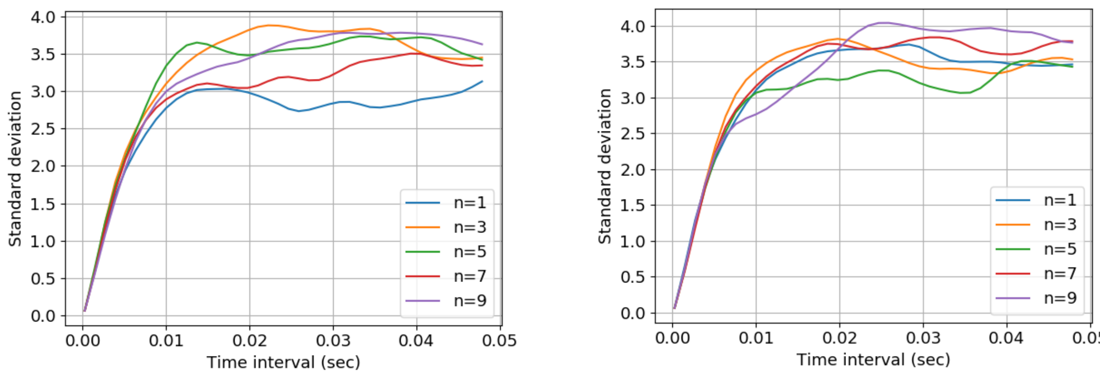


Figure 4.8: Effect of the number of inputs on the variability. Numerical experiment, of variable number (n) of Poisson processes of alpha functions, with constant total rate (100 Hz), height (4 mV) and shape parameter (0.003 sec). The two plots resulted from the same experiment, but with different random seed.

### 4.3.2 Application of the proposed methods on intracellular recordings

#### Distribution of local extrema of postsynaptic potential

The method was applied on the membrane potential of cell 61 on the preferred orientation. The results are shown in Fig. 4.9. In both histograms, we can see peaks emerging at relatively equal intervals. More specifically, for the pre-spiking period, there seem to be six peaks, separated by about 5 mV interval and for the inhibitory period four peaks, every about 3 mV. Regarding the histogram of local maxima, we fitted five Gaussian bells. In terms of the Eq. 3.6, the fitted value corresponding to parameter *a* (initial point) was -76.1 mV (95% confidence bounds -76.91 – -75.29 mV) and to *b* (estimated amplitude) was 4.74 mV (95% confidence bounds 4.38 – 5.01 mV). Regarding the local minima values, three Gaussian bells were fitted. The resulting value for parameter *a* was -76.09 mV (95% confidence bounds -77.04 – -75.13 mV) and for *b* was 3.13 mV (95% confidence bounds 2.54 – 3.73 mV). The adjusted R-squared value for the fitting for the local maxima was 0.81 and for local minima 0.76. The fitted parameter a for both cases are close to real neuron's resting potential which is -76.33 mV.

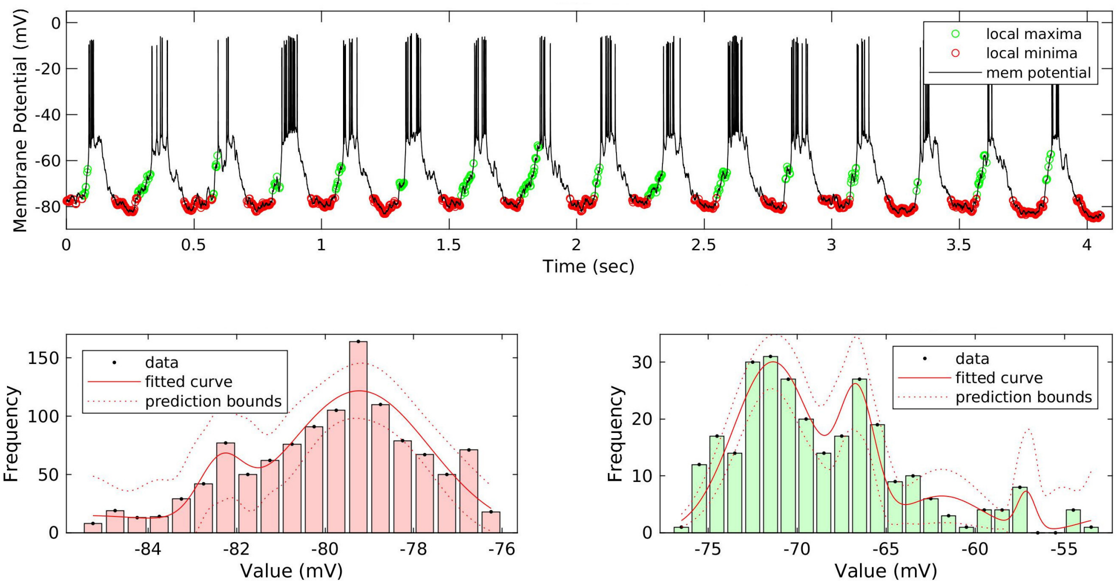


Figure 4.9: (Top) The membrane potential for cell 61 on preferred orientation. Red and green circles indicate the local minima and maxima, respectively. (Bottom) The histograms of the local minima (Left) and maxima (Right) and the fitted curves of Gaussian bells (red line). Histogram's bin size of local minima was set at 0.5 mV and for local maxima at 1 mV.

## Measure of variability

The measure of variability was applied to the intracellular recordings. Fig. 4.10 shows the variability of each defined period, namely pre-spiking period, inhibitory, excitatory and for all the duration of the recording. The dashed lines represent the variability of simulated data, estimating the pre-spiking and inhibitory period. More details about the estimation process are in the following subsection. As it was mentioned earlier, we focused only on the pre-spiking (green line) and inhibition (orange line) periods, since the spikes of the postsynaptic cell affect the shape of the membrane potential. The variability of the pre-spiking period stops at earlier intervals, due to its short duration and the defined limit of 100 points to be used in the calculation.

The variability of the pre-spiking period increases until the 0.01 sec interval, much sharper than that of the inhibitory period. Following that initial increase, the variability oscillates with a decreasing trend. The maxima of the oscillations appear at relatively uniform intervals. Specifically, the variability peaks at 0.01 sec, 0.02 sec and approaches a peak at 0.03 sec. In between these peaks, its value is minimized.

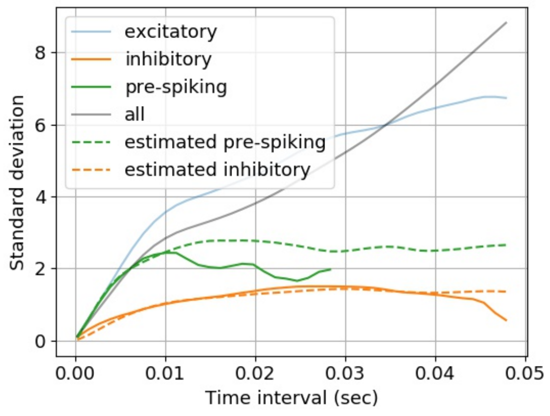
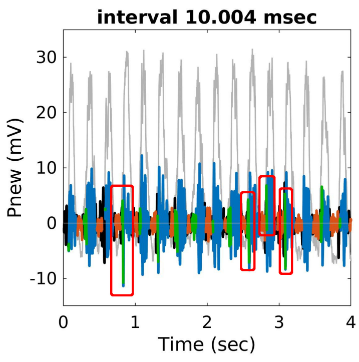


Figure 4.10: Variability of the membrane potential of cell 61 on preferred orientation for each defined period pre-spiking (green), inhibitory (orange), excitatory (blue) and for all the duration of the recording (black). We focus only on the pre-spiking and inhibitory periods. Standard deviation of the pre-spiking period stops at interval 0.028 sec, because for longer intervals there are less points than the limit set to be considered in the calculation. The dashed lines show the variability of simulated data, estimating the features of the presynaptic input.

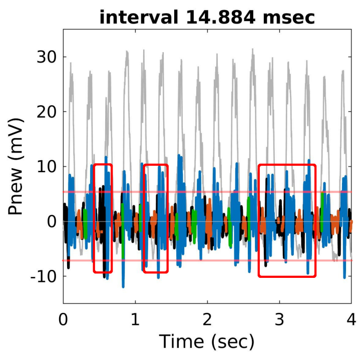
These oscillations, drew our attention and we tried to explain this behavior. It seems that this is due to a combination of the short duration of the pre-spiking period and the increase of the membrane potentials in steps. As the intervals used in the calculation become longer, in order to compare farther points, the points located at the bounds of each period are excluded. These affects the standard deviation, since the mean of the *Pnew* values could be greatly affected. Such a case is illustrated at Fig. 4.11, where the number of bursts, whose *Pnew* values (green line) are used, becomes significantly small. For this reason, we may need to consider even shorter intervals.

Another characteristic that contributes to these oscillations is the way that the membrane potential increases. The greatest values of *Pnew* are in cases, where the point of interest has value different from the average of the two other points. For instance, the central point is a local minimum, whereas the points located before and after the examined interval are on peaks. A common case, due to the increasing trend of the pre-spiking period, is that the point of interest and the preceding point have similar values, while on the contrary the preceding point has much higher value, since it is closer to the firing threshold. Fig. 4.12 shows periods of the membrane potential, where this is the case for certain intervals.

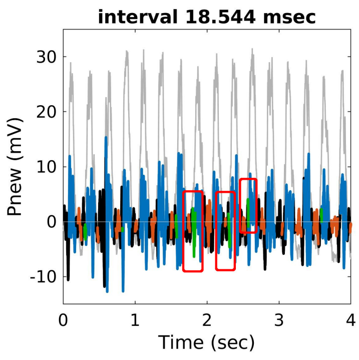
In contrast, the variability of the membrane potential during inhibition increases more smoothly and stays at lower levels compared to the pre-spiking period. Also, the variability does not oscillate. The findings agree with the outcomes of the study of the distribution of local minima, where the results suggested lower amplitude of inhibitory presynaptic input compared to the excitation. Additionally, the membrane potential during periods of inhibition, stays relatively close to the resting potential (about -76 – -85 mV), compared to periods of excitation, where we see values of larger scale (about -76 – -50 mV). Moreover, the slower/smoother increase of the variability is linked with presynaptic input of longer duration, which may be possible in this case. The decrease after 0.03 sec and sharper decline after 0.045 sec are due to the long duration of the intervals compared to the duration of the inhibition, which has as a result the exclusion of *Pnew* values from the calculation of the standard deviation.



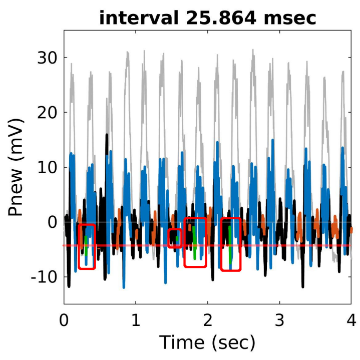
(a) Areas with high Pnew values are indicated by red border.



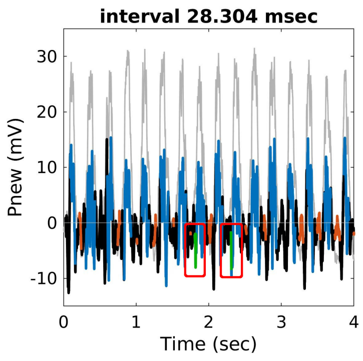
(b) Areas that previously had high Pnew values, are now excluded from the calculation (indicated by red border).



(c) Areas with high Pnew values are indicated by red border.

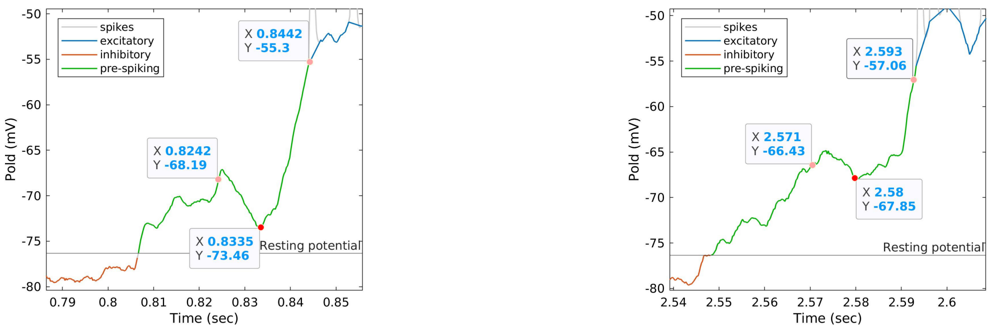


(d) Areas with high Pnew values are indicated by red border.

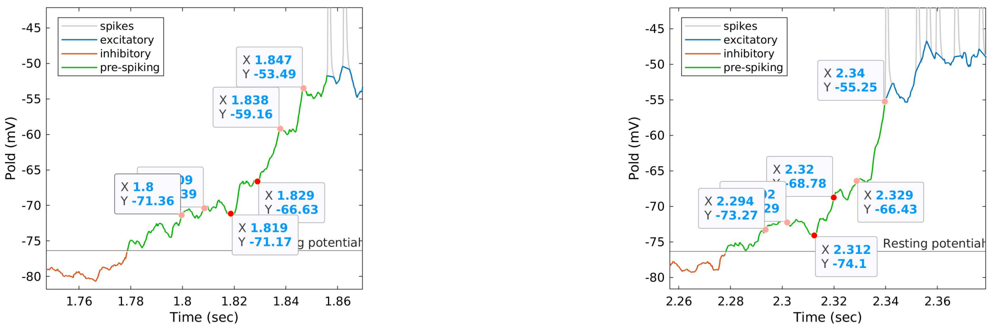


(e) The only values of Pnew that are considered for the pre-spiking period, indicated by red border.

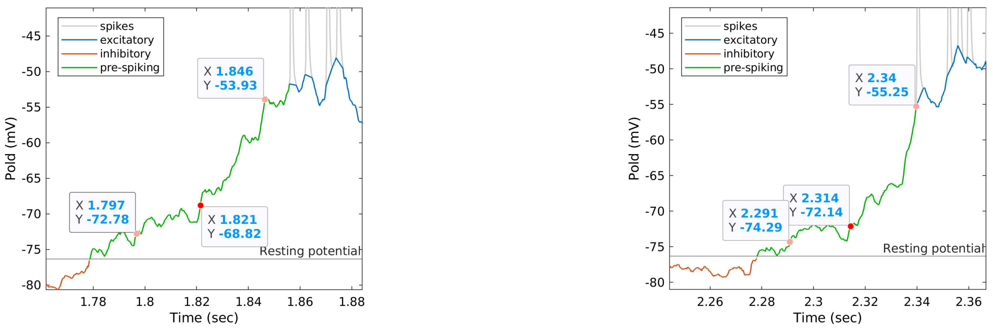
Figure 4.11: Pnew values at the corresponding intervals that cause oscillations of the standard deviation (Fig.4.10). The colored lines represent the different periods (pre-spiking (green), inhibitory (orange), excitatory (blue) and the duration excluded completely from the calculation (black)). The grey lines show the membrane potential.



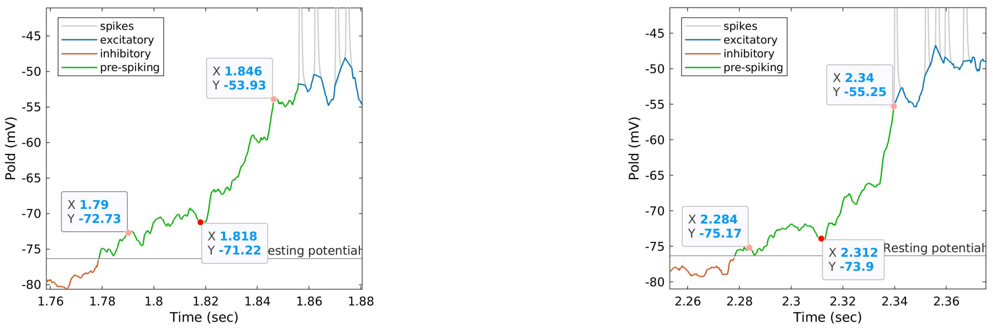
(a) Interval between points 10.004 msec.



(b) Interval between points 18.544 msec.



(c) Interval between points 25.864 msec.



(d) Interval between points 28.304 msec.

Figure 4.12: Close-up of the membrane potential of areas that caused high values of Pnew (see corresponding plots of Fig.4.11). Red dots represent the central points, whose value is compared with the average of the points preceding and succeeding at a certain interval (colored light red).

### Estimating variability of intracellular recordings

Based on the observations of the previous sections and the understanding gained so far about how variability is affected by the different parameters of alpha functions, we tried to estimate real neuron's variability by simulation. The estimated variability is shown with dashed lines in Fig. 4.10. The duration of the experiments was 1 sec. For this reason the estimated pre-spiking variability does not show a decreasing trend. A single Poisson process of alpha functions with constant rate 90 Hz was used in each case. For the pre-spiking period, we used alpha functions of height 3.8 mV and shape parameter of 0.0028 sec. On the other hand, we used wider alpha functions for the estimation of the inhibitory period's variability, shape parameter was set at 0.0045 sec. The height of alpha functions was adjusted at 1.3 mV.

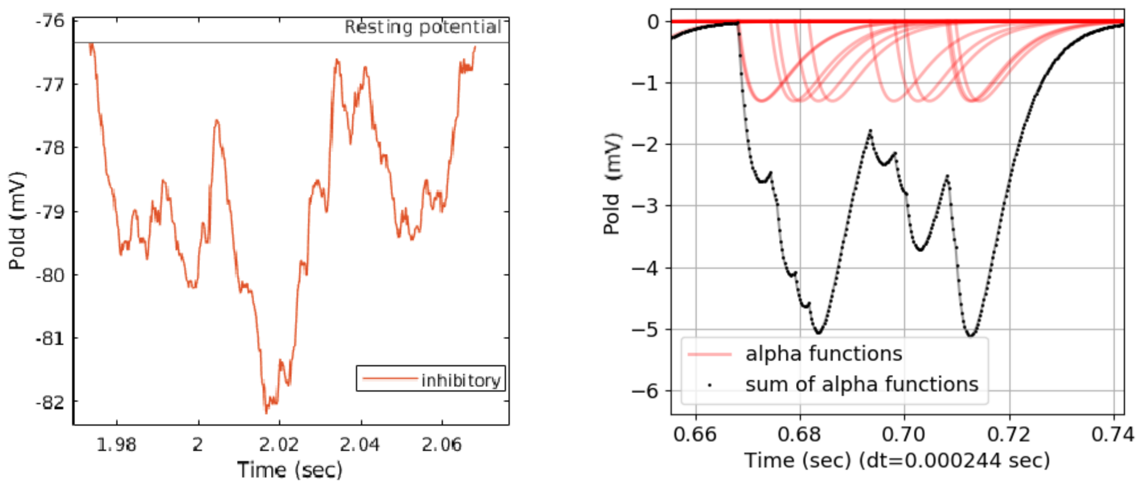


Figure 4.13: Comparison of cell's 61 membrane potential and the estimation of a single Poisson process of alpha functions. (Left) Close-up of inhibitory period of the membrane potential of cell 61. (Right) Close-up of the estimated membrane potential for the inhibitory period (black dotted line), as the sum of alpha functions (red lines). Note that the resting potential of real neuron (left) is -76.3 mV, whereas for the simulation it was set at 0 mV.

A closer look to the estimated potential is shown in Fig. 4.13. The numerical experiment captures the big structures of real neuron's membrane potential. However, smaller, sharper oscillations of the membrane potential are not shown in the estimated data. This explains the lower estimated variability for intervals shorter than 0.005 sec (see Fig. 4.10).

Decreasing the duration of alpha functions had as a result a higher increasing rate of the variability compared to the real data. This observation raises questions on what could be the duration of intervals that are more informative on the presynaptic input parameters. Standard deviation of *Pnew* for longer intervals provides information for the general form of the membrane potential, which can be useful for the understanding the synchrony and rate of the input; an area that has not been extensively studied at the current work. In contrast, shorter intervals reveal more details on the features of individual presynaptic input, such as amplitude and duration.

## 4.4 Discussion

We first approached the problem by applying the NPSS metric on the first spike of each burst and burst-group. In Fig. 3.5 we can see that there is a clear distinction between the two categories of spikes, with the first spike of burst-groups (colored green) having lower firing threshold than the first spikes of bursts (colored red), where the membrane potential is at higher levels. Previous research suggests that "when cells (recordings from visual cortex in vivo) become depolarized, spike threshold will increase to reduce cellular sensitivity to further slow depolarizations while preserving, or even enhancing, the relative sensitivity to rapid depolarizations" [5].

Comparing the results of NPSS applied on the first spike of each group of bursts only and of each burst, we notice a difference in normalized slope values, as well as bigger variation of the bounds in the case of burst-groups. The reason of this variance and as a consequence possible inexact results could be the fact that the slope of the first spike of burst-groups might not represent the actual slope of the membrane potential. According to the definition of the slope, start-point of the slope was set at the last time the membrane potential reaches $V_{rest}$. However, as shown in Fig. 3.3, the actual pre-spiking slope for the first spike of the group (green spike) starts at a later point (about 0.02 sec later) and it is much steeper. Underestimating the pre-spiking slope could result in inaccurate bounds and in considering the contribution of coincidence detection to the firing to be smaller than it really is.

For the specific cell examined so far, in cases of stimulus with angle different than the preferred (and opposite direction) the cell did not fire, so we could not test the NPSS metric and confirm the initial hypothesis. Another question that emerged was the fact that after the first spike of a burst-group the membrane potential stayed high (even higher than the firing-threshold) and the following spikes might result only from one EPSP. If that is the case, the operational mode of the neuron can not be determined. Furthermore, there is a big variation of bound values, as shown in Fig. 4.1b, which might lead to inaccuracies.

In order to learn about real neuron's presynaptic input, we developed two methods: (1) the study of the distribution of membrane potential's local extrema and (2) the measure of variability. Both methods were applied first on simplified numerical experiments, before their application on the real data.

We were able to obtain an estimate of the presynaptic input's amplitude, by fitting a curve of Gaussian bells on the histogram of membrane potential' s local extrema. We noted the sensitivity of confidence bounds to the starting coefficient values and dependency of the method on the clarity of histogram's peaks due to histogram's resolution (bin size). Furthermore, another factor that affects the results and can be considered as almost decisive is the number of fitted Gaussian bells, which is empirically set. After solving these dependencies and ensuring a better accuracy of the results, we could combine the estimated amplitudes with the measure of the variability. Specifically, by normalizing the measured variability with the estimated amplitude from the fitting, we could "remove" the effects of input amplitude and focus on the duration and synchrony of the input.

The histograms were used initially for the determination of the amplitude of the presynaptic input. However, they may also provide information about the synchrony of the input i.e., for synchronized EPSPs, the histogram would have more values on higher membrane potential, because more EPSPs would occur at the same time. Focusing on the histogram of local maxima, we notice that the frequency of values above -65 mV decreases significantly compared to the preceding values. In other words, more than two simultaneous EPSPs occur less frequently. This is also visible from the plot of membrane potential (Fig. 4.9 (top)). The local maxima, marked with green circle, are more dense on lower values of membrane potential. Based on these preliminary results, it will be interesting to examine the potential of the method to extract any characteristics of the synchronization of the presynaptic potentials. A fact that contributes to these results is that the membrane potential tends to rise steeply to the first spike of the burst, after a certain point, without having any peaks along the way. For instance, see burst at 1.3 – 1.47 sec of Fig. 4.9, where the only peaks are concentrated at -70 mV. Such behavior might prevent us from gaining a better understanding of the synchronization, since the method mentioned above relies on the peaks - local maxima and minima of the membrane potential.

Further to our study, we examined membrane potential's variability and managed to approach it using a single Poisson process of alpha functions. The results from the simplified numerical experiments showed a clear correlation between amplitude, duration of presynaptic input and variability, on which the estimation of real neuron's variability was based. However, the amplitude of excitatory and inhibitory input used for the estimation is lower compared to the results from the study of local extrema distribution. As suggested by the fitting of Gaussian bells on the histograms, the amplitude of excitatory and inhibitory input is 4.74 mV and 3.13 mV, respectively. In contrast, for the simulation the corresponding values were 3.8 mV and 1.3 mV. Possible reasons for this discrepancy are the reliance of the fitting method on the number of Gaussian bells, initial coefficient values and resolution of the histogram, that could lead to biased results. Moreover, although the estimated variability through the simulation is close to that of the real neuron, as shown by Fig. 4.13, the simulated potential lacks smaller/sharper oscillations, that exist in the real data. This difference is also reflected as lower variability of the simulated data for short intervals. This is an area that requires to be further examined. Is the lower variability related to the duration of the alpha functions used in the simulation, or could rate and synchrony improve the estimation?

The role of synchrony on variability still remains an open question, since it was not examined to the extent required in order to provide sufficient information and come to a conclusion. Better understanding of the input synchrony could have also been helpful in the definition of more precise coincidence window for the application of the NPSS metric, in order to achieve more accurate results.

# Chapter 5

# Conclusion

## Contents

## 5.1 Conclusion

The work presented in this thesis aimed to provide us with a better understanding of the operation mode of simple cells of the cat primary visual cortex. Moreover, we demonstrated two novel methods, that were used to extract information about the presynaptic input.

We first used the NPSS metric on the experimental data to determine its operation mode. Due to the bursting behavior of the simples cells, the measure was applied on selected spikes i.e., first spikes of burst and groups of bursts. The results suggest that the neuron acts more as a temporal integrator than as a coincidence detector. However, the accuracy of the results is questionable, due to the big variations of the bounds and possible non-representative slope that was taken into account to the calculation. Another question that arises, is that the NPSS is based on the slope of the membrane potential before the spikes, which are the limited or even non-existent in cases of non-preferred stimuli.

Then, the need arose to determine the parameters of the presynaptic input, i.e. amplitude, duration and degree of synchronization, since they are required for the simulation and necessary to understand the neural behavior. We provided preliminary results of the study of the distribution of local extrema, in order to infer the amplitude of the individual presynaptic events, from the postsynaptic membrane potential. More specifically, we studied the distribution of local minima during inhibition and local maxima during the pre-spiking period.

The last part of this thesis is devoted to the calculation and interpretation of the variability of the membrane potential, through simpler experiments. The results showed, that the changes in the amplitude of presynaptic input shift vertically the variability. Furthermore the duration of the individual presynaptic events is related with the initial increase of the variability. The synchrony of the input was discussed briefly and requires to be further examined.

## 5.2 Future Work

The current thesis provides the grounds for future work. We firstly identified some limitations and doubts for the accuracy of the application of the NPSS measure on the experimental data. We may need to reconsider how the slopes are defined, since the slopes especially for the first spike of burst-groups might not be representative. A definition of a more precise coincidence window would also be possible, if more information about the input synchrony could be acquired using the proposed methods.

The study of the distribution of local extrema of the membrane potential was initially used to determine the amplitude of the presynaptic input. Future work could attempt to eliminate the dependency of the method to empirically set factors, namely the number of fitted Gaussian bells, initial coefficient values and histogram's resolution. Moreover, the method could be combined with the measure of variability, by normalizing the variability with the estimated input amplitude and using the measure for the determination of input duration and synchrony. In addition, as mentioned in previous chapters, the histograms could provide us with information about the synchrony of the input, as well. The statistical properties of the distributions could be further examined and any potential outcome could be used to the completion of the simulation.

So far, only the membrane potential of the preferred orientation was used in the methodology. Applying the measure of variability and studying the distribution of the local extrema of the membrane potential for non-preferred stimuli could be valuable to the understanding of the characteristics of the presynaptic input and of cells behavior in general.

The effects of input's synchrony on the variability of the postsynaptic membrane potential is another point that requires further work. The use of inhomogeneous Poisson processes or added noise to the timing of the events could be a possible part of the subsequent work.

At last, after acquiring enough information about the parameters of the input and the neuron itself, we could use the simulation to verify the results of the NPSS metric and come to a conclusion about the operational mode of the neuron. At that point, we could test other existing metrics for the same purpose, i.e., the neural mode and drive [24] and compare their suitability and accuracy for the current experimental data.

# References

[1] M. Abeles. Role of the cortical neuron: integrator or coincidence detector? *Israel Journal of Medical Sciences*, 18(1):83, 1982.

[2] E. D. Adrian and Y. Zotterman. The impulses produced by sensory nerve endings. *The Journal of Physiology*, 61(4):465–483, 1926.

[3] J. S. Anderson, M. Carandini, and D. Ferster. Orientation tuning of input conductance, excitation, and inhibition in cat primary visual cortex. *Journal of Neurophysiology*, 84(2):909–926, 2000.

[4] R. Azouz and C. M. Gray. Dynamic spike threshold reveals a mechanism for synaptic coincidence detection in cortical neurons in vivo. *Proceedings of the National Academy of Sciences*, 97(14):8110–8115, 2000.

[5] R. Azouz and C. M. Gray. Adaptive coincidence detection and dynamic gain control in visual cortical neurons in vivo. *Neuron*, 37(3):513 – 523, 2003.

[6] W. Bair and C. Koch. Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey. *Neural Computation*, 8(6):1185–1202, 1996.

[7] A. Bell, Z. F. Mainen, M. Tsodyks, and T. J. Sejnowski. Balancing" of conductances may explain irregular cortical spiking. *La Jolla, CA: Institute for Neural Computation Technical Report INC-9502*, 1995.

[8] O. Bernander, R. J. Douglas, K. Martin, and C. Koch. Synaptic background activity influences spatiotemporal integration in single pyramidal cells. *Proceedings of the National Academy of Sciences*, 88(24):11569–11573, 1991.

[9] T. H. Brown, E. W. Kairiss, and C. L. Keenan. Hebbian synapses: biophysical mechanisms and algorithms. *Annual Review of Neuroscience*, 13(1):475–511, 1990.

[10] G. Bugmann. Determination of the fraction of active inputs required by a neuron to fire. *Biosystems*, 89(1-3):154–159, 2007.

[11] G. Bugmann, C. Christodoulou, and J. G. Taylor. Role of temporal integration and fluctuation detection in the highly irregular firing of a leaky integrator neuron model with partial reset. *Neural Computation*, 9(5):985–1000, 1997.

[12] M. Carandini and D. Ferster. Membrane potential and firing rate in cat primary visual cortex. *Journal of Neuroscience*, 20(1):470–484, 2000.

[13] M. Carandini, F. Mechler, C. S. Leonard, and J. A. Movshon. Spike train encoding by regular-spiking cells of the visual cortex. *Journal of Neurophysiology*, 76(5):3425–3441, 1996.

[14] B. Chapman, K. R. Zahs, and M. P. Stryker. Relation of cortical cell orientation selectivity to alignment of receptive fields of the geniculocortical afferents that arborize within a single orientation column in ferret visual cortex. *Journal of Neuroscience*, 11(5):1347–1358, 1991.

[15] P. Dayan and L. F. Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT Press, Cambridge, MA, 2001.

[16] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1(1):1–47, 1991.

[17] R. Fredericksen, F. Verstraten, and W. Van De Grind. An analysis of the temporal integration mechanism in human motion perception. *Vision Research*, 34(23):3153–3170, 1994.

[18] J. L. Gardner, A. Anzai, I. Ohzawa, and R. D. Freeman. Linear and nonlinear contributions to orientation tuning of simple cells in the cat's striate cortex. *Visual Neuroscience*, 16(6):1115–1121, 1999.

[19] K. D. Harris and A. Thiele. Cortical state and attention. *Nature Reviews Neuroscience*, 12(9):509–523, 2011.

[20] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[21] J. A. Hirsch, J.-M. Alonso, R. C. Reid, and L. M. Martinez. Synaptic integration in striate cortical simple cells. *Journal of Neuroscience*, 18(22):9517–9528, 1998.

[22] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[23] J. P. Jones and L. A. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1233–1258, 1987.

[24] J. Kanev, A. Koutsou, C. Christodoulou, and K. Obermayer. Integrator or coincidence detector: A novel measure based on the discrete reverse correlation to determine a neuron's operational mode. *Neural Computation*, 28(10):2091–2128, 2016.

[25] H. Kato, P. Bishop, and G. Orban. Hypercomplex and simple/complex cell classifications in cat striate cortex. *Journal of Neurophysiology*, 41(5):1071–1095, 1978.

[26] P. König, A. K. Engel, and W. Singer. Integrator or coincidence detector? the role of the cortical neuron revisited. *Trends in Neurosciences*, 19(4):130–137, 1996.

[27] A. Koutsou, C. Christodoulou, G. Bugmann, and J. Kanev. Distinguishing the causes of firing with the membrane potential slope. *Neural Computation*, 24(9):2318–2345, 2012.

[28] L. Krubitzer. The magnificent compromise: cortical field evolution in mammals. *Neuron*, 56(2):201–208, 2007.

[29] P. Lansky, P. Sanda, and J. He. The parameters of the stochastic leaky integrate-and-fire neuronal model. *Journal of Computational Neuroscience*, 21(2):211–223, 2006.

[30] P. Lansky, P. Sanda, and J. He. Effect of stimulation on the input parameters of stochastic leaky integrate-and-fire neuronal model. *Journal of Physiology-Paris*, 104(3-4):160–166, 2010.

[31] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997.

[32] M. E. Mazurek, J. D. Roitman, J. Ditterich, and M. N. Shadlen. A role for neural integrators in perceptual decision making. *Cerebral Cortex*, 13(11):1257–1269, 2003.

[33] J. A. Movshon, I. D. Thompson, and D. J. Tolhurst. Receptive field organization of complex cells in the cat's striate cortex. *The Journal of Physiology*, 283(1):79–99, 1978.

[34] B. R. Payne and A. Peters. The concept of cat primary visual cortex. In B. R. Payne and A. Peters, editors, *The Cat Primary Visual Cortex*, pages 1 – 129. Academic Press, San Diego, 2002.

[35] X. Pei, T. Vidyasagar, M. Volgushev, and O. Creutzfeldt. Receptive field analysis and orientation selectivity of postsynaptic potentials of simple cells in cat visual cortex. *Journal of Neuroscience*, 14(11):7130–7140, 1994.

[36] R. C. Reid and J.-M. Alonso. Specificity of monosynaptic connections from thalamus to visual cortex. *Nature*, 378(6554):281–284, 1995.

[37] M. N. Shadlen and W. T. Newsome. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology*, 4(4):569–579, 1994.

[38] M. N. Shadlen and W. T. Newsome. The variable discharge of cortical neurons: implications for connectivity, computation, and information coding. *Journal of Neuroscience*, 18(10):3870–3896, 1998.

[39] S. M. Sherman and R. Guillery. On the actions that one nerve cell can have on another: distinguishing "drivers" from "modulators". *Proceedings of the National Academy of Sciences*, 95(12):7121–7126, 1998.

[40] W. R. Softky. Simple codes versus efficient codes. *Current Opinion in Neurobiology*, 5(2):239–247, 1995.

[41] W. R. Softky and C. Koch. The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps. *Journal of Neuroscience*, 13(1):334–350, 1993.

[42] M. Stimberg, R. Brette, and D. F. Goodman. Brian 2, an intuitive and efficient neural simulator. *Elife*, 8:e47314, 2019.

[43] H. C. Tuckwell. *Introduction to theoretical neurobiology: volume 2, nonlinear and stochastic theories*, volume 8. Cambridge University Press, 1988.

[44] R. R. d. R. van Steveninck, G. D. Lewen, S. P. Strong, R. Koberle, and W. Bialek. Reproducibility and variability in neural spike trains. *Science*, 275(5307):1805–1808, 1997.

[45] B. Wandell, S. Dumoulin, and A. Brewer. Visual cortex in humans. *Encyclopedia of Neuroscience*, 10:251–257, 2009.

[46] F. Worgotter and C. Koch. A detailed model of the primary visual pathway in the cat: comparison of afferent excitatory and intracortical inhibitory connection schemes for orientation selectivity. *Journal of Neuroscience*, 11(7):1959–1979, 1991.

[47] C. Zavou, A. Kkoushi, A. Koutsou, and C. Christodoulou. Synchrony measure for a neuron driven by excitatory and inhibitory inputs and its adaptation to experimentally-recorded data. *Biosystems*, 161:46 – 56, 2017.

# Appendix A

# Source code

The following source code was implemented using the following tools:

- MATLAB version R2020a
- Python version 3.7.6
- Anaconda version 4.8.3
- Brian version 2.3

## A.1 NPSS

Listing A.1: NPSS.m - Implementation of the NPSS metric.

```
function [m, M] = NPSS(str_values, end_values, ISIs, bin, shiftUpBound, shiftLowBound)
% Calculate NPSS values.
%
% Parameters
% ----------
%   str_values:    str_values(:,1) the time of each start-point (sec).
%                  str_values(:,2) the value of each start-point (Volts).
%   end_values:    end_values(:,1) the time of each end-point (sec).
%                  end_values(:,2) the value of each end-point (Volts).
%   ISIs:          inter-spike intervals for each spike (sec).
%   bin:           size of bin in order to calculate upper/lower bounds per isi
%                  size (sec).
%   shiftUpBound:  vertical shifting of the upper bound.
%   shiftLowBound: vertical shifting of the lower bound.
%
% Returns
% -------
%   m:             vector of slope values for each spike.
%   M:             vector of normalized slope values for each spike using the
%                  bounds from polynomial fitted function.

    %% Calculate slope
    m = (end_values(:,2) - str_values(:,2))./(end_values(:,1) - str_values(:,1));

    %% Calculate upper and lower bounds
    [lowBounds, upBounds, polyfit_lowBounds, polyfit_upBounds, timeslots] =
        UpperLowBounds(ISIs, m, bin, shiftUpBound, shiftLowBound);

    %% Normalize slopes - Calculate NPSS
    M = [];% vector of normalized slope for each spike
    slots = [];
    for i = 1:length(m)
        % find in which timeslot does the current ISI lie
        temp = find(timeslots >= ISIs(i),1);
        if length(temp) == 0
            temp = length(timeslots)+1;
        end
        slots = [slots;temp-1];
    end

    L = [];U = [];
    for i = 1:length(slots)
        L = [L;polyfit_lowBounds(slots(i))]; % lower bound for each spike
        U = [U;polyfit_upBounds(slots(i))];  % upper bound for each spike
    end

    M = (m-L)./(U-L);
end

function [lowBounds, upBounds, polyfit_lowBounds, polyfit_upBounds, timeslots] =
    UpperLowBounds(ISIs, m, bin, shiftUpBound, shiftLowBound)
    %% Histogram of ISI and slopes
    split = []; lowBounds = []; upBounds = [];
    max_isi = max(ISIs); max_isi = round(ceil(max_isi*1e2)) / 1e2;

    timeslots = 0:bin:max_isi;

    binCounts = []; counts = zeros(1,length(timeslots));
    for i = 1:length(timeslots)
        % separate all gradients in timeslot ranges where they lie into
        temp = find(ISIs >= ((i-1)*bin) & ISIs<(i*bin));
        if length(temp)==0
            timeslots(i) = -1;
            counts(i) = 0;
        else
            split{i} = m(temp);
            lowBounds = [lowBounds;min(m(temp))]; % lowest slope in the specific bin
            upBounds = [upBounds;max(m(temp))]; % highest slope in hte specific bin
            binCounts = [binCounts,repmat((i-1)*bin,1,length(temp))];
            counts(i) = length(temp);
        end
    end
    temp = find(timeslots>-1);
    timeslots = timeslots(temp);

    %% Initial bounds
    bins_num = length(timeslots);
    figure; box;
    subplot(2,1,1);
    hold on;
    plot(timeslots,upBounds,'bo');
    plot(timeslots,lowBounds,'r+');
    hold off;
    legend('upper␣bound','lower␣bound');
    xlabel('ISI␣length[s]'); ylabel('slope␣bounds');

    subplot(2,1,2);
    hist(binCounts,bins_num, 'b');
    xlim([-inf inf]); ylim([-inf inf]);
    xlabel('ISI␣length[s]'); ylabel('frequency');
    axis([0,max_isi,0,max(counts)]);

    %% Fit 3rd degree polynomial
    polyfit_upper = polyfit(timeslots', upBounds, 3);
    polyfit_low = polyfit(timeslots', lowBounds, 3);

    polyfit_upBounds = polyval(polyfit_upper, timeslots);
    polyfit_lowBounds = polyval(polyfit_low, timeslots);

    % Shift fitted function so as to enclose the majority of slope values
    polyfit_upBounds = polyfit_upBounds + shiftUpBound;
    polyfit_lowBounds = polyfit_lowBounds + shiftLowBound;

    figure('Position', [10 10 8 10]); box; hold on;
    % Plot bounds
    p1 = plot(timeslots,upBounds,'bo');
    p2 = plot(timeslots,lowBounds,'r+');

    p3 = plot(timeslots,polyfit_upBounds,'k');
    p4 = plot(timeslots,polyfit_lowBounds,'--k');

    xlim([-inf inf]); ylim([-inf inf]);
    l = legend([p1(1), p2(1), p3(1), p4(1)], 'upper␣bounds␣per␣isi', 'lower␣bounds␣per␣
        isi', 'fitted␣function␣of␣upper␣bounds', 'fitted␣function␣of␣lower␣bounds');
    set(l, 'Location','southoutside');
    xlabel('length␣of␣ISI␣(sec)'); ylabel('pre-spike␣slope␣bounds');
    hold off;
end
```

Listing A.2: bursts_npss.m - Application of the NPSS metric on real data.

```
GLOBALS;
close all; clear all;
```

```matlab

%% Load selected cell and orientation data
icell = 61;
[time ,potential, Vth, Vrest, samplerate, nsamples, spikes, spikeheight, spikeduration, istim, iexp] = loadcellinfo(icell,270);
timestep = 1/samplerate;

%% Plot membrane potential for the total duration
figure('Position', [10 10 21 7]); clf; hold on;
plot(time, potential);
plot(time, ones(size(time))*Vth);
plot(time, ones(size(time))*Vrest);
plot(time, ones(size(time))*spikeheight);
hold off;
xlim([-inf inf]); ylim([-inf inf]);
xlabel('Time (sec)'); ylabel('Membrane potential (mV)');
legend('membrane potential', 'spike threshold', 'resting potential', 'spike height');

%% Define Bursts
[t_bursts, t_burst_groups, index_pre_burst_spikes, index_burst_groups, in_bursts] = detect_bursts(potential, spikes, timestep);
sprintf("Number of bursts found: %d", size(t_bursts,2))

% Plot membrane potential showing the beginning of each burst
figure('Position', [10 10 21 7]); clf; hold on;
p1 = plot(time, potential); hold on

p2 = plot([t_bursts; t_bursts], repmat(ylim',1,size(t_bursts,2)), '-r');
p3 = plot([t_burst_groups; t_burst_groups], repmat(ylim',1,size(t_burst_groups,2)), '-g');

p4 = plot(time, ones(size(time))*Vrest, 'k:');
p5 = plot(time, ones(size(time))*Vth,  'r:');

%% Find start/end *index-time-point* to be used to calculate the slope
% For burst_group : last time when potential is at the resting potential
% before a spike, until time when reaches threshold (and spike follows)
% For in_bursts: use the minimum potential until the spike

% Start index of slope (for burst groups)
str_index_burst_groups = [];
for i = 1:numel(index_burst_groups)
    str_index_burst_groups(end+1) = find(potential(1:index_burst_groups(i)) < Vrest, 1, 'last') + 1;
end

% End index of slope (for burst groups)
end_index_burst_groups = [];
```

```
for i = 1:numel(str_index_burst_groups)
    end_index_burst_groups(end+1) = find(potential(str_index_burst_groups(i):end) > Vth
        , 1, 'first') + str_index_burst_groups(i) − 1;
end

% Find bursts index, excluding the beginning of each burst-group
intersect_spikes = intersect(in_bursts,index_burst_groups);
index_bursts_ingroup = setxor(in_bursts,intersect_spikes);

% Find previous spike of those in in_bursts_only
tmp = ismember(spikes, index_bursts_ingroup);
temp_index = find(tmp);
index_bursts_ingroup_pre = spikes(temp_index − 1);

str_index_bursts_ingroup = [];
end_index_bursts_ingroup = [];
for i = 1:numel(index_bursts_ingroup)
    % Start index of slope (for bursts IN groups) - minimum between the spikes
    % Find the minimum potential value between the spikes
    % min(potential(index_bursts_ingroup_pre(i) + 1:index_bursts_ingroup(i)) to find
        the minimum after the previous spike
    str_index_bursts_ingroup(end+1) = find(potential(index_bursts_ingroup_pre(i):
        index_bursts_ingroup(i)) == min(potential(index_bursts_ingroup_pre(i)+1:
        index_bursts_ingroup(i))), 1, 'last') + index_bursts_ingroup_pre(i) − 1;


    % End index of slope (for bursts in groups)
    if potential(str_index_bursts_ingroup(i)) > Vth % if start is above Vth set end to
        the time of spike as found by Carantini
        end_index_bursts_ingroup(i) = index_bursts_ingroup(i);
    else
        end_index_bursts_ingroup(end+1) = find(potential(str_index_bursts_ingroup(i):
            index_bursts_ingroup(i) + 1) < Vth, 1, 'last') + str_index_bursts_ingroup(i
            ) − 1;
    end
end

%% Find start/end *values* of potential
% For burst_group: set start/end potential values to Vrest and Vth, respectively
% For in_bursts: start value is set as it is

str_values_burst_groups = [time(str_index_burst_groups) Vrest(ones(size(
    str_index_burst_groups, 2),1))];
end_values_burst_groups = [time(end_index_burst_groups) Vth(ones(size(
    end_index_burst_groups, 2),1))];

str_values_bursts_ingroup = [time(str_index_bursts_ingroup) potential(
    str_index_bursts_ingroup)];
```

```
% If start value is under the Vth end value is set to Vth otherwise,
% is left as it is found by Carandini
end_values_bursts_ingroup = [time(end_index_bursts_ingroup) Vth(ones(size(
    end_index_bursts_ingroup, 2),1))]; % set all to Vth
end_values_bursts_ingroup(find(potential(str_index_bursts_ingroup) > Vth), 2) =
    potential(end_index_bursts_ingroup(find(potential(str_index_bursts_ingroup) > Vth))
    );

p6 = plot(str_values_burst_groups(:, 1),str_values_burst_groups(:, 2), 'r*');
p7 = plot(end_values_burst_groups(:, 1),end_values_burst_groups(:, 2), 'r*');
p8 = plot(str_values_bursts_ingroup(:, 1),str_values_bursts_ingroup(:, 2), 'r*');
p9 = plot(end_values_bursts_ingroup(:, 1),end_values_bursts_ingroup(:, 2), 'r*');

xlim([-inf inf]); ylim([-90 0]); box on;
l = legend([p1(1), p2(1), p3(1), p4(1), p5(1), p6(1)], 'membrane potential', 'burst', '
    burst-group', 'resting potential', 'spike threshold', 'start/end point slope');
xlabel('Time (sec)'); ylabel('Membrane potential (mV)');
hold off;

%% Merge results for all the bursts
str_values_all_bursts = sortrows([str_values_burst_groups; str_values_bursts_ingroup]);
end_values_all_bursts = sortrows([end_values_burst_groups; end_values_bursts_ingroup]);

%% Calculate ISI between bursts
% Define window size as the time at Vth between the end and start of the
% spikes.
%
% in_bursts: index of all the burst-spikes found by Carandini
% index_pre_burst_spikes is the index of the previous spikes of each burst.

% 'Fix' index of spike in case the membrane potential at the time is below Vth
index_pre_burst_spikes_Vth = [1];
for i = 2:numel(index_pre_burst_spikes)
    index_pre_burst_spikes_Vth(end + 1) = find(potential(index_pre_burst_spikes(i):end)
        > Vth, 1, 'first') + index_pre_burst_spikes(i) - 1;
end

% Start of ISI
isi_start=[1]; % for the first spike of the sequence
for i = 2:numel(in_bursts)
    p = find(potential(index_pre_burst_spikes_Vth(i):in_bursts(i)) < Vth, 1, 'first');
    if length(p) == 0 % in case p is empty
        p = find(potential(index_pre_burst_spikes_Vth(i):in_bursts(i)) < potential(
            index_pre_burst_spikes_Vth(i)), 1, 'first');
    end
    isi_start = [isi_start;(index_pre_burst_spikes_Vth(i) + p - 1)];
end

```

```
% End of ISI
isi_end = int64(end_values_all_bursts(:,1)/timestep); %in_bursts;

ISIs = time(isi_end)-time(isi_start);

%% NPSS for all bursts

[m, M] = NPSS(str_values_all_bursts ./ [1 1000], end_values_all_bursts ./ [1 1000], ...
            ISIs, 0.008, 0.6, -0.25);

% Plot NPSS/slope
figure('Position', [10 10 8 10]);
subplot(2,1,1);plot(time(in_bursts),m,'.');ylabel('slope');
subplot(2,1,2);plot(time(in_bursts),M,'.');ylabel('Normalised␣Slope/M');
xlabel('Time␣in␣record[s]');

figure('Position', [10 10 8 10]);boxplot(M);
xlabel(''); ylabel('Normalised␣Slope/M');

%% NPSS for burst-groups only
[val,index] = (intersect(in_bursts, index_burst_groups));

[m, M] = NPSS(str_values_burst_groups ./ [1 1000], end_values_burst_groups ./ [1 1000],
    ...
            ISIs(index), 0.025, 0, 0);

% Plot NPSS/slope
figure('Position', [10 10 8 10]);box;
subplot(2,1,1);plot(time(index_burst_groups),m,'.');ylabel('slope');
subplot(2,1,2);plot(time(index_burst_groups),M,'.');ylabel('Normalised␣Slope/M');
xlabel('Time␣in␣record[s]');

figure('Position', [10 10 8 10]);boxplot(M);
xlabel(''); ylabel('Normalised␣Slope/M');

%% NPSS for bursts in group only
[val,index] = (intersect(in_bursts, index_bursts_ingroup));

[m, M] = NPSS(str_values_bursts_ingroup ./ [1 1000], end_values_bursts_ingroup ./ [1
    1000], ...
            ISIs(index), 0.008, 0, 0);

% Plot NPSS/slope
figure('Position', [10 10 8 10]);
subplot(2,1,1);plot(time(index_bursts_ingroup),m,'.');ylabel('slope');
subplot(2,1,2);plot(time(index_bursts_ingroup),M,'.');ylabel('Normalised␣Slope/M');
xlabel('Time␣in␣record[s]');

```

```
figure('Position', [10 10 8 10]);boxplot(M);
xlabel(''); ylabel('Normalised␣Slope/M');

%% Plot of spikes
tt = [-50:19];
t0 = 51;

pots = NaN*ones(length(tt),1);
pots_group = NaN*ones(length(tt),1);
pots_IN_group = NaN*ones(length(tt),1);
nsamples = size(potential,1);
for spike = spikes(:)'
    samples = spike + tt;
    samples(find(samples>nsamples)) = nsamples;
    samples(find(samples<1)) = 1;
    pots(:,end+1) = potential(samples); % all spikes
    if ismember(spike, double(index_burst_groups(:)'))
        pots_group(:,end+1) = potential(samples); % first spike of group
    elseif ismember(spike, double(index_bursts_ingroup(:)'))
        pots_IN_group(:,end+1) = potential(samples); % first spike of burst
    end
end

% then drop the first column (they are NaNs)
if size(pots,2)>1
    pots = pots(:,2:end);
end

if size(pots_group,2)>1
    pots_group = pots_group(:,2:end);
end

if size(pots_IN_group,2)>1
    pots_IN_group = pots_IN_group(:,2:end);
end

meds = median(pots, 2 );

figure('Position', [10 10 8 10]); clf; hold on;
ttms = [tt]*1000/samplerate;
p1 = plot( ttms, [pots], 'color', 0.8*[1 1 1] ); hold on % grey
p2 = plot( ttms, [pots_group], 'color', 0.8*[0 1 0] ); hold on % green
p3 = plot( ttms, [pots_IN_group], 'color', 0.8*[1 0 0] ); hold on % red
p4 = plot( ttms, [meds], 'linewidth',3, 'color', 'k' );
box on;

xlim([-inf inf]); ylim([-inf inf]);
l = legend([p1(1), p2(1), p3(1), p4(1)], 'all␣spikes', 'first␣spike␣of␣burst-group', '
    first␣spike␣of␣burst', 'median␣membrane␣potential');
set(1, 'Location','southoutside');
xlabel('Time␣(msec)'); ylabel('Membrane␣potential␣(mV)');
```

Listing A.3: detect_bursts.m - Utility function that returns the index of bursts and burst groups.

```
function [t_bursts, t_burst_groups, index_pre_burst_spikes, index_burst_groups,
    in_bursts] = detect_bursts(potential, spikes, timestep)

    % Empirically choose a time period so that close spikes are considered a burst.
    b = 0.007; % sec maximum time between spikes in a burst
    b_group = 0.05; % sec maximum time between spikes in a burst-group[!]
    t_spikes = spikes*timestep; % multiply index of spikes with time step

    t_bursts = [t_spikes(1)]; % set first burst to the first spike
    t_burst_groups = [t_spikes(1)]; % set first burst to the first spike
    previous_spike = t_spikes(1);
    index_pre_burst_spikes = [1]; % set index of first pre spike at 1

    % Find beginning of each burst
    for spike = t_spikes(2:end)'
        % Find beginning of a burst
        if spike-previous_spike > b
            t_bursts(end+1) = spike;
            index_pre_burst_spikes(end +1) = int64(previous_spike/timestep);
        end

        % Find beginning of a burst-group
        if spike-previous_spike > b_group
            t_burst_groups(end+1) = spike;
        end
        previous_spike = spike;
    end

    index_burst_groups = int64(t_burst_groups(:)/timestep); % index of burst group
    in_bursts = int64(t_bursts(:)/timestep); % index of burst

end
```

## A.2 Proposed Methods

### A.2.1 Distribution of local extrema

Listing A.4: peaks.m - Application of the method on real data.

```matlab
GLOBALS;
close all; clear all;

%% Get cell's data
[time, potential, Vth, Vrest, samplerate, nsamples, spikes, spikeheight, spikeduration,
    istim, iexp] = loadcellinfo(61,270);
timestep = 1/samplerate;

%% Local minima for inhibitory period
min_indx = intersect(find(islocalmin(potential)), find(potential<Vrest));
localmin = potential(min_indx);

%% Local maxima for prespiking period

% Pre-spiking period
prespiking_index = [];

if spikes
    [t_bursts, t_burst_groups, index_pre_burst_spikes, index_burst_groups, in_bursts] =
        detect_bursts(potential, spikes, timestep);
    sprintf("Number of bursts found: %d", size(t_bursts,2))

    % Start index of slope
    preburst_rest_index = []; % index before burst at resting potential
    for i = 1:numel(index_burst_groups)
        preburst_rest_index(end+1) = find(potential(1:index_burst_groups(i)) < Vrest,
            1, 'last') + 1;
    end

    % In case the potential is above thresh before the first burst
    % insert at the beginning of the list 0
    if preburst_rest_index(1) > index_burst_groups(1)
        preburst_rest_index = [1 preburst_rest_index];
    end

    for i = 1:min(length(preburst_rest_index),length(index_burst_groups))
        prespiking_index = [prespiking_index, preburst_rest_index(i):index_burst_groups
            (i)-1];
    end
end

[localmax,max_idx] = findpeaks(potential);
max_idx = intersect(max_idx, prespiking_index);

%% Plot results
figure('Renderer', 'painters', 'Position', [5 10 35 20]);

```

```
% Membrane potential
subplot(2,2,[1,2]);
p1 = plot(time(max_idx), potential(max_idx),'o', 'MarkerSize', 4,'MarkerEdgeColor','g')
    ; hold on;
p2 = plot(time(min_indx), potential(min_indx),'o', 'MarkerSize', 4, 'MarkerEdgeColor','
    r');
p3 = plot(time, potential,'k-');
title('Cell␣61␣-␣Preferred␣Orientation')
xlabel('Time␣(sec)'); ylabel('Membrane␣Potential␣(mV)');
legend([p1(1), p2(1), p3(1)], 'local␣maxima', 'local␣minima', 'mem␣potential');
xlim([0 4.1])
ylim([-90 5])

bins = 25; % number of bins for histograms

%% Local minima
subplot(2,2,3);

[counts, bin_edges] = histcounts(potential(min_indx), 'BinWidth', 0.5);
centers = bin_edges(1:end-1)+mean(diff(bin_edges))/2;

bar(centers, counts, 'r');alpha(.2);
title({'Histogram␣of␣local␣minima␣values', 'of␣inhibitory␣period'});
xlabel('Value␣(mV)'); ylabel('Frequency');

% Fit Gaussian bells on histogram for pre-spiking period
x = centers; y = counts;

ft = fittype(['h1␣*␣exp(-(x␣-␣(a␣-␣1*b)).^2/w1)␣+␣' ...
              'h2␣*␣exp(-(x␣-␣(a␣-␣2*b)).^2/w2)␣+␣' ...
              'h3␣*␣exp(-(x␣-␣(a␣-␣3*b)).^2/w3)']);

% Print coefficient names
coeffnames(ft);

% Guess values to start with.
% [a, b, h1, ..., h3, w1, ..., w3]
startPoints = [Vrest, 3, 100, 70,1.5, 4, 2, 1.5];
Lower_bounds = [-80, 0, 50, 25, 5,0.5, 0.5, 0.5];
Upper_bounds = [-70, 6, 200, 200, 200, 8, 8, 8];

% Now the next line is where the actual model computation is done.
[curvefit,gof,output] = fit(x', y', ft, ...
                            'Lower', Lower_bounds,...
                            'Upper', Upper_bounds,...
                            'Start', startPoints)

hold on;
```

```
plot(curvefit,x', y','predfunc');
ylim([0, 170])
xlabel('Value (mV)');
ylabel('Frequency');

%% Local maxima
subplot(2,2,4);
[counts, bin_edges] = histcounts(potential(max_idx), 'BinWidth', 1);
centers = bin_edges(1:end-1)+mean(diff(bin_edges))/2;

bar(centers, counts, 'g');alpha(.2);
title({'Histogram of local maxima values', 'of pre-spiking period'});
xlabel('Value (mV)'); ylabel('Frequency');

% Fit Gaussian bells on histogram for pre-spiking period
x = centers; y = counts;

ft = fittype(['h1 * exp(-(x - (a + 1*b)).^2/w1) + ' ...
              'h2 * exp(-(x - (a + 2*b)).^2/w2) + ' ...
              'h3 * exp(-(x - (a + 3*b)).^2/w3) + ' ...
              'h4 * exp(-(x - (a + 4*b)).^2/w4) + ' ...
              'h5 * exp(-(x - (a + 5*b)).^2/w5)']);

% Print coefficient names
coeffnames(ft);

% Guess values to start with.
% [a, b, h1, ..., h5, w1, ..., w5]
startPoints =  [Vrest, 4.5, 17, 31, 27, 10, 10, 1.5, 1.5, 1.5, 1.5, 1.5];
Lower_bounds = [-78, 3, 1, 1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5, 0.5];
Upper_bounds = [-74, 6, 45, 45, 45, 45, 10, 10, 10, 10, 10, 10];

% Now the next line is where the actual model computation is done.
[curvefit,gof,output] = fit(x', y', ft, ...
                            'Lower', Lower_bounds,...
                            'Upper', Upper_bounds,...
                            'Start', startPoints)

hold on;
plot(curvefit,x', y','predfunc');
ylim([0, inf])
xlabel('Value (mV)');
ylabel('Frequency');
```

### A.2.2 Measure of variability

#### Validation - Numerical Experiments

Listing A.5: variability_experiments.py - Simulations for each input parameter to study the effects on variability.

```
import matplotlib.pyplot as plt
import math
import random
import params as pms
import utils as local_utl

from scipy import signal

import pandas as pd
import numpy as np
np.random.seed(0)
random.seed(0)

plt.rcParams.update({'font.size': 13, 'axes.titlesize': 13, 'axes.grid': True})


def get_alpha_fun(t, a, scale_fac):
    """Returns a single alpha function based on the given parameters.

    Args:
        t (list of float): time.
        a (float): shape parameter alpha (time to reach the peak).
        scale_fac (float): height of the alpha function.

    Returns:
        list of float: alpha function.
    """

    f = (t/a)*np.exp(-t/a)
    return scale_fac*f/np.max(f)


def poisson_process(num, rate, dur, dt, t0=0.0, p=None):
    """Returns the time of events in a Poisson process.

    Reference: https://stackoverflow.com/questions/1155539/how-do-i-generate-a-poisson-
        process
    Note: inter-arrival times are exponentially distributed with mean 1/rate

    Args:
        num (int): number of events.
        rate (float): rate.
        dur (float): duration.
        dt (float): time step.
        t0 (float, optional): initial time. Defaults to 0.0.
        p (list of float, optional): probability of each time step to exclude
            the event. Defaults to None.

    Returns:
        list of float:  time of events.
    """

    if p is None:
        p = np.ones(int(dur/dt))

    e = []  # time of events
    for i in range(1, int(num)):
        # exponentially distributed random numbers
        t0 += random.expovariate(rate)
        if t0 > dur:
            break
        e.append(t0)

    e = np.array(e)  # convert e to np array
    e_i = np.array(e/dt, dtype=int)  # index of events
    #e_i = e_i[np.where(np.random.binomial(1, p[e_i], e_i.shape[0]))]
    return e_i


def alpha_wave(a, scale_fac, intervals, dt, spike_index_per_input=None, rate=100.0,
               pold=None, dur=1.0, num_input=1, p=None, plot_results=True):
    """Generates a signal of alpha functions based on the given parameters.

    Args:
        a (float): shape parameter alpha (time to reach the peak).
        scale_fac (float): height of the alpha function.
        intervals (list of float): intervals to be used for the variability
            measure.
        dt (float): time step.
        spike_index_per_input (list of int, optional): indices of each alpha
            function. Defaults to None.
        rate (float, optional): Rate of the Poisson process. Defaults to 100.
        pold (list of float, optional): Initial Pold. Defaults to None.
        dur (float, optional): duration. Defaults to 1.0.
        num_input (int, optional): number of presynaptic inputs. Defaults to 1.
        p (list of float, optional): probability of each time step to exclude
            the event. Defaults to None.
        plot_results (bool, optional): if True plots intermediate results.
```

```
            Defaults to True.

    Returns:
        (list of float, list of float, list of float, list of float, ): the
            standard deviation of Pnew for the given intervals, for: (1) all the
            duration, (2) pre-spiking, (3) excitatory and (4) inhibitory period
            periods.
    """

    t = np.arange(0, dur, dt)

    if pold is None:
        pold = np.zeros(len(t))

    if plot_results:
        plt.figure(figsize=(10, 5))
        plt.plot(t, pold, 'r-', alpha=0.3)

    if spike_index_per_input is None:
        spike_index_per_input = []
        for i in range(num_input):
            spike_index_per_input.append(
                poisson_process(rate*dur, rate, dur, dt, p=p))

    alpha_fun = get_alpha_fun(t, a, scale_fac)

    for spike_index_i in spike_index_per_input:
        for i in spike_index_i:
            if plot_results:
                plt.plot(t[:len(t)-i]+i*dt, alpha_fun[:len(alpha_fun)-i], 'r-',
                         lw=2, alpha=0.3, label='alpha␣functions')
            pold[i:] = pold[i:]+alpha_fun[:len(alpha_fun)-i]

    if plot_results:
        plt.plot(t, pold, 'k-', alpha=0.4)
        plt.plot(t, pold, 'ko', markersize=1, label='sum␣of␣alpha␣functions')

        plt.ylabel('Pold')
        plt.xlabel('Time␣(sec)␣(dt={}␣sec)'.format(dt))

        # to avoid duplicate legend entries
        handles, labels = plt.gca().get_legend_handles_labels()
        labels, ids = np.unique(labels, return_index=True)
        handles = [handles[i] for i in ids]
        plt.legend(handles, labels, loc='best')

    return get_std(pold, t, dt, intervals, plot_results=plot_results)

```

```

def get_std(pold_all, time, dt, intervals, prespiking_index=None,
            excitatory_index=None, inhibitory_index=None, plot_results=True):
    """Plots the standard deviation of Pnew  with different color each period
    i.e. excitatory(blue), inhibitory(orange) and pre-spiking(green).

    Args:
        pold_all (list of float): membrane potential (Pold).
        time (list of float): time.
        dt (float): time step.
        intervals (list of floats): duration of intervals used for Pnew
            calculation.
        prespiking_index (list of int, optional): indices of pre-spiking period.
            Defaults to None.
        excitatory_index (list of int, optional): indices of excitatory period.
            Defaults to None.
        inhibitory_index (list of int, optional): indices of inhibitory period.
            Defaults to None.
        plot_results (bool, optional): if True plots intermediate results.
            Defaults to True.

    Returns:
        (list of float, list of float, list of float, list of float, ): the
            standard deviation of Pnew for the given intervals, for: (1) all the
            duration, (2) pre-spiking, (3) excitatory and (4) inhibitory period
            periods.
    """

    if plot_results:
        # Plot pold values
        plt.figure(figsize=(10, 5))
        plt.plot(time, pold_all, 'k-', alpha=0.4, label='all')
        plt.plot(time, pold_all, 'k-', linewidth=1, ms=4)

    std_all = np.full(len(intervals), np.nan)

    std_prespiking = None
    std_excitatory = None
    std_inhibitory = None

    if not excitatory_index is None:
        std_excitatory = np.full(len(intervals), np.nan)
        pold_excitatory = np.full(len(pold_all), np.nan)
        pold_excitatory[excitatory_index] = pold_all[excitatory_index]
        if plot_results:
            plt.plot(time, pold_excitatory, 'C0-', label='excitatory')

    if not prespiking_index is None:
        std_prespiking = np.full(len(intervals), np.nan)
        pold_prespiking = np.full(len(pold_all), np.nan)
        pold_prespiking[prespiking_index] = pold_all[prespiking_index]
        if plot_results:
            plt.plot(time, pold_prespiking, 'C2-', label='prespiking')

    if not inhibitory_index is None:
        std_inhibitory = np.full(len(intervals), np.nan)
        pold_inhibitory = np.full(len(pold_all), np.nan)
        pold_inhibitory[inhibitory_index] = pold_all[inhibitory_index]
        if plot_results:
            plt.plot(time, pold_inhibitory, 'C1-', label='inhibitory')

    if plot_results:
        plt.xlabel('Time␣(sec)␣(dt={}␣sec)'.format(dt))
        plt.ylabel('Pold')
        plt.legend()

        # Pnew subplots
        fig, _ = plt.subplots(figsize=(16, 8))
        step = 15
        axes_y = 3
        axes_x = math.ceil(
            len([i for i in intervals if (i-1) % step == 0])/axes_y)
        subplot_index = 1

    for j in range(len(intervals)):
        pnew_all = np.full(len(pold_all), np.nan)

        for i in range(intervals[j], len(pold_all)-intervals[j]):
            pnew_all[i] = pold_all[i] - 0.5 * \
                (pold_all[i-intervals[j]]+pold_all[i+intervals[j]])

        if len(pnew_all[~np.isnan(pnew_all)]) > 100:
            # exclude nan values
            std_all[j] = np.std(pnew_all[~np.isnan(pnew_all)])

        if not prespiking_index is None:
            pnew_prespiking = np.full(len(pold_all), np.nan)
            pnew_prespiking_index = local_utl.get_interval_indexes(
                intervals[j], prespiking_index)
            pnew_prespiking[pnew_prespiking_index] = pnew_all[pnew_prespiking_index]

            if len(pnew_prespiking[~np.isnan(pnew_prespiking)]) > 100:
                std_prespiking[j] = np.std(
                    pnew_prespiking[pnew_prespiking_index])

        if not excitatory_index is None:
            pnew_excitatory = np.full(len(pold_all), np.nan)
            pnew_excitatory_index = local_utl.get_interval_indexes(
                intervals[j], excitatory_index)
            pnew_excitatory[pnew_excitatory_index] = pnew_all[pnew_excitatory_index]

            if len(pnew_ex[~np.isnan(pnew_ex)]) > 100:
                std_excitatory[j] = np.std(
                    pnew_excitatory[pnew_excitatory_index])

        if not inhibitory_index is None:
            pnew_inhibitory = np.full(len(pold_all), np.nan)
            pnew_inhibitory_index = local_utl.get_interval_indexes(
                intervals[j], inhibitory_index)
            pnew_inhibitory[pnew_inhibitory_index] = pnew_all[pnew_inhibitory_index]

            if len(pnew_inhibitory[~np.isnan(pnew_inhibitory)]) > 100:
                std_inhibitory[j] = np.std(
                    pnew_inhibitory[pnew_inhibitory_index])

        if plot_results and (intervals[j]-1) % step == 0:
            # Plot pnew values for current interval
            ax = plt.subplot(axes_y, axes_x, subplot_index)
            subplot_index += 1
            ax.plot(time, pold_all, 'k-', alpha=0.2, ms=4, label='pold')
            ax.plot(time, pnew_all, 'k-', label='all')

            if not excitatory_index is None:
                ax.plot(time, pnew_excitatory, 'C0-', label='excitatory')

            if not prespiking_index is None:
                ax.plot(time, pnew_prespiking, 'C2-', label='prespiking')

            if not inhibitory_index is None:
                ax.plot(time, pnew_inhibitory, 'C1-', label='inhibitory')

            ax.set_title('interval {:.4}sec'.format(intervals[j]*dt))
            ax.set_xlim(0, 1)

    if plot_results:
        handles, labels = ax.get_legend_handles_labels()
        plt.legend(handles, labels, bbox_to_anchor=(2, 0), loc='lower right')
        plt.subplots_adjust(top=0.97, bottom=0.065, left=0.05,
                            right=0.975, hspace=0.5, wspace=0.4)

        # add a big axis, hide frame
        fig.add_subplot(111, frameon=False)
        # hide tick and tick label of the big axis
        plt.tick_params(labelcolor='none', top=False,
```

```python
                        bottom=False, left=False, right=False)
        plt.xlabel("Time")
        plt.ylabel("Pnew")
        plt.grid(b=None)

    return std_all, std_prespiking, std_excitatory, std_inhibitory


def plot_std(std_all, intervals, std_prespiking=None, std_excitatory=None,
             std_inhibitory=None, std_exp_df=None, ax=None, kwargs={}):
    """Plots the standard deviation of Pnew  with different color each period
    i.e. excitatory(blue), inhibitory(orange) and pre-spiking(green).

    Args:
        std_all (list of float): standard deviation of Pnew, during the
            total duration for all the given intervals.
        intervals (list of floats): duration of intervals used for Pnew
            calculation.
        std_prespiking (list of float): standard deviation of Pnew, during the
            pre-spiking period for all the given intervals. Defaults to None.
        std_excitatory (list of float): standard deviation of Pnew, during the
            excitatory period for all the given intervals. Defaults to None.
        std_inhibitory (list of float): standard deviation of Pnew, during the
            inhibitory period for all the given intervals. Defaults to None.
        std_exp_df (pandas dataframe, optional): The corresponding results of
            the experimental data. Defaults to None.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots(figsize=(5, 5))

    if not (std_exp_df is None):
        ax.plot(intervals, std_exp_df['std_ex'],
                color='C0', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_in'],
                color='C1', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_prespiking'],
                color='C2', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_all'],
                color='k', linestyle='--', alpha=0.4)

    ax.plot(intervals, std_all, color='k', label='all')

    if not std_excitatory is None:
        ax.plot(intervals, std_excitatory, color='C0', label='excitatory')

    if not std_prespiking is None:
        ax.plot(intervals, std_prespiking, color='C2', label='prespiking')

    if not std_inhibitory is None:
        ax.plot(intervals, std_inhibitory, color='C1', label='inhibitory')

    ax.set_xlabel('Time␣interval␣(sec)')
    ax.set_ylabel('Standard␣deviation')

    ax.legend()
    ax.set(**kwargs)


def plot_std_compare(std_arr, label_arr, intervals, std_exp_df=None,
                     ax=None, kwargs={}):
    """Plots given data on a single figure.

    Args:
        std_arr (list of list of float): multiple lists of the standard
            deviation.
        label_arr (list of str): the label for each corresponding list in
            std_arr.
        intervals (list of floats): duration of intervals used for Pnew
            calculation.
        std_exp_df (pandas dataframe, optional): The corresponding results of
            the experimental data. Defaults to None.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots(figsize=(5, 5))

    if not (std_exp_df is None):
        ax.plot(intervals, std_exp_df['std_all'],
                color='k', linestyle='--', alpha=0.4, label='all')
        ax.plot(intervals, std_exp_df['std_ex'],
                color='C0', linestyle='--', alpha=0.4, label='excitatory')
        ax.plot(intervals, std_exp_df['std_in'],
                color='C1', linestyle='--', alpha=0.4, label='inhibitory')
        ax.plot(intervals, std_exp_df['std_prespiking'],
                color='C2', linestyle='--', alpha=0.4, label='pre-spiking')

    for i, std in enumerate(std_arr):
        ax.plot(intervals, std,  label=label_arr[i])

    ax.set_xlabel('Time␣interval␣(sec)')
    ax.set_ylabel('Standard␣deviation')
```

```python

    if len(std_arr) < 7:
        ax.legend(loc='best')
    else:
        plt.legend(loc='center␣left', bbox_to_anchor=(1, 0.5))
    ax.set(**kwargs)


if __name__ == "__main__":

    # read membrane potential of experimental data
    mem_filename = '/home/george/Undergraduate-Thesis[Git]/Data/Carandini/61.csv'
    pold_exp_df = pd.read_csv(mem_filename, header=0, squeeze=True)

    # read std of experimental data
    stds_filename = '~/Undergraduate-Thesis[Git]/Data/Simulation/std.csv'
    std_exp_df = pd.read_csv(stds_filename, header=0, squeeze=True)

    intervals = np.array(range(1, 200, 5))  # in time steps
    dt = pms.dt  # time-step ( seconds)
    dur = 1.0

# 1. Duration (width/alpha parameter)
# 1.1 Alpha functions (equal distance between spikes)
# 1.1.1 Low rate
    std_cmp = []
    r = 25.0
    h = 4.
    a_list = np.arange(0.001, 0.01, 0.002)

    for a in a_list:
        std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
            a, h, intervals, dt,
            spike_index_per_input=[(np.arange(0.0, dur, dur/r)/dt).astype(int)],
            plot_results=False)
        std_cmp.append(std_all)

    fig, (a0, a1) = plt.subplots(2, 1, gridspec_kw={
        'height_ratios': [3, 1]}, figsize=(5, 8))
    fig.suptitle(
        "Constant␣height(h={})␣and␣rate(r={})\n␣Variable␣shape␣parameter(a)␣-␣width".
            format(h, r))
    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["a={:.3f}".format(a)
                                        for a in a_list], intervals*dt, None, a0)
    a0.set_ylim([0, 3])
    a0.set_xlim([0, 0.05])

    for a in a_list:
        alpha_fun = get_alpha_fun(np.arange(0, 1.0, dt), a, h)
        a1.plot(np.arange(0, len(alpha_fun), 1)*dt, alpha_fun)
    a1.set_xlim([0, 0.05])

    a1.set_title("Alpha␣function")
    a1.set_ylabel('Value␣(mV)')
    a1.set_xlabel('Time␣(sec)')

    plt.tight_layout()
    fig.subplots_adjust(top=0.88)

    plt.show()

# 1.1.2 High rate
    std_cmp = []
    r = 100.0
    h = 4.
    a_list = np.arange(0.001, 0.01, 0.002)

    for a in a_list:
        std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
            a, h, intervals, dt,
            spike_index_per_input=[(np.arange(0.0, dur, dur/r)/dt).astype(int)],
            plot_results=False)
        std_cmp.append(std_all)

    fig, (a0, a1) = plt.subplots(2, 1, gridspec_kw={
        'height_ratios': [3, 1]}, figsize=(5, 8))
    fig.suptitle(
        "Constant␣height(h={})␣and␣rate(r={})\n␣Variable␣shape␣parameter(a)␣-␣width".
            format(h, r))
    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["a={:.3f}".format(a)
                               for a in a_list], intervals*dt, None, a0)
    a0.set_ylim([0, 3])
    a0.set_xlim([0, 0.05])

    for a in a_list:
        alpha_fun = get_alpha_fun(np.arange(0, 1.0, dt), a, h)
        a1.plot(np.arange(0, len(alpha_fun), 1)*dt, alpha_fun)
    a1.set_xlim([0, 0.05])

    a1.set_title("Alpha␣function")
    a1.set_ylabel('Value␣(mV)')
    a1.set_xlabel('Time␣(sec)')

    plt.tight_layout()
    fig.subplots_adjust(top=0.88)

    plt.show()

# 1.2 Poisson process
    std_cmp = []
    h = 4.0
    r = 100
    a_list = np.arange(0.001, 0.01, 0.002)

    for a in a_list:
        random.seed(0)
        std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
            a, h, intervals, dt, rate=r, plot_results=False)
        std_cmp.append(std_all)

    fig, (a0, a1) = plt.subplots(2, 1, gridspec_kw={
        'height_ratios': [3, 1]}, figsize=(5, 8))
    fig.suptitle(
        "Constant␣height(h={})␣and␣rate(r={})\n␣Variable␣shape␣parameter(a)␣-␣width".
            format(h, r))
    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["a={:.3f}".format(a)
                                  for a in a_list], intervals*dt, None, a0)
    a0.set_xlim([0, 0.05])

    for a in a_list:
        alpha_fun = get_alpha_fun(np.arange(0, 1.0, dt), a, h)
        a1.plot(np.arange(0, len(alpha_fun), 1)*dt, alpha_fun)
    a1.set_xlim([0, 0.05])

    a1.set_title("Alpha␣function")
    a1.set_ylabel('Value␣(mV)')
    a1.set_xlabel('Time␣(sec)')

    plt.tight_layout()
    fig.subplots_adjust(top=0.88)

    plt.show()

# 2.1 Amplitude (single Poisson process)
    std_cmp = []
    a = 0.003
    r = 100
    h_list = np.arange(2, 5, 0.5)

    for h in h_list:
        random.seed(0)
```

```python
        std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
            a, h, intervals, dt, rate=r, plot_results=False)
        std_cmp.append(std_all)

    fig, (a0, a1) = plt.subplots(2, 1, gridspec_kw={
        'height_ratios': [3, 1]}, figsize=(5, 8))
    fig.suptitle(
        "Constant␣width(a={})␣and␣rate(r={})\n␣Variable␣height(h)".format(a, r))

    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["h="+str(h)
                               for h in h_list], intervals*dt, None, a0)

    for h in h_list:
        alpha_fun = get_alpha_fun(np.arange(0, 1.0, dt), a, h)
        a1.plot(np.arange(0, len(alpha_fun), 1)*dt, alpha_fun)
    a1.set_xlim([0, 0.018])

    a1.set_title("Alpha␣function")
    a1.set_ylabel('Value␣(mV)')
    a1.set_xlabel('Time␣(sec)')

    plt.tight_layout()
    fig.subplots_adjust(top=0.88)

    plt.show()

# 2.2 see Matlab

# 3. Rate
# 3.1 Compare rate - Alpha functions equal distance between spikes
# intervals resolution affects the accuracy of the results for high rates and
# appear wrong oscillations in variability
    intervals = np.array(range(1, 200, 1))
    std_cmp = []
    a = 0.004
    h = 4.0
    r_list = np.arange(30, 100, 20)
    r_list = np.append(r_list, 300)
    r_list = np.append(r_list, 500)

    for r in r_list:
        std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
            a, h, intervals, dt,
            spike_index_per_input=[(np.arange(0.0, dur, dur/r)/dt).astype(int)],
            plot_results=False)
        std_cmp.append(std_all)

    fig, a0 = plt.subplots(1, 1, figsize=(5, 5))
    fig.suptitle(
        "Constant␣width(a={})␣and␣height(h={})\n␣Variable␣rate(r)".format(a, h))

    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["r="+str(r)
                               for r in r_list], intervals*dt, std_exp_df=None, ax=a0)

    plt.tight_layout()
    fig.subplots_adjust(top=0.83)

    # reset values for the rest experiments
    intervals = np.array(range(1, 200, 5))
    dt = pms.dt

    plt.show()

# 3.2 Compare rate - single Poisson process
    random.seed(0)
    std_cmp = []
    a = 0.003
    h = 4.0
    r_list = np.arange(30, 100, 20)
    r_list = np.append(r_list, 300)
    r_list = np.append(r_list, 500)
    n_list = [1]

    for r in r_list:
        for n in n_list:
            std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
                a, h, intervals, dt, rate=r, num_input=n, plot_results=False)
            std_cmp.append(std_all)

    fig, a0 = plt.subplots(1, 1, figsize=(6, 5))
    fig.suptitle(
        "Constant␣width(a={}),␣height(h={})\n␣Variable␣rate␣(r)".format(a, h))

    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["r="+str(r)
                               for r in r_list], intervals*dt, None, a0)

    plt.tight_layout()
    fig.subplots_adjust(top=0.83)

    plt.show()

# 3.3 Compare Multiple Poisson processes - Equal accumulated rate
    random.seed(800)
    std_cmp = []
    a = 0.003
    h = 4.0
    r_list = [100]  # range(50,500,50)
    n_list = range(1, 10, 2)

    for r in r_list:
        for n in n_list:
            std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
                a, h, intervals, dt, rate=r/n, num_input=n, plot_results=False)
            std_cmp.append(std_all)

    fig, a0 = plt.subplots(1, 1, figsize=(6, 5))
    fig.suptitle(
        "Constant␣width(a={}),␣height(h={}),␣rate(r={})\n␣Variable␣number␣of␣inputs␣(n)".format(a, h, r))

    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["n="+str(n)
                               for n in n_list], intervals*dt, None, a0)

    plt.tight_layout()
    fig.subplots_adjust(top=0.83)

    plt.show()

# Same experiment different random seed
    random.seed(0)
    std_cmp = []
    a = 0.003
    h = 4.0
    r_list = [100]  # range(50,500,50)
    n_list = range(1, 10, 2)

    for r in r_list:
        for n in n_list:
            std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
                a, h, intervals, dt, rate=r/n, num_input=n, plot_results=False)
            std_cmp.append(std_all)

    fig, a0 = plt.subplots(1, 1, figsize=(6, 5))
    fig.suptitle(
        "Constant␣width(a={}),␣height(h={}),␣rate(r={})\n␣Variable␣number␣of␣inputs␣(n)".format(a, h, r))

    a0.set_title("Variability")
    plot_std_compare(std_cmp, ["n="+str(n)
                               for n in n_list], intervals*dt, None, a0)

    plt.tight_layout()
    fig.subplots_adjust(top=0.83)

    plt.show()

# Final estimation result
    random.seed(0)
    std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
                0.0028, 3.8, intervals, dt, rate=90, num_input=1,
                plot_results=False)
    std_cmp = [std_all]
    std_all, std_prespiking, std_excitatory, std_inhibitory = alpha_wave(
                0.0045, -1.3, intervals, dt, rate=90, num_input=1,
                plot_results=False)
    std_cmp.append(std_all)

    fig, a0 = plt.subplots(1, 1, figsize=(6, 5))
    plot_std_compare(std_cmp, ['estimated␣pre-spiking', 'estimated␣inhibitory'],
                     intervals*dt, std_exp_df, a0)

    plt.show()
```

## Application to intracellula recordings

Listing A.6: variability.m - Application of the measure to real data.

```
GLOBALS;
close all; clear all;

%% Get cell's data
[time ,potential, Vth, Vrest, samplerate, nsamples, spikes, spikeheight, spikeduration,
     istim, iexp] = loadcellinfo(61,270);
timestep = 1/samplerate;

%% Remove spikes from data
dt0 = -1;
dt1 = spikeduration*samplerate-dt0;
pold_all = remove_spikes( potential, spikes, nsamples, samplerate, dt0, dt1);

%%  Separate presynaptic excitatory and inhibitory periods
thresh = Vrest;
pold_ex = NaN*ones(length(pold_all),1);
pold_ex(pold_all>thresh) = pold_all(pold_all>thresh);

pold_in = NaN*ones(length(pold_all),1);
pold_in(pold_all<=thresh) = pold_all(pold_all<=thresh);
```

```

%% Pre-spiking period
prespiking_index = [];

if spikes
    [t_bursts, t_burst_groups, index_pre_burst_spikes, index_burst_groups, in_bursts] =
         detect_bursts(potential, spikes, timestep);
    sprintf("Number of bursts found: %d", size(t_bursts,2))

    % Start index of slope
    preburst_rest_index = []; % index before burst at resting potential
    for i = 1:numel(index_burst_groups)
        preburst_rest_index(end+1) = find(potential(1:index_burst_groups(i)) < Vrest,
            1, 'last') + 1;
    end

    % In case the potential is above thresh before the first burst
    % insert at the beginning of the list 1
    if preburst_rest_index(1) > index_burst_groups(1)
        preburst_rest_index = [1 preburst_rest_index];
    end

    for i = 1:min(length(preburst_rest_index),length(index_burst_groups))
        prespiking_index = [prespiking_index, preburst_rest_index(i):index_burst_groups
            (i)-1];
    end
end
prespiking_potential = NaN*ones(length(pold_all),1);
prespiking_potential(prespiking_index) = pold_all(prespiking_index);

%% Plot potential and different
figure('Position', [10 10 21 7]); hold on;
p1 = plot(time, potential, 'color', 0.8*[1 1 1], 'LineWidth',1);
p2 = plot(time, pold_ex,'color',[0, 0.4470, 0.7410], 'LineWidth',1);
p3 = plot(time, pold_in,'color',[0.8500, 0.3250, 0.0980], 'LineWidth',1);
p4 = plot(time, prespiking_potential, 'color', [0 0.7 0], 'LineWidth',1);
yline(thresh,'--','Resting␣potential');
legend([p1(1), p2(1), p3(1), p4(1)], 'spikes', 'excitatory', 'inhibitory', 'pre-spiking
    ');
xlabel('Time␣(sec)'); ylabel('Pold␣(mV)');
xlim([0 4]); ylim([-90 0]);
box on;

%% Calculate pnew and standard deviation
intervals = 1:5:200;
std_all=NaN*ones(length(intervals),1);
std_ex=NaN*ones(length(intervals),1);
std_in=NaN*ones(length(intervals),1);
```

```
std_prespiking=NaN*ones(length(intervals),1);

figure();
ax=NaN*ones(length(intervals),1);
rows = 5;

for j = 1:numel(intervals)
    pnew_all = NaN*ones(length(pold_all),1);
    pnew_ex = NaN*ones(length(pold_all),1);
    pnew_in = NaN*ones(length(pold_all),1);
    pnew_prespiking = NaN*ones(length(pold_all),1);

    for i = intervals(j)+1:numel(pold_all)-intervals(j)
        pnew_all(i) = pold_all(i) - 0.5*(pold_all(i-intervals(j))+pold_all(i+intervals(
            j)));
    end

    % pnew excitatory indexes
    pnew_ex_index = pnewIntervalIndexes(intervals(j), find(pold_all>thresh));
    pnew_ex(pnew_ex_index) = pnew_all(pnew_ex_index);

    % pnew inhibitory indexes
    pnew_in_index = pnewIntervalIndexes(intervals(j), find(pold_all<=thresh));
    pnew_in(pnew_in_index) = pnew_all(pnew_in_index);

    % pnew pre-spiking indexes
    pnew_prespiking_index = pnewIntervalIndexes(intervals(j), prespiking_index);
    pnew_prespiking(pnew_prespiking_index) = pnew_all(pnew_prespiking_index);

    std_all(j) = std(pnew_all(~isnan(pnew_all))); % exclude nan values

    if numel(pnew_ex(~isnan(pnew_ex))) > 100
        std_ex(j) = std(pnew_ex(~isnan(pnew_ex)));
    end

    if numel(pnew_in(~isnan(pnew_in))) > 100
        std_in(j) = std(pnew_in(~isnan(pnew_in)));
    end

    if numel(pnew_prespiking(~isnan(pnew_prespiking))) > 100
        std_prespiking(j) = std(pnew_prespiking(~isnan(pnew_prespiking)));
    end

    ax(j) = subplot(rows,ceil(length(intervals)/rows),j); hold on;
    plot(time,pold_all - Vrest, 'color', 0.7*[1 1 1], 'LineWidth',1);
    yline(0, 'color', 0.8*[1 1 1]);
    plot(time,pnew_all, 'k', 'LineWidth',2);
    plot(time, pnew_ex,'color',[0, 0.4470, 0.7410], 'LineWidth',2);
    plot(time, pnew_in,'color',[0.8500, 0.3250, 0.0980], 'LineWidth',2);
    plot(time, pnew_prespiking,'color',[0 0.7 0], 'LineWidth',2);
    xlabel('Time␣(sec)'); ylabel('Pnew␣(mV)', 'LineWidth',2);
    box on;
    set(gca,'fontsize',13);
    title(strcat({'interval␣'}, num2str(intervals(j)*(1/samplerate)*1000), {'␣msec'}));
end

set(ax, 'xlim',[0 4.0],'ylim',[-10 20]);

%% Plot standard deviation
figure();
hold on;
p1 = plot(intervals*(1/samplerate), std_ex);
p2 = plot(intervals*(1/samplerate), std_in);
p3 = plot(intervals*(1/samplerate), std_prespiking,'color',[0 0.7 0]);
p4 = plot(intervals*(1/samplerate), std_all, 'k-');
legend([p4(1), p1(1), p2(1), p3(1)], 'all', 'excited', 'inhibited', 'pre-spiking');
xlabel('Time␣interval␣(sec)'); ylabel('Standard␣deviation');
set(gca,'fontsize',13);
box on;

%% Write std to csv file
T = array2table([std_all std_ex std_in std_prespiking]);
T.Properties.VariableNames(1:4) = {'std_all', 'std_ex', 'std_in', 'std_prespiking'}
writetable(T,'~/Undergraduate-Thesis[Git]/Data/Simulation/std.csv');
```

Listing A.7: pnewIntervalIndexes.m - Utility function that returns the indices of Pnew to be used in the standard deviation.

```
function [interval_indexes] = pnewIntervalIndexes(interval, indexes)
% Returns the given indexes, after removing <interval> elements from the
% beginning and ending of each sequence of consecutive numbers.

index_bounds = indexBounds(indexes);

interval_indexes = [];

for i = 1:2:numel(index_bounds)

    if index_bounds(i)+interval >= index_bounds(i+1)
        continue
    end
    interval_indexes = [interval_indexes, index_bounds(i)+interval:index_bounds(i+1)-
        interval];
end

end
```

```

function index_bounds = indexBounds(indexes)
% Returns the bounds (first and last element) of consecutive numbers.
%
% Example:
% [1 , 3, 4, 5, 7, 8, 9, 10] -> [1, 3, 5, 7, 10]

index_bounds = (indexes(1));

for i = 1:numel(indexes)
    if i==numel(indexes)
        index_bounds = [index_bounds, indexes(i)];
        break
    end

    if indexes(i)+1==indexes(i+1)
        continue
    else
        index_bounds = [index_bounds, indexes(i)];
        index_bounds = [index_bounds, indexes(i+1)];
    end

end

end
```

Listing A.8: remove_spikes.m - Utility function that removes the spikes from the membrane potential.

```
function [potential_spikes_rem] = remove_spikes(potential, spikes, nsamples, samplerate
    , dt0, dt1)
% Remove spikes from membrane potential.
% dt0: relative time before/beginning of spike
% dt1: relative time after/end of spike
    potential_spikes_rem = potential;

    if ~isempty(spikes)
        for spike = spikes(:)'
            t0 = max(spike+dt0, 1);
            t1 = min(spike+dt1, nsamples);

            p0 = potential(t0);
            p1 = potential(t1);
            potential_spikes_rem(t0:t1) = p0+(p1-p0)/(t1-t0)*([t0:t1]'-t0);
        end

    end
end
```

## A.3 Simulation

### A.3.1 Neuron model

Listing A.9: simulation.py - Simulation of the V1 neuron.

```
import utils as local_utils
import firing_rate
import kernel
import plots

import numpy as np
from tqdm import tqdm  # progress bar
from brian2 import *
# Suppress resolution conflict warnings
BrianLogger.suppress_name('resolution_conflict')


def execute(Vth, Vreset, Vrest, theta,
            refract, dur, tau, tau_syn_ex, tau_syn_in,
            xs_on_ex, ys_on_ex, xs_off_ex, ys_off_ex,
            xs_on_inh, ys_on_inh, xs_off_inh, ys_off_inh,
            X, Y, lx, dx, ly, dy, sigma_center, sigma_surround,
            stimulus, t, we, wi, num_rfc, r0=0.0, L0=0.0, G_ex=100.0, G_inh=None,
            plot_results=True, ax_pot=None, ax_rf=None, ax_firing_rate=None):
    """Execute the simulation based on the given parameters.

    Args:
        Vth (float): firing threshold.
        Vreset (float): reset potential.
        Vrest (float): resting potential.
        theta (float): stimulus orientation.
        refract (float): duration of refractory period.
        dur (float): duration of the simulation.
        tau (float): membrane potential's time constant.
        tau_syn_ex (float): presynaptic excitatory time constant.
        tau_syn_in (float): presynaptic inhibitory time constant.
        xs_on_ex (list of float): x coordinates of excitatory ON receptive
            fields.
        ys_on_ex (list of float): x coordinates of excitatory ON receptive
            fields.
        xs_off_ex (list of float): x coordinates of excitatory OFF receptive
            fields.
        ys_off_ex (list of float): x coordinates of excitatory OFF receptive
```

```
            fields.
        xs_on_inh (list of float): x coordinates of inhibitory ON receptive
            fields.
        ys_on_inh (list of float): x coordinates of inhibitory ON receptive
            fields.
        xs_off_inh (list of float): x coordinates of inhibitory OFF receptive
            fields.
        ys_off_inh (list of float): x coordinates of inhibitory OFF receptive
            fields.
        X (ndarray of floats): x coordinates of the mesh grid.
        Y (ndarray of floats): y coordinates of the mesh grid.
        lx (float): size of the grid (x axis)
        dx (float): resolution of the grid (x axis).
        ly (float): size of the grid (y axis)
        dy (float): resolution of the grid (y axis).
        sigma_center (float): size of the central region (should be less than
            sigma_surround).
        sigma_surround (float): size of the surround region (should be greater
            than sigma_center).
        stimulus (list of float ndarray): the stimulus at each time step.
        t (list of float): time.
        we (float): weight of excitatory presynaptic input.
        wi (float): weight of inhibitory presynaptic input.
        num_rfc (int): number of receptive fields.
        r0 (float, optional): any background firing that may occur when stimulus
            is zero. Defaults to 0.0.
        L0 (float, optional): the threshold value that must be attained before
            firing begins. Defaults to 0.0.
        G_ex (float, optional): constant of proportionality for the excitation.
            Defaults to 100.
        G_inh (float, optional): constant of proportionality for the inhibition.
            Defaults to None.
        plot_results (bool, optional): if True makes the related plots. Defaults
             to True.
        ax_pot (axes object, optional): specified target axes for the membrane
            potential. Defaults to None.
        ax_rf (axes object, optional): specified target axes for the receptive
            fields. Defaults to None.
        ax_firing_rate (axes object, optional): specified target axes for the
            firing rate. Defaults to None.

    Returns:
        (brian object, brian object, brian object, brian object): state monitors
            for: (1) LIF neuron, (2) LIF spike monitor, (3) excitatory and (4)
            inhibitory presynaptic spikes monitors.
    """

    # default value for G_inh is the same as G_ex
    if G_inh is None:
        G_inh = G_ex

    # Definition of neuron model
    eqs = '''
          dv/dt = (Vrest-v+x+w)/tau : volt

          dx/dt = (y-x)/tau_syn_ex : volt
          dy/dt = -y/tau_syn_ex : volt

          dw/dt = (z-w)/tau_syn_in : volt
          dz/dt = -z/tau_syn_in : volt
          '''

    lif = NeuronGroup(1, eqs, threshold='v>Vth', reset='v=Vreset',
                      method='exact', refractory=refract, dt=defaultclock.dt)
    lif.v = Vrest

    xs_on_ex_rot, ys_on_ex_rot = local_utils.rotate_point_arr(
        xs_on_ex, ys_on_ex, theta)
    xs_off_ex_rot, ys_off_ex_rot = local_utils.rotate_point_arr(
        xs_off_ex, ys_off_ex, theta)
    xs_on_inh_rot, ys_on_inh_rot = local_utils.rotate_point_arr(
        xs_on_inh, ys_on_inh, theta)
    xs_off_inh_rot, ys_off_inh_rot = local_utils.rotate_point_arr(
        xs_off_inh, ys_off_inh, theta)

    kernels_on_ex = []
    for i, (x, y) in enumerate(zip(xs_on_ex_rot, ys_on_ex_rot)):
        kernels_on_ex.append(kernel.spatial_kernel(
            X, Y, x, y, sigma_center, sigma_surround, inverse=1))

    kernels_on_inh = []
    for i, (x, y) in enumerate(zip(xs_on_inh_rot, ys_on_inh_rot)):
        kernels_on_inh.append(kernel.spatial_kernel(
            X, Y, x, y, sigma_center, sigma_surround, inverse=1))

    kernels_off_ex = []
    for i, (x, y) in enumerate(zip(xs_off_ex_rot, ys_off_ex_rot)):
        kernels_off_ex.append(kernel.spatial_kernel(
            X, Y, x, y, sigma_center, sigma_surround, inverse=-1))

    kernels_off_inh = []
    for i, (x, y) in enumerate(zip(xs_off_inh_rot, ys_off_inh_rot)):
        kernels_off_inh.append(kernel.spatial_kernel(
            X, Y, x, y, sigma_center, sigma_surround, inverse=-1))

    kernels_ex = kernels_on_ex + kernels_off_ex
    kernels_inh = kernels_off_inh + kernels_on_inh

    firing_rate_ex = firing_rate.calculate_firing_rate(
        kernels_ex, stimulus, t, r0, L0, G_ex)
    firing_rate_inh = firing_rate.calculate_firing_rate(
        kernels_inh, stimulus, t, r0, L0, G_inh)

    firing_rate_ex_arr = TimedArray(firing_rate_ex*Hz, dt=defaultclock.dt)
    firing_rate_inh_arr = TimedArray(firing_rate_inh*Hz, dt=defaultclock.dt)

    poisson_gr_ex = PoissonGroup(
        2*num_rfc, rates='firing_rate_ex_arr(t,i)', dt=defaultclock.dt)
    poisson_gr_inh = PoissonGroup(
        2*num_rfc, rates='firing_rate_inh_arr(t,i)', dt=defaultclock.dt)

    # The on_pre keyword defines what happens when a presynaptic spike
    # arrives at a synapse.
    # excitatory synapses & inhibitory synapses
    synapses_ex = Synapses(poisson_gr_ex, lif, on_pre='y␣+=␣we*exp(1)')
    synapses_inh = Synapses(poisson_gr_inh, lif, on_pre='z␣+=␣wi*exp(1)')

    synapses_ex.connect()
    synapses_inh.connect()

    lif_state_monitor = StateMonitor(lif, ['v'], record=True)
    lif_spike_monitor = SpikeMonitor(lif)
    # excitatory presynaptic action potentials
    epsp_monitor = SpikeMonitor(poisson_gr_ex)
    # inhibitory presynaptic action potentials
    ipsp_monitor = SpikeMonitor(poisson_gr_inh)

    # Create a Network object in order to prevent "Magic Error" from Brian
    # Reference: Brian2 MagicNetwork documentation
    net = Network(lif)
    net.add(synapses_ex, synapses_inh,
            poisson_gr_inh, poisson_gr_ex,
            lif_state_monitor, lif_spike_monitor,
            epsp_monitor, ipsp_monitor)

    # Run simulation
    net.run(dur)

    if plot_results:
        plots.plot_mem(lif_state_monitor, epsp_monitor,
                       ipsp_monitor, show_pre_spikes=True,
                       ax=ax_pot,
                       kwargs={'title': (str(round(math.degrees(theta))) + "degrees")})

        plots.plot_receptive_field(kernels_on_ex, kernels_off_ex,
            kernels_on_inh, kernels_off_inh, stimulus, lx, ly, t=0, ax=ax_rf,
            kwargs={'title': (str(round(math.degrees(theta))) + "degrees")})

        plots.plot_firing_rate_per_rfc(firing_rate_ex,
            np.concatenate((xs_on_ex_rot, xs_off_ex_rot), axis=0),
            np.concatenate((ys_on_ex_rot, ys_off_ex_rot), axis=0), t,
            ax=ax_firing_rate)
        plots.plot_firing_rate_per_rfc(firing_rate_inh,
            np.concatenate((xs_on_inh_rot, xs_off_inh_rot), axis=0),
            np.concatenate((ys_on_inh_rot, ys_off_inh_rot), axis=0), t,
            ax=ax_firing_rate)

    return lif_state_monitor, lif_spike_monitor, epsp_monitor, ipsp_monitor


if __name__ == "__main__":
    import params as pms

    # Declare plots
    # Number of subplots per axis
    axes_x = 4
    axes_y = 2

    # Membrane potential plots
    fig_pot, axs_pot = plt.subplots(axes_y, axes_x, sharex=True, sharey=True)
    fig_pot.suptitle('Membrane␣Potential')
    axs_pot[axes_y-1, axes_x-1].set_visible(False)  # remove last subplot

    # Receptive field plots
    fig_rf, axs_rf = plt.subplots(axes_y, axes_x, sharex=True, sharey=True)
    fig_rf.suptitle('LGN␣Receptive␣Fields')
    axs_rf[axes_y-1, axes_x-1].set_visible(False)  # remove last subplot

    # Firing rate plots
    fig_fire_rate, axs_fire_rate = plt.subplots(
        axes_y, axes_x, sharex=True, sharey=True)
    fig_fire_rate.suptitle('Firing␣Rate')
    axs_fire_rate[axes_y-1, axes_x-1].set_visible(False)  # remove last subplot

    for index, theta_i in enumerate(tqdm(pms.Theta)):
        lif_state_monitor, lif_spike_monitor, epsp_monitor, ipsp_monitor = execute(
            pms.Vth, pms.Vreset, pms.Vrest, theta_i,
            pms.refract, pms.dur, pms.tau, pms.tau_syn_ex, pms.tau_syn_in,
            pms.xs_on_ex, pms.ys_on_ex, pms.xs_off_ex, pms.ys_off_ex,
            pms.xs_on_inh, pms.ys_on_inh, pms.xs_off_inh, pms.ys_off_inh,
            pms.X, pms.Y, pms.lx, pms.dx, pms.ly, pms.dy,
            pms.sigma_center, pms.sigma_surround,
            pms.stimulus, pms.t, pms.we, pms.wi, pms.num_rfc,
            pms.r0, pms.L0, pms.G_ex, pms.G_inh, plot_results=True,
            ax_pot=axs_pot[index//axes_x, index % axes_x],
            ax_rf=axs_rf[index//axes_x, index % axes_x],
            ax_firing_rate=axs_fire_rate[index//axes_x, index % axes_x])

    plt.show()
```

Listing A.10: params.py - All the parameters used in the simulation.

```
import numpy as np
from brian2 import *

import stimuli

# Grid parameters
dx = dy = 0.05              # resolution - step size (degrees)
lx = ly = 8.0               # size (degrees)
x = np.arange(-lx/2, lx/2, dx)
y = np.arange(-ly/2, ly/2, dy)
X, Y = np.meshgrid(x, y)    # grid

# Center-Surround parameters (values from Archie and Mel, 2000)
# Convert from minutes(') to degrees x 1/60
sigma_center = 10.6 * (1/60)    # size of the central region (degrees)
sigma_surround = 31.8 * (1/60)  # size of the surround region (degrees)
# number of RF-centers per type(ON/OFF and ex/inh)
num_rfc = 10

# Time parameters
# set dt for all objects (check Brian2 doc)
defaultclock.dt = 0.000244 * second
dt = defaultclock.dt/second          # time-step
# default value for defaultclock.dt = 0.0001 sec
# in experimental data time-step was 0.000244sec
dur = 4*second                       # duration of the simulation
t = np.arange(0., dur/second, dt)    # array of time instances

# Sine grating - spatial parameters
# so that an ON-center part is on the light and OFF-surround part on dark spot
K = 0.25/(0.39) * (2 * np.pi)   # spatial frequency (cycle per degree)
Phi = 0 * np.pi                 # spatial phase (radians)
# maximum degree of difference between light and dark areas
A = 1         # amplitude of sine grating
omega = 4     # temporal frequency (Hz)
Theta = [0, pi/6, pi/3, pi/2, 2*pi/3, 5*pi/6, pi]  # orientation (radians)
stimulus = stimuli.sine_grating(A, K, Phi, omega, X, Y, t)

```

```
# Simple cell parameters
# In experimental data the first spike of each burst has lower threshold (-55*mV)
# than the rest. (firing threshold for all the spikes -49.2197*mV).
# Lower the threshold for all spikes to that of the first spike
Vth = -55*mV  # threshold potential equal to the Vth for first spike in experimental
Vreset = -55*mV
Vrest = -76.3369*mV    # resting potential
tau = 7.5*ms           # time constant for LIF
tau_syn_ex = 1.0*ms    # time constant for ex. synapses
tau_syn_in = 1.0*ms    # time contant for inh. synapses

# Remember to set refractory period shorter than the time for burst detection
refract = 6*ms         # refractory period
we = 4.6*mV            # weight of excitatory synapses
wi = -1.4*mV           # weight of inhibitory synapses

# Positions of RF centers
# ratio of a subfield of the RF (see Jones and Palmer 1987)
ratio = 1.7
rf_w = (2 * np.pi)/K   # width of RF
rf_l = ratio * rf_w  # length of RF

# At each point there is an excitatory ON/OFF-center and an inhibitory OFF/ON-center
xs_on_ex = xs_off_inh = np.full(num_rfc, 0)
xs_off_ex = xs_on_inh = np.full(num_rfc, (0.5 * (2 * np.pi))/K)
ys_on_ex = ys_off_inh = np.arange(-rf_l/2, rf_l/2, rf_l/num_rfc)
ys_off_ex = ys_on_inh = np.arange(-rf_l/2, rf_l/2, rf_l/num_rfc)

# Firing rate params
r0 = 0       # any background firing that may occur when s = 0
L0 = 0       # the threshold value that L must attain before firing begins
G_ex = 135   # constant of proportionality (upper limit) - excitatory
G_inh = 100  # constant of proportionality (upper limit) - inhibitory
```

### A.3.2 Stimulus and Receptive fields

Listing A.11: kernel.py - Definition of the kernel of the receptive fields.

```
import numpy as np


def spatial_kernel(X, Y, x, y, sigma_center, sigma_surround, inverse=1):
    """Creates a spatial kernel for the LGN cells' receptive field.

    The kernel is defined as the difference of Gaussians.

    Reference: Eq 2.45, Theoretical Neuroscience Dayan & Abbot (2011).
```

```python

    Args:
        X (ndarray of floats): x coordinates of the mesh grid.
        Y (ndarray of floats): y coordinates of the mesh grid.
        x (float): x coordinate of the position of the kernel on the grid.
        y (float): y coordinate of the position of the kernel on the grid.
        sigma_center (float): size of the central region (should be less than
            sigma_surround).
        sigma_surround (float): size of the surround region (should be greater
            than sigma_center).
        inverse (int, optional): determines if the type of the kernel (ON/OFF).
            Takes values -1 or 1 only. If the given value is 1 the kernel is of
            type ON, otherwise OFF. Defaults to 1.

    Returns:
        ndarray of float: the spatial kernel, normalized so that for the perfect
            stimulus (for the case of type ON:1 in the center and -1 in the
            surround region) the firing rate is equal to 1.
    """

    Z = ((X - x)**2 + (Y - y)**2)  # move center

    center = (17.0 / (2*np.pi * (sigma_center**2))) * \
        np.exp(-Z / (2*(sigma_center**2)))

    surround = (16.0 / (2*np.pi * (sigma_surround**2))) * \
        np.exp(-Z / (2*(sigma_surround**2)))

    Z = center - surround

    # points outside circle/receptive field center are set to zero
    pts = (X-x)**2+(Y-y)**2 >= (2*sigma_surround)**2
    Z[pts] = 0

    # normalize by multiplying with scale factor(dividing number of points/area)
    return (Z * inverse) / np.sum(np.absolute(Z))
```

Listing A.12: stimuli.py - Definition of the stimulus.

```python
import numpy as np


def sine_grating(A, K, phi, omega, X, Y, t, theta=0.0, solid=True):
    """Generates a sinusoidal grating for every time step.

    Slightly modified Eq. 2.18, Theoretical Neuroscience Dayan & Abbot (2011).

    Note: The rotation of the stimulus can be set to 0 degrees, and instead
```

```
        rotate the position of the receptive field centers.

    Args:
        A (float): amplitude.
        K (float): spatial frequency.
        phi (float): phase.
        omega (float): temporal frequency.
        X (ndarray of float): x coordinates of the mesh grid.
        Y (ndarray of float): y coordinates of the mesh grid.
        t (list of float): time.
        theta (float, optional): rotation angle. Defaults to 0.0.
        solid (bool, optional): if true the stimulus will have only 1 or -1
            values, otherwise it will contain intermediate values. Defaults to
            True.

    Returns:
        ndarray of floats: stimulus; a sinusoidal grating for every time step.
    """

    stimulus = np.zeros((t.size, X[0].size, Y[0].size))

    for i, ti in enumerate(t):  # for each time-step
        stimulus[i, ...] = A * np.cos((K * X * np.cos(theta) -
                                       K * Y * np.sin(theta) - phi) +
                                      omega * 2 * np.pi * ti)

    if solid:
        stimulus[stimulus < 0] = -1
        stimulus[stimulus > 0] = 1

    return stimulus
```

Listing A.13: firing_rate.py - Definition of the presynaptic firing rate.

```
import numpy as np


def calculate_firing_rate(kernels, stimulus, t, r0=0.0, L0=0.0, G=1.0):
    """Calculation of the firing rate of LGN cells based on the given kernels
    and stimulus.

    Reference: Eq. 2.8 & 2.9, Theoretical Neuroscience Dayan and Abott (2011).

    Args:
        kernels (list of float ndarray): list of kernels.
        stimulus (list of float ndarray): the stimulus at each time step.
        t (list of float): list of time instaces.
        r0 (float, optional): any background firing that may occur when stimulus
```

```
            is zero. Defaults to 0.0.
        L0 (float, optional): the threshold value that must be attained before
            firing begins. Defaults to 0.0.
        G (float, optional): constant of proportionality. Defaults to 1.0.

    Returns:
        ndarray of float: the firing rate for each time step.
    """

    rate = np.zeros([t.size, len(kernels)])

    # Multiply spatial-kernel with sinusoidal stimulus at each time step
    for i, t in enumerate(t):
        rate[i, ...] = [np.sum(kernel * stimulus[i, ...])
                        for kernel in kernels]

    # Add background firing
    rate += r0
    # Rectify firing rates
    rate[rate < L0] = 0
    # Multiply with the constant of proportionality G
    return G*rate
```

### A.3.3 Utilities

Listing A.14: plots.py - Utility functions for plotting results.

```
import numpy as np
from brian2 import second, mV, plt
from matplotlib import colors
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm


def plot_kernels(kernels, lx, ly,  kwargs={}, ax=None):
    """Plots the given kernels.

    Args:
        kernels (list of ndarray of float): list of kernels.
        lx (float): dimensions (x axes).
        ly (float): dimensions (y axes).
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
        ax (axes object, optional): specified target axes. Defaults to None.
    """

    if not kernels:
        return

    if ax is None:
        fig, ax = plt.subplots()

    kernels_sum = np.sum(kernels, axis=0)
    masked_data = np.ma.masked_where(kernels_sum == 0.0, kernels_sum)
    ax.imshow(masked_data, extent=[-lx/2, lx/2, ly/2, -ly/2],
              interpolation='none', **kwargs)


def plot_kernel_3D(kernel, X, Y, kwargs={}, ax=None):
    """Plots single given kernel in 3D plot.

    Args:
        kernel (ndarray of float): spatial kernel of the receptive field.
        X (ndarray of floats): x coordinates of the mesh grid.
        Y (ndarray of floats): y coordinates of the mesh grid.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
        ax (axes object, optional): specified target axes. Defaults to None.
    """

    if ax is None:
        fig = plt.figure()
        ax = fig.gca(projection='3d')

    kernel[kernel == 0] = np.nan
    ax.plot_surface(X, Y, kernel, rstride=1, cstride=1, alpha=0, linewidth=0.5,
                    edgecolors='k')

    cmap = colors.ListedColormap(['blue', 'red'])
    bounds = [-1, 0, 1]
    norm = colors.BoundaryNorm(bounds, 2)
    ax.contourf(X, Y, kernel, 10, offset=-0.002, cmap=cmap,
                norm=norm, alpha=0.4)

    ax.set_xlim(-1.2, 1.2)
    ax.set_ylim(-1.2, 1.2)
    ax.set_zlim(-0.002, 0.01)
    ax.view_init(10, 45)
    ax.set_xlabel('x␣(degrees)')
    ax.set_ylabel('y␣(degrees)')
    ax.set_zlabel('Ds')
    # ax.set_zticks([0])


def plot_stimulus(stimulus, t, lx, ly, ax=None):
    """Plots the given stimulus at the specified index/time.

    Args:
        stimulus (ndarray of float): stimulus for every time step.
        t (int): time index.
        lx (float): dimensions (x axes).
        ly (float): dimensions (y axes).
        ax (axes object, optional): specified target axes. Defaults to None.
    """

    if ax is None:
        fig, ax = plt.subplots()

    ax.imshow(stimulus[t, :], extent=[-lx/2, lx/2, ly/2, -ly/2],
              cmap='binary_r')


def plot_firing_rate_per_rfc(rates, xs, ys, t, ax=None, kwargs={}):
    """Plots the given firing rates in time.

    Args:
        rates (ndarray of float): the firing rate for every time step of each
            LGN cell.
        xs (list of float): x coordinates of the receptive fields.
        ys (list of float): y coordinates of the receptive fields.
        t (list of float): time.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    for index, (x, y) in enumerate(zip(xs, ys)):
        ax.plot(t, rates[:, index], label="x={}␣y={}".format(x, y))

    ax.set_xlabel('Time␣(sec)')
    ax.set_ylabel('Firing␣rate␣(Hz)')

    ax.set(**kwargs)


def plot_stimulus_per_rfc(stimulus, xs, ys, t, ax=None):
    """Plots stimulus value per receptive field based on the position.

    Args:
        stimulus (ndarray of float): stimulus for every time step.
        xs (list of float): x coordinates of the receptive fields.
        ys (list of float): y coordinates of the receptive fields.
        t (list of float): time.
        ax (axes object, optional): specified target axes. Defaults to None.
    """

    if ax is None:
        fig, ax = plt.subplots()

    for x, y in zip(xs, ys):
        sine_xy = stimulus[:, int((y+ly/2)/dy), int((x+lx/2)/dx)]
        ax.plot(t, sine_xy, label='x={}␣y={}'.format(x, y))


def plot_postition_per_rfc(xs, ys, ax=None):
    """Plots a scatter plot of the positions of the centers of each receptive
    field.

    Args:
        xs (list of float): x coordinates of the receptive fields.
        ys (list of float): y coordinates of the receptive fields.
        ax (axes object, optional): specified target axes. Defaults to None.
    """

    # plot position per RF
    if ax is None:
        fig, ax = plt.subplots()

    for x, y in zip(xs, ys):
        ax.scatter(x, y, label='x={}␣y={}'.format(x, y))


def plot_receptive_field(kernels_on_ex, kernels_off_ex,
                         kernels_on_inh, kernels_off_inh,
                         stimulus, lx, ly, t=0, ax=None, kwargs={}):
    """Plots receptive fields and the stimulus on 2D plot.

    Args:
        kernels_on_ex (ndarray of float): kernels for excitatory center ON
            receptive field.
        kernels_off_ex (ndarray of float): kernels for excitatory center OFF
            receptive field.
        kernels_on_inh (ndarray of float): kernels for inhibitory center ON
            receptive field.
        kernels_off_inh (ndarray of float): kernels for inhibitory center OFF
            receptive field.
        stimulus (ndarray of float): stimulus for every time step.
        lx (float): dimensions (x axes).
        ly (float): dimensions (y axes).
        t (int, optional): time index. Defaults to 0.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    cmap = colors.ListedColormap(['blue', 'red'])
    bounds = [-1, 0, 1]
    norm = colors.BoundaryNorm(bounds, 2)
    kernel_kwargs = {'alpha': 0.3, 'cmap': cmap, 'norm': norm}

    plot_stimulus(stimulus, t, lx, ly, ax)
    plot_kernels(kernels_on_ex, lx, ly, kernel_kwargs, ax)
    plot_kernels(kernels_off_ex, lx, ly, kernel_kwargs, ax)
    #plot_kernels(kernels_on_inh, lx, ly, kernel_kwargs, ax)
    #plot_kernels(kernels_off_inh, lx, ly, kernel_kwargs, ax)
    ax.set(**kwargs)

    ax.set_xticks([])
    ax.set_yticks([])


def plot_mem(lif_state_monitor, ex_presyn_spike_monitor, in_presyn_spike_monitor,
             show_pre_spikes=True, ax=None, kwargs={}):
    """Plots the membrane potential (black) and the time of presynaptic
    excitatory (blue) and inhibitory (red) input.

    Args:
        lif_state_monitor (brian object): state monitor of the LIF neuron.
        ex_presyn_spike_monitor (brian object): excitatory presynaptic spike
            monitor.
        in_presyn_spike_monitor (brian object): inhibitory  presynaptic spike
            monitor.
        show_pre_spikes (bool, optional): if true plots the presynaptic spikes,
            below the membrane potential, otherwise not. Defaults to True.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    # Plot membrane potential
    ax.plot(lif_state_monitor.t/second, lif_state_monitor[0].v/mV, color='k')
    ax.set(**kwargs)

    if not show_pre_spikes:
        return

    # Plot time of pre-synaptic spikes
    offset = -80
    for spike_ex, spike_inh in zip(ex_presyn_spike_monitor.spike_trains().values(),
                                  in_presyn_spike_monitor.spike_trains().values()):

        ax.plot(spike_ex/second, np.zeros_like(spike_ex/second) +
                offset, '|', color='b', alpha=0.4)
        ax.plot(spike_inh/second, np.zeros_like(spike_inh/second) +
                offset, '|', color='r', alpha=0.4)

        offset -= 1

    ax.set_xlabel('Time (sec)')
    ax.set_ylabel('Membrane Potential (mV)')


def plot_bursts(lif_state_monitor, bursts_all, burst_groups, dt, ax=None,
                kwargs={}):
    """Plots the membrane potential indicating the bursts (red line) and
    burst-groups (green line).

    Args:
        lif_state_monitor (brian object): state monitor of the LIF neuron.
        bursts_all (list of int): indices of all bursts.
        burst_groups (list of int): indices of burst-groups.
        dt (float): time step.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    for b in bursts_all:
        plt.axvline(x=b*dt, color='r')

    for bg in burst_groups:
        plt.axvline(x=bg*dt, color='g')

    ax.plot(lif_state_monitor.t/second, lif_state_monitor[0].v/mV, color='k')
    ax.set(**kwargs)


def plot_pold(pold_ex, pold_in, prespiking_potential, time,
              thresh, ax=None, kwargs={}):
    """Plots the membrane potential with different color each period i.e.
    excitatory(blue), inhibitory(orange) and pre-spiking(green).

    Args:
        pold_ex (list of float): membrane potential of excitatory
            period.
        pold_in (list of float): membrane potential of inhibitory
            period.
        prespiking_potential (list of float): membrane potential of pre-spiking
            period.
        time (list of float): time.
        thresh (float): firing threshold potential.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    ax.plot(time, pold_ex, label='excited', linewidth=1)
    ax.plot(time, pold_in, label='inhibited', linewidth=1)
    ax.plot(time, prespiking_potential, label='pre-spiking', linewidth=1)
    ax.axhline(y=thresh, color='r', linestyle='-', label='Resting potential')

    ax.set_xlabel('Time (sec)')
    ax.set_ylabel('Pold (mV)')

    ax.legend()
    ax.set(**kwargs)


def plot_pnew(pnew_ex, pnew_in, pnew_prespiking, pold_all, time, interval,
              ax=None, kwargs={}):
    """Plots Pnew  with different color each period i.e. excitatory(blue),
    inhibitory(orange) and pre-spiking(green).

    Args:
        pnew_ex (list of float): Pnew values for excitatory period.
        pnew_in (list of float): Pnew values for inhibitory period.
        pnew_prespiking (list of float): Pnew values for pre-spiking period.
        pold_all (list of float): membrane potential (Pold).
        time (list of float): time.
        interval (float): duration of interval used for Pnew calculation.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    ax.plot(time, pold_all, 'k-', alpha=0.2)
    #ax.plot(time, pold_all, 'ko', markersize=1, alpha=0.3)

    ax.axhline(y=0, color='k', alpha=0.2)

    ax.plot(time, pnew_ex, linewidth=1)
    ax.plot(time, pnew_in, linewidth=1)
    ax.plot(time, pnew_prespiking, linewidth=1)

    ax.set_xlabel('Time␣(sec)')
    ax.set_ylabel('Pnew␣(mV)')

    ax.set_title('interval␣{:.3f}␣msec'.format(interval))
    ax.set_xlim(0, 4)

    ax.set(**kwargs)


def plot_std(std_ex, std_in, std_prespiking, std_all, intervals,
             std_exp_df=None, ax=None, kwargs={}):
    """Plots the standard deviation of Pnew  with different color each period
    i.e. excitatory(blue), inhibitory(orange) and pre-spiking(green).

    Args:
        std_ex (list of float): standard deviation of Pnew, during the
            excitatory period for all the given intervals.
        std_in (list of float): standard deviation of Pnew, during the
            inhibitory period for all the given intervals.
        std_prespiking (list of float): standard deviation of Pnew, during the
            pre-spiking period for all the given intervals.
        std_all (list of float): standard deviation of Pnew, during the
            total duration for all the given intervals.
        intervals (list of floats): duration of intervals used for Pnew
            calculation.
        std_exp_df (pandas dataframe, optional): The corresponding results of
            the experimental data. Defaults to None.
        ax (axes object, optional): specified target axes. Defaults to None.
        kwargs (dict, optional): pyplot arguments. Defaults to {}.
    """

    if ax is None:
        fig, ax = plt.subplots()

    if not (std_exp_df is None):
        ax.plot(intervals, std_exp_df['std_ex'],
                color='C0', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_in'],
                color='C1', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_prespiking'],
                color='C2', linestyle='--', alpha=0.4)
        ax.plot(intervals, std_exp_df['std_all'],
                color='k', linestyle='--', alpha=0.4)

    ax.plot(intervals, std_ex, color='C0', label='excited')
    ax.plot(intervals, std_in, color='C1', label='inhibited')
    ax.plot(intervals, std_prespiking, color='C2', label='pre-spiking')
    ax.plot(intervals, std_all, label='all', color='k')

    ax.set_xlabel('Time interval (msec)')
    ax.set_ylabel('Standard deviation')

    ax.legend()
    ax.set(**kwargs)
```

Listing A.15: utils.py - Misc. utilities.

```
import numpy as np


def rotate_point(x, y, theta):
    """Rotate the given point (x, y) at an angle theta.

    Args:
        x (float): x coordinate.
        y (float): y coordinate.
        theta (float): orientation angle.

    Returns:
        (float, float): rotated x and y coordinates.
    """

    xx = x*np.cos(theta) - y*np.sin(theta)
    yy = x*np.sin(theta) + y*np.cos(theta)
    return xx, yy


def rotate_point_arr(xs, ys, theta):
    """Rotates multiple points at an angle theta.

    Args:
        xs (list of floats): x coordinates.
        ys (list of floats): y coordinates.
        theta (float): orientation angle.

    Returns:
        (lists of floats, lists of floats): two lists containing the
```

```python
            rotated x and y coordinates.
    """

    xs_rot = np.zeros(xs.size)
    ys_rot = np.zeros(ys.size)

    for i, (x, y) in enumerate(zip(xs, ys)):
        xs_rot[i], ys_rot[i] = rotate_point(x, y, theta)
    return xs_rot, ys_rot


def concat_arr(a, b, c, d):
    """Concatenates four lists into one.

    Args:
        a (list of float): a list to be concatenated.
        b (list of float): a list to be concatenated.
        c (list of float): a list to be concatenated.
        d (list of float): a list to be concatenated.

    Returns:
        list of float: the concatenation of the given lists.
    """

    e = np.concatenate((a, b), axis=0)
    f = np.concatenate((c, d), axis=0)
    return np.concatenate((e, f), axis=0)


def get_index_bounds(indexes):
    """Returns the bounds (first and last element) of consecutive numbers.

    Example:
    [1 , 3, 4, 5, 7, 8, 9, 10] -> [1, 3, 5, 7, 10]

    Args:
        indexes (list of int): integers in an ascending order.

    Returns:
        list of int: the bounds (first and last element) of consecutive numbers.
    """

    index_bounds = [indexes[0]]

    for i, v in enumerate(indexes):
        if i == len(indexes)-1:
            index_bounds.append(v)
            break

        if v+1 == indexes[i+1]:
            continue
        else:
            index_bounds.append(v)
            index_bounds.append(indexes[i+1])

    return index_bounds


def get_interval_indexes(interval, indexes):
    """Returns the given indexes, after removing <interval> elements from the
    beginning and ending of each sequence of consecutive numbers.

    Args:
        interval (int): number of elements to be removed from beginning and
            ending of each sequence of consecutive numbers.
        indexes (list of int): integers in an ascending order.

    Returns:
        list of int: the given indexes, after removing <interval> elements from
            the beginning and ending of each sequence of consecutive numbers.
    """

    if len(indexes) == 0:
        return 0

    index_bounds = get_index_bounds(indexes)

    interval_indexes = []

    for i in range(0, len(index_bounds)-1, 2):
        if index_bounds[i]+interval >= index_bounds[i+1]:
            continue

        interval_indexes.extend(range(index_bounds[i]+interval,
                                      index_bounds[i+1]-interval+1))

    return interval_indexes
```