Abstract

The physical processes that creates electrical signals in neurons are well understood, but how the signals are processed into actions and thoughts has yet to receive a scientifically robust answer Cell type classification is of high importance because the function of different neurons is still largely a mystery.

Contents

1 Introduction **5**

2 Theory **7**
- 2.1 The Neuron . . . 7
- 2.2 Electrical Activity . . . 8
- 2.3 Action Potential . . . 9
- 2.4 Neuron Models . . . 9
- 2.5 Electrodes . . . 9
- 2.6 Calculating Extracellular Potential . . . 9
- 2.7 Neuron & LFPy . . . 11

3 Methods **13**
- 3.1 Spike Width Measurement . . . 13
- 3.2 Spike Amplitude Measurement . . . 14
- 3.3 LFPyUtil . . . 14
- 3.4 Blue Brain . . . 20

4 Results **21**
- 4.1 Pettersen & Einevoll (2008) Reproduction . . . 21
- 4.2 Optimal Width Definition . . . 26

5 Discussion **29**

A Appendix **31**

Bibliography **35**

Introduction

Since the conception of neuroscience the neurons function have been studied on many levels from the properties of the cell membrane to clustered networks of neurons.

There are several types of neurons and it was early noticed that different kind of neurons gave different types of signals.

This was of much interest because

Modern types of classification uses genotype?, the structure of the neuron and the electric signal.

It is useful to seperate interneurons from pyramidal neurons as pyramidal neurons are excitatory and interneurons are inhibitory.

Is it possible to seperate interneurons from pyramidal neurons soly based on the shape of the action potental.

Knowing the neuron type is important for research. While doing single cell recordings on alive subjects the researchers are recording in the dark. The electrodes only pick up on electrical signals from the brain, so the researcher does not know exactly what they cell they are recording.

Surely the different types of neurons are specialized at certain functions

In this article we show that interneurons can be seperated from pyramidal neurons based on data and models from the Blue Brain project. The program makes an easy way to do the same analysis on future models as long as the models can be loaded with LFPy.

Theory

2.1 The Neuron 7
2.2 Electrical Activity 8
2.3 Action Potential 9
2.4 Neuron Models 9
2.5 Electrodes 9
2.6 Calculating Extracellular Potential 9
2.7 Neuron & LFPy 11

The Neuron

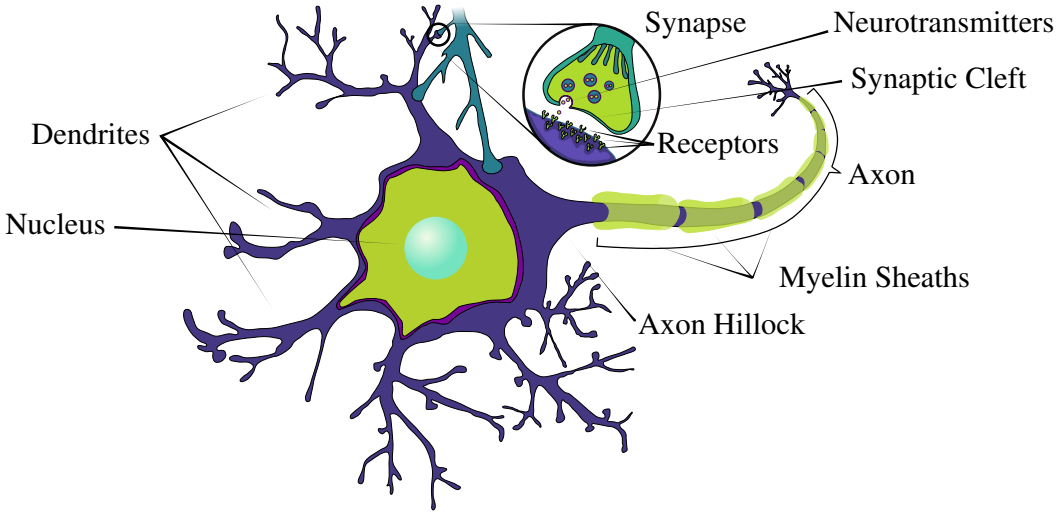

Figure 2.1: Stuff about this neuron.

Neurons are electrically excitiable cells that are a fundamental part of all brain functions. Other names include nerve cells, neurone or more colloquially brain cells. Neurons form in big networks which process information, and in the human brain there is an estimated 10^{11} neurons.

Special proteins in the cell membrane enables the neuron to fire action potentials when it is electrically excited. These action potentials are sharp voltage changes that propagates through the full structure of the neuron. The same properties that makes the neuron able to fire makes the action potential regenerative, meaning it will propagate without decay.

The body of the neuron, the soma, has dendrites and the axon attached to it. The dendrites and the axon are very thin branching structures with a width usually in the order of $1\,\mu\mathrm{m}$. While

neurons often have many dendrites directly attached to the soma there is only one axon attached to the soma at the axon hillock. The axon can branch several times before it ends and usually connects to the dendrites of other neurons via synapes.

The synapes are electrically sensitive which allows information to pass between neurons. Though the majority of all synapes are axo-dendritic (axon to dendrite), other junctions are also possible. Other junctions include but are not limited to, dendrite to dendrite, axon to axon and axon to blood vessel. When an action potential reaches a synapse it will activate the synapse and pass information to the connect neuron. The information that is passed along depends on the type of synapse, and if it is of a chemical or electrical type.

Electrical Activity

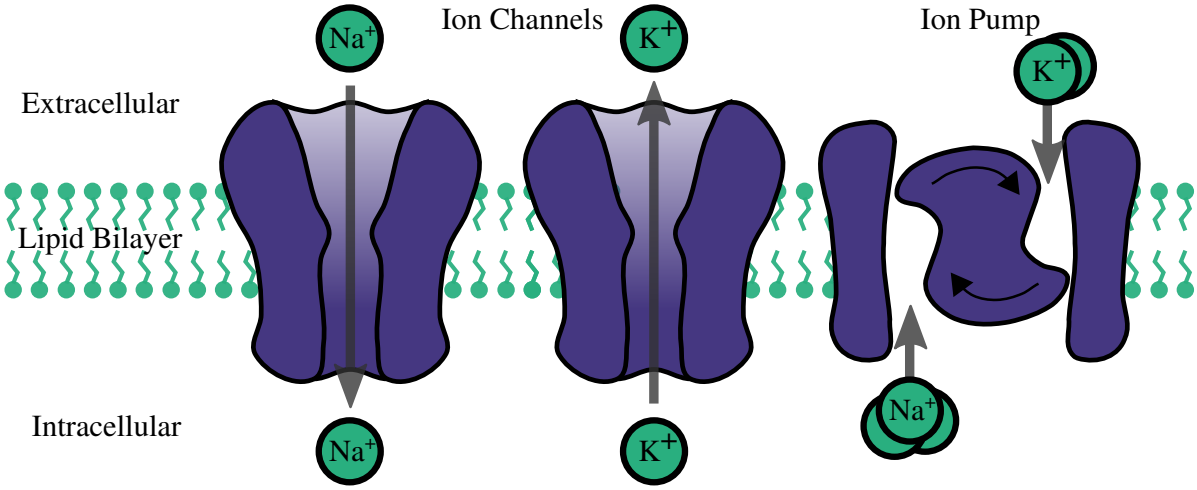

Figure 2.2: Something about ion pumps and channels.

The potential difference between the inside and outside the neurons are caused by different concentrations of ions in the extracellular and intracellular medium. The ions cannot pass through the cell membrane as it consists of a $5\,\mathrm{nm}$ lipid bilayer which is mostly impenetrable to ions.

In the membrane sits ion channels and ion pumps which can have selective permeability to ions, this creates a potential gradient across the membrane. The most significant ions in this process are Sodium (Na^+), Potassium (K^+), Calcium (Ca^{2+}), Magnecium (Mg^{2+}) and Chloride (Cl^-). Ion channels are divided between passive channels and active channels where the active channels can change permeability under certain conditions while passive channels have a constant permeability.

The ion pumps differ from the channels by activly transporting certain ions through the membrane. For instance, the Sodium-Pottasium exchanger pushes two K^+ ions out of the cell for every three Na^+ it pushes into the cell. Doing this creates a net loss of charge inside the cell and the pump is therefore electrogenic. Not all pumps are electrogenic, the Sodium-Hydrogen exchanger transports H^+ and Na^+ without effecting the net charge. For each H^+ ion out of the cell the pump pushes one Na^+ into the cell.

To understand the electrical activity of neurons it is useful to view the neuron as an electronic circuit where the ion channels, ion pumps and the membrane serve as different electronic components.

Hodgkin & Huxley [15], Connor & Stevens [5], and Sterratt et al. [24]

Action Potential

Action potentials are sharp increases in the membrane potential followed by a less sharp decrease towards the resting potential. In the the depolarization phase the potential rises towards the peak magnitude, while in the repolarization phase the potential decreases towards the cells resting potential. When the potental is below the resting potential it reaches the afterhyperpolarization phase before it returns to its resting potential.

Neuron Models

There are multiple models for neurons, some of the main groups are point models and compartmental models. List many models? Multi-compartmental models can be useful to understand the processing of neurons with complex morphological structures

Electrodes

Calculating Extracellular Potential

The extracellular potential is the electric potential generated from the transmembrane currents in the neurons. When a neuron fires this can be seen from the extracellular potential which will have a spike which is similar to the intracellular spike.

By modelling the neuron as compartments and approximating each compartment as a spherical volume current source at position $\mathbf{r_0}$, the potential at at position $\mathbf{r}$ at time t will be,

$$\mathbf{E}(\mathbf{r}, \mathbf{t}) = \frac{1}{4\pi\sigma}\frac{I_0(t)}{|\mathbf{r} - \mathbf{r_0}|} \tag{2.1}$$

$$\mathbf{E}(\mathbf{r}, \mathbf{t}) = \sum_{n=1}^{N} \frac{1}{4\pi\sigma}\frac{I_n(t)}{|\mathbf{r} - \mathbf{r_0}|} \tag{2.2}$$

Potential from compartments modelled as line sources.

$$\mathbf{E}(\mathbf{r}, \mathbf{t}) = \frac{1}{4\pi\sigma}\sum_{n=1}^{N} I_n(t)\frac{dr_n}{|\mathbf{r} - \mathbf{r_0}|} \tag{2.3}$$

$$= \frac{1}{4\pi\sigma}\sum_{n=1}^{N} I_n(t)\frac{1}{\Delta s_n}\log\left|\frac{\sqrt{h_n^2 + \rho_n^2} - h_n}{\sqrt{l_n^2 + \rho_n^2} - l_n}\right| \tag{2.4}$$

Taken from Lindén et al. [17]

This equation rests on two assumptions,

1. The permeability μ of the extracellular medium is the same as that of vacuum μ_0.
2. The quasistatic approximation which lets the time derivatives, $\partial E/\partial t$, be ignored as source terms. See **??**

The extracellular potential can be calculated using Maxwell's equations and the continuity equation if the spatial distribution (morphology) of transmembrane currents and the extracellular conductivity is known.

In the quasistatic approximation, since $\nabla \times \mathbf{E} = \mathbf{0}$, the electric field can be expressed with a scalar potential.

Forward problem = calculate the potential from the current source, inverse problem is used in magnetoenchephalography (important). The amplitude of a spike in the extracellular potential is usually in the magnintude of $< 200\mu$V. The noise of electrodes vary, but can be as much as 20μV. This limits the range electrodes can record from.

The currents sum to zero, while the spike is very visible, there are many small currents in the dendrites with opposite current. ([13])

The extracellular spike width tend to increase with distance from soma because of the neuronal morphology. This article used a passive neuron model with different morphologies to show that the spike width increases with distance to soma. The spike amplitude also decreases with distance to soma and seems to follow a power law. ([22]).

The shape of extracellular spikes are mainly depedent on the membrane currents and the morphology of the cell. Some of the effects from the morphology of the cell are increased spike width and decreased amplitude from distance to soma.

Many things here from around page 245. When the conductivity σ and the current generators are know, Maxwell's equations and the continuity equation equation can be used to calculate the electric field E and magnetic field B. (TODO: Copied text) ([13])

Background

Recording is usually done using electrodes, this makes recording the membrane potential more challenging than recording from the extracellular medium as the electrode has to be very close or inside the cell. At the time of writing, recording the membrane potential of a concious subject is nearly impossible, this makes understanding extracellular potentials vital for current research.

Early calculations was done by Rall 1962 investigating the interaction between action potentials and synapes using cylinders as the current source. (TODO: Read article, make more understandble.) Holt and Koch 1999 added comparmental models to reconstruct pyramidal neurons.

The information about the transmembrane current is usually difficult to obtain, as well as the morphology.

Neuron & LFPy

LFPy is a Python module that uses Neuron and the mentioned methods to calculate the electric field outside the neuron. [17]

Background

Methods

Methods mentioned here have been developed spesifically for this research.

3.1 Spike Width Measurement 13
3.2 Spike Amplitude Measurement 14
3.3 LFPyUtil 14
3.3.1 About 14
3.3.2 Minimal Working Examples 14
3.3.3 List of Simulations 19
3.4 Blue Brain 20

Spike Width Measurement

Most extracellular spikeshapes has a minimum value greater than the maximum value, but this is not always the case when measuring far away from soma.

Width Type I - Peak-to-peak:
Width is measured as the time from the minimum potential to the maximum. This is the time from the polarization phase to the afterhyperpolarization phase.

Width Type II - Width at Half Amplitude:
Width is measured as the duration the spike is below half amplitude of the signal measured from the baseline at the start of the signal.

Width Type II - Width at Half Amplitude:
Width is measured as the duration the spike is below half amplitude of the signal measured from the baseline at the start of the signal.

There are three prevailant ways of defining spike width which has been named Type I, Type II and Type III.

Type I: Width is measured from "peak-to-peak". This method can easily be implemented by measuring the width from minimum to maximum value. In the cases where the spike is flipped or is not well defined, this definition is not correct.

As such the implementation of this definition was done by measuring the time from the the maximum absolute value to the preceding maximal absolute value on the opposing side.

Spike Amplitude Measurement

Amplitude is easier to calculate than the width of a spike, but there are still different ways to define the amplitude. In most cases the amplitude is defined as the distance from a baseline, in a similar manner as a sinusoid. The baseline is not always well define with spikeshapes. For the intracellular potential the baseline can for instance be set at the firing threshold, but for extracellular spikes this is not possible. Maybe more common is to set the baseline as the value during quiescence. For modelling purposes this is can provide difficulties.

LFPyUtil

About

LFPyUtil is a python package that was created for this project with the purpose to simplify the simulation pipeline for multiple neurons and creating an easy to use interface when developing new simulations. LFPyUtil extends and uses the package LFPy to accomplish this. Simulations can be run in parallel and data from simulations can automatically be saved and loaded to avoid unnecessary processing time.

LFPy is a Python package created to calculate extracellular potentials. Another major feature is wrapping the cell model and electrodes from the NEURON simulation environment into Python objects, such as the `LFPy.Cell` and `LFPy.StimIntElectrode` classes. This makes working with NEURON more pythonic as it can be argued that NEURON is state-based. That is, even though the Python interface of NEURON uses objects, the objects are bound to the state system. In practical terms this means that all functions and variables with NEURON are global static.

While NEURON has support for paralell processing of single simulations it does not have any inherit support for running multiple independent simulations. For "embarrasingly parallel" situations like this, users have had to resort to creating their own methods to start each simulation independently. For instance, one solution would be to use a Python script to run other scripts through the command line, effectivly starting new processes. In addition there is no function to reset the simulation environment which can make previous simulations affect later ones.

LFPyUtil uses Python's multiprocessing package to run independent simulations and thus overcomes some of the shortcomings of NEURON and LFPy.

Minimal Working Examples

These examples show how to create a new custom simulation from scratch and how to use them with LFPyUtil. A basic understanding of object-oriented programming and Python is required.

To run a simulation LFPyUtil must first have a `LFPy.Cell` object it can use to interact with the model. The cell object gives access to functions such as `Cell.simulate()` which starts the NEURON simulation. A template of such a function can be seen in listing 1. A fully working example of such a function can be seen in the appendix (**??**).

Listing 1: load_model_simple.py

```
import LFPy

def get_cell(neuron_name):
    """
    Load a spesific model based on the input string and return
    a LFPy Cell object

    :param string neuron_name:
        String to identify the neuron model that will be loaded.
    :returns:
        Cell object from LFPy.
    """

    cell = LFPy.Cell( some cell parameters ... )

    return cell
```

LFPyUtil use subclasses of the class `LFPyUtil.sims.Simulation`. to organize simulations. Listing 2 shows a very minimal example of such a subclass. If the functions `simulate`, `process_data` and `plot` are not overrided, a `NotImplementedError` will be raised.

Listing 2: new_simulation_class_simple.py

```
from LFPy_util.sims import Simulation

class CustomSimulation(Simulation):
    def __init__(self):
        # Inherit the LFPyUtil simulation class.
        Simulation.__init__(self)
        # These values are used by the super class to save and load data.
        self.set_name("custom_sim")

    def simulate(self, cell):
        pass

    def process_data(self):
        pass

    def plot(self, dir_plot):
        pass
```

The LFPyUtil simulation class has been created to reflect four parts of a typical neuron simulation. (1) Initilization, (2) simulation/gather data, (3) processing data and (4) plotting the data. These four parts are retained in the initilization of the object, the `__init__()` function, a simulation function `simulate(cell)`, a process function `process_data()` and a plotting function `plot()`. Run parameters, plot parameters and data are stored in dictionaries in the simulation class and the variables are named `run_param`, `plot_param` and `data` respectively. The LFPyUtil simulation class has additional properties that among other things enable saving and loading data and naming those files. Because of this a new simulation class should inherit LFPyUtil's simulation class.

Listing 3 defines a simulation class that inherits the LFPyUtil simulation class, but adds some more functionality. The function `simulate` inserts a stimulus electrode and applies a current to soma with parameters defined in `__init__`. The membrane potential is then stored in the `data` dictionary. The function `process_data` creates a normalized version of the membrane potental and saves it. The function `plot` plots the membrane potential. To exemplify the use of `plot_param`, a conditional statement is used to decide whether or not to plot the normalized version.

Listing 3: new_simulation_class.py

```
import LFPy
import LFPy_util
import matplotlib.pyplot as plt
from LFPy_util.sims import Simulation

class CustomSimulation(Simulation):
    def __init__(self):
        """
        Typical initialization function, called when a new instance
        is created.
        """
        # Inherit the LFPyUtil simulation class.
        Simulation.__init__(self)
        # These values are used by the super class to save and load data.
        self.set_name("custom_sim")

        # Create some parameters that are used by the simulate method.
        self.run_param['delay'] = 100       # ms.
        self.run_param['duration'] = 300    # ms.
        self.run_param['amp'] = 1.0         # nA.

        # Create a parameters used by the plotting function.
        self.plot_param['plot_norm'] = True

    def simulate(self, cell):
        """
        Setup and starts a simulation, then gathers data.

        :param LFPy.Cell cell:
            Cell object from LFPy.
        """
        # Create an electrode with LFPy.
        soma_clamp_params = {
            'idx': cell.somaidx,
            'amp': self.run_param['amp'],
            'dur': self.run_param['duration'],
            'delay': self.run_param['delay'],
            'pptype': 'IClamp'
        }
        stim = LFPy.StimIntElectrode(cell, **soma_clamp_params)

        cell.simulate()

        # Store the data .
        self.data['soma_v'] = cell.somav
        self.data['soma_t'] = cell.tvec

    def process_data(self):
        """
        Process data from the simulate function, usually to prepare
        the data for plotting. This function creates a normalized
        version of the membrane potential.
        """
        soma_v_norm = self.data['soma_v'].copy()
        soma_v_norm -= soma_v_norm[0]
        soma_v_norm /= soma_v_norm.max()
        self.data['soma_v_norm'] = soma_v_norm

    def plot(self, dir_plot):
        """
        Plot data from the simulate and process_data function.
        This functions plots the membrane potential and the
        normalized version.

        :param string dir_plot:
            Path to the directory where plots should be saved.
        """
        plt.plot(self.data['soma_t'], self.data['soma_v'])
        # Save the plot to input directory with the name "custom_sim_mem".
        LFPy_util.plot.save_plt(plt, "custom_sim_mem", dir_plot)
        # plot_param can be used to affect the plotting.
        if self.plot_param['plot_norm']:
            plt.plot(self.data['soma_t'], self.data['soma_v_norm'])
            # Save the plot to input directory.
            LFPy_util.plot.save_plt(plt, "custom_sim_mem_norm", dir_plot)
```

The newly created simulation class can now be used to run a complete simulation as seen in listing 4.

Listing 4: first_simulation.py

```
# Import the get_cell function defined previously.
from load_model import get_cell

# Import the newly defined simulation class.
from new_simulation_class import CustomSimulation

# Load the model, using a string to identify which model will be returned.
cell = get_cell("pyramidal_1")

# Create an instance of the custom simulation class.
sim_custom = CustomSimulation()

sim_custom.simulate(cell)
sim_custom.process_data()
# Plots are stored in a folder called first_simulation.
sim_custom.plot("first_simulation")
```

If one attempted to run the simulation function in listing 3 more than once in the same program, we would experience that subsequent simulations would be affected by previous simulations. This is because a new electrode will be applied once for each call to the simulation function. With NEURON or LFPy there are no easy way to reset the environment to prevent this. One workaround is to use Python's multiprocessing package to create multiple independent processes. Doing this for every simulation will require excessive amounts of code. LFPyUtil can simplify this with tools that requires less code and are compatible with classes like the one defined in listing 3.

The class `LFPyUtil.Simulator` accepts one or more objects that inherit LFPyUtil's simulation class and can run them either in serial or paralell and either in a new independent processes or in the same process. Listing 5 shows how the newly created simulation class can be used with `LFPyUtil.Simulator` to run multiple simulations in parallel and in independt processes.

Listing 5: multiple_simulations.py

```
import LFPy_util
from load_model import get_cell
from new_simulation_class import CustomSimulation

sim = LFPy_util.Simulator()
sim.set_cell_load_func(get_cell)
# The string is passed to the get_cell function.
sim.set_neuron_name("pyramidal_1")

sim_custom_1 = CustomSimulation()
sim_custom_1.run_param['amp'] = 1.25 # nA

sim_custom_2 = CustomSimulation()
sim_custom_2.run_param['amp'] = 0.75 # nA
# Avoid sim_custom_2 overwriting the data from sim_custom_1.
sim_custom_2.set_name("custom_sim_2")

# Add the simulations to a list we want to run.
sim.push(sim_custom_1)
sim.push(sim_custom_2)

sim.simulate()
sim.plot()
```

The function `Simulator.push` adds the simulation objects to a list. When the function `Simulator.simulate()` is ran, the `simulate()` function of those objects will be

called in parallel and in independt processes. The `Simulator.plot()` function will call `process_data()` and `plot(dir_plot)` functions of those objects, and the parameter `dir_plot` will be created based on the name of the simulation class and the name of the neuron name. In this case the plots are stored in the directories `./pyramidal_1/plot/custom_sim/` and `./pyramidal_1/plot/custom_sim_2/`.

The `Simulator` class utilizes save and load features of the simulation class when the `Simulator.simulate()` function is called. This makes the dictionaries `data` and `run_param` be saved to file. If `Simulator.plot()` is ran without `Simulator.simulate()` being called first, the program will notice that the simulations are missing the `data` dictionary. It will then attempt to load the `data` and `run_param` from file. After listing 5 has been run once line 22 can thus be commented out and the data will be loaded instead of running the simulation.

LFPyUtil comes with some predefined simulations that have been used for results in this article. Listing 6 shows an example of how to use the predefined simulations.

Listing 6: predefined_simulations.py

```
1  import LFPy_util
2  from load_model import get_cell
3
4  sim = LFPy_util.Simulator()
5  sim.set_cell_load_func(get_cell)
6  sim.set_output_dir("predefined_simulations")
7
8  sim.set_neuron_name("pyramidal_1")
9
10 sim_electrode = LFPy_util.sims.MultiSpike()
11 sim_electrode.run_param['spikes'] = 3
12
13 sim_intra = LFPy_util.sims.Intracellular()
14
15 # Add the simulations to a list we want to run.
16 sim.push(sim_electrode, False)
17 sim.push(sim_intra)
18
19 sim.simulate()
20 sim.plot()
```

The class `MultiSpike` searches for the input current that will result in 3 spikes and then applies that electrode. It also plots some figures, like the input current and the spikes. `Intracellular` simulates and plots statistics about some intracellular recordings. The details of the predefined simulations can be found in section 3.3.3 List of Simulations.

Note the extra condition, `False`, in line 16. This tells the simulator that this simulation should not be run in an independent process. When the `MultiSpike` simulation is finished, the electrode it used to generate 3 spikes is still loaded in NEURON. When the simulation function of `Intracellular` runs, the electrode will excite the neuron and generate 3 spikes. This feature can be used to link simulations and makes the simulations modular.

The previous examples use only one neuron model. To be able to run many different models it is usefull to define a function such as the one in listing 7.

Listing 7: simulator.py

```
import LFPy_util
from load_model import get_cell

def get_simulator(neuron_name):
    sim = LFPy_util.Simulator()
    sim.set_cell_load_func(get_cell)
    sim.set_output_dir("multiple_neurons")
    sim.set_neuron_name(neuron_name)

    sim_electrode = LFPy_util.sims.MultiSpike()
    sim_electrode.run_param['spikes'] = 3

    sim_intra = LFPy_util.sims.Intracellular()

    sim.push(sim_electrode, False)
    sim.push(sim_intra)

    return sim
```

To do the same simulation as in listing 6, one could write:

Listing 8: run_simulator.py

```
from simulator import get_simulator

sim = get_simulator("pyramidal_1")
sim.simulate()
sim.plot()
```

Listing 9 runs the two predefined simulations with two different neuron models. The class `LFPyUtil.SimulatorManager` takes a list of neuron names and passes them to the `get_simulator(neuron_name)` function in listing 7. The parameter `neuron_name` are the elements of the list of neuron names. After running the code once, `simm.simulate()` can be commented out and the simulations will load data instead of running the simulations.

The final simulation uses three files: listing 1, listing 7 and listing 9 and 2 predefined simulation classes.

Listing 9: multiple_neurons.py

```
from simulator import get_simulator

neurons = ["pyramidal_1", "pyramidal_2"]

simm = LFPy_util.SimulatorManager()
# Number of LFPy_util.Simulator objects  that will run in parallel.
simm.concurrent_neurons = 8
simm.set_neuron_names(neurons)
simm.set_sim_load_func(get_simulator)

simm.simulate()
simm.plot()
```

List of Simulations

SphereRand: SphereRand places electrodes placed in uniformly distributied locations around the soma within a default radius of $50\,\mu\mathrm{m}$. Spike timing is detected by thresholding the soma membrane potential. That timing is applied to all electrodes such that all electrodes measure the same part of the simulation. If the signal has several spikes the spike index must be supplied, the default setting uses the first spike.

Blue Brain

The Blue Brain project released XXX models based upon neurons from the hind-limb somatosensory cortex from 2-week-old Wistar Han rats.

The neuron models are based on the classication criteria set by the Blue Brain team there is only 2 classes of pyramidal neurons avaiable in L5, while the diversity in interneuron models are much greater. The number of available models were based on the variability of neurons depending on their morphological type and electrophysiological response to stimuli. As most of the encountered pyramidal neurons had a similar morphological structure and response to stimuli the team choose to only recoqnize two morphological types and one electrophysiological type, referred to as m-type and e-type.

Results

4.1 Pettersen & Einevoll (2008) Reproduction 21
4.1.1 Simulation 21
4.1.2 Results 24
4.2 Optimal Width Definition 26

Pettersen & Einevoll (2008) Reproduction

To verify that the simulation environment could be trusted some results from Pettersen & Einevoll [22] was replicated. Spesifically the spike width and amplitude dependency in relation to the distance from soma was compared to current results.

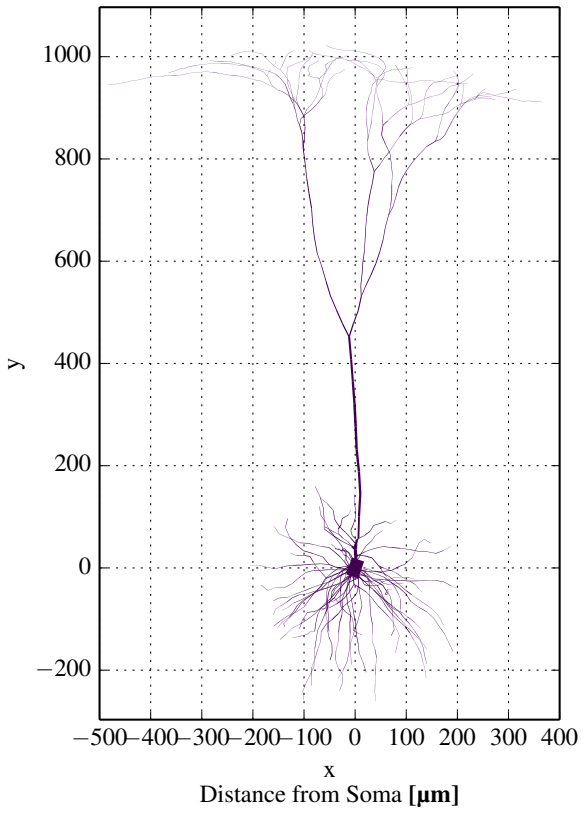

Figure 4.1: Morphology of Mainen & Sejnowski [18] cell. The apical dendrites are located along the y-axis after rotation with PCA.

Simulation

Cell: The Mainen & Sejnowski [18] morphology was used with a passive model, which is the same model used in Pettersen & Einevoll [22]. The cell was rotated using PCA (principal component analysis) on the compartment positions. This calculates three orthogonal vectors such that the positions of the compartments has the greatest variance along the first principal component, second highest along the second and third most along the third. The first principal component was made paralell to the y-axis which puts the apical dendrites along this axis (fig. 4.1).

Spike Generation: To recreate the action potential used in Pettersen & Einevoll [22] a spike was generated using the Connor-Stevens model (Connor & Stevens [5] and Connor et al. [6])

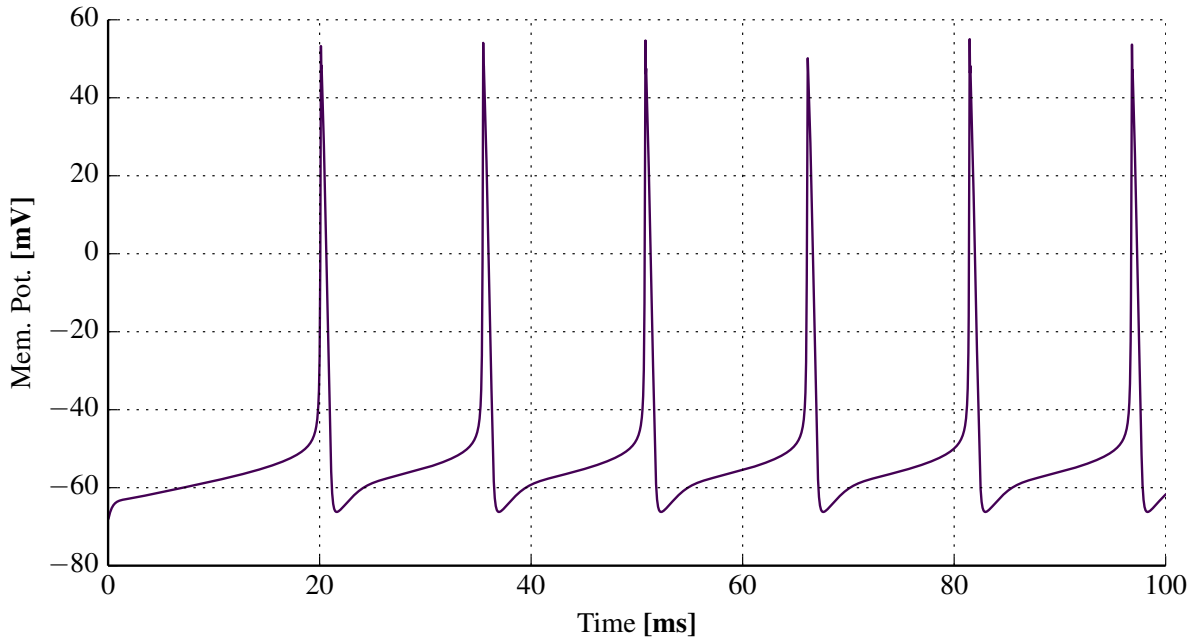

Figure 4.2: Simulation of the Connor-Stevens model using parameters from Dayan & Abbott [9]. A similar graph is shown in fig. 6.1 (B) in their book.

using the same parameters as Dayan & Abbott [9] and Pettersen & Einevoll [22]. In fig. 4.2 the Connor-Stevens simulation is shown where the second spike was used for further analysis. This spike had an amplitude of $119.49\,\mathrm{mV}$ from baseline. The baseline was estimated to $-53.26\,\mathrm{mV}$ and the peak at $53.26\,\mathrm{mV}$. These values matches Dayan & Abbott [9], but not the spike used in Pettersen & Einevoll [22] which had an amplitude of $83\,\mathrm{mV}$ from baseline. Pettersen & Einevoll [22] does not go into further detail about the creation of the action potential other than stating the action potential were similar to Dayan & Abbott [9]. The difference might be explained by the fact that action potentials from pyramidal neurons often peaks at $20\,\mathrm{mV}$, and that this was achived by scaling the original signal from the Connor-Stevens model. To compensate for the difference the action potental used in further simulations were scaled to $83\,\mathrm{mV}$ (fig. 4.3).

The input current was set to $12.6\,\mu\mathrm{A\,cm^{-2}}$. and was very carefully adjusted to make the magnitude spectrum (fig. 4.5) similar to Pettersen & Einevoll [22] figure 3. Without adjustment the magnitude spectrum tended to have a different inital value, from 6 to $8\,\mathrm{mV}$, and was not as smooth.

Parameters: Parameters for the Neuron simulation were the same as Pettersen & Einevoll [22] and the aforementioned action potential was used as a boundary condition in soma. This was accomplished by setting the membrane potential equal to the action potential in all soma sections using the `.play` vector function in Neuron. This "excites" the neuron even though there are no ion channels in the model.

Membrane resistance $R_m = 30\,\mathrm{k\Omega\,cm^{-2}}$, membrane capacitance $C_m = 1\,\mu\mathrm{F\,cm^{-2}}$, axial resistance $R_a = 150\,\Omega\,\mathrm{cm^{-2}}$, time resolution $dt = 2^{-5}\,\mathrm{ms}$. The reversal potential was set to zero.

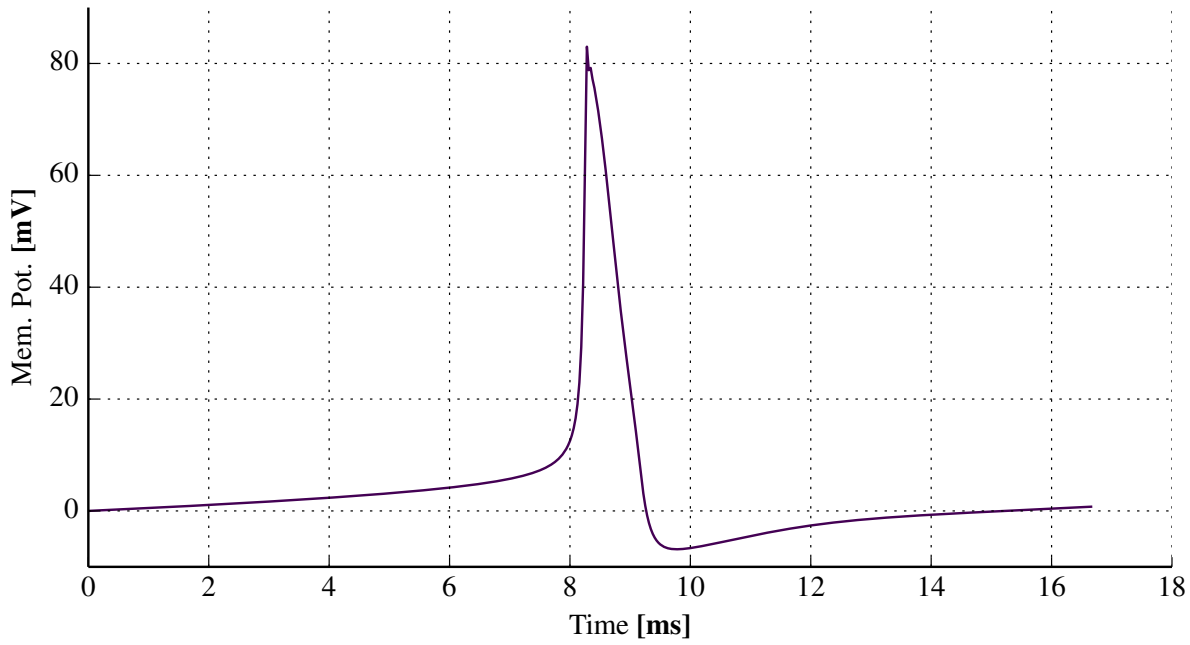

Figure 4.3: The second spike in fig. 4.2 scaled to $83\,\mathrm{mV}$ to match the action potential used in Pettersen & Einevoll [22].

Electrode Positions: Recording sites were placed in the xz-plane at 11 linearly spaced positions along 36 lines with equal angular spacing (fig. 4.4). Pettersen & Einevoll [22] states the recording positions were in the plane perpendicular to the apical dendrites, this is ensured by the rotation done with PCA and putting the electrodes in the xz-plane.

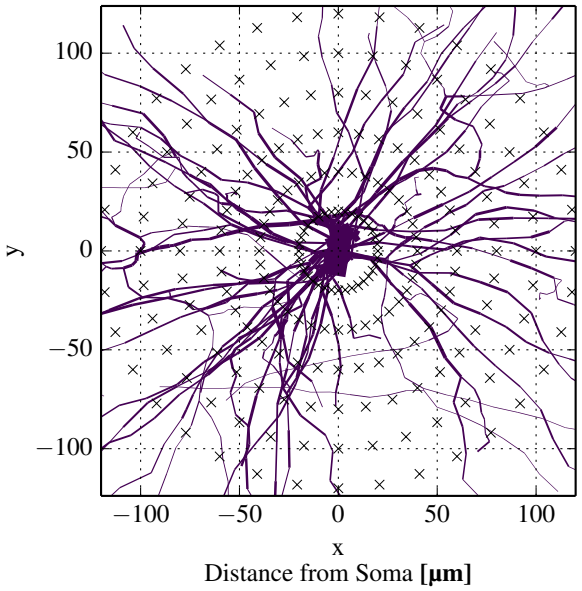

Figure 4.4: Electrode positions placed in a plane around soma perpendicular to the axis along the apical dendrites.

Spike Width & Amplitude: A baseline was set as the value at the start of the signal. Amplitude was calculated as the difference between the maximum value and the baseline. The spike width was calculated at as the width at half maximum value.

At $dt = 2^{-5}\,\mathrm{ms}$, the spike width from the Connor-Steven model was $0.5625\,\mathrm{mV}$. This is similar to the reported spike width from Pettersen & Einevoll [22] which was was $0.55\,\mathrm{ms}$. Because the dt used in their simulations was $dt = 2^{-5}\,\mathrm{ms} = 0.031\,25\,\mathrm{ms}$, their resulting spike width must have been rounded to the nearest $0.05\,\mathrm{ms}$.

Results

The action potental that was used in Pettersen & Einevoll [22] is similar to the one used here. The amplitude of the fourier transform is displayed in fig. 4.5, which is in close resemblance to the action potential in fig. 3 in the paper.

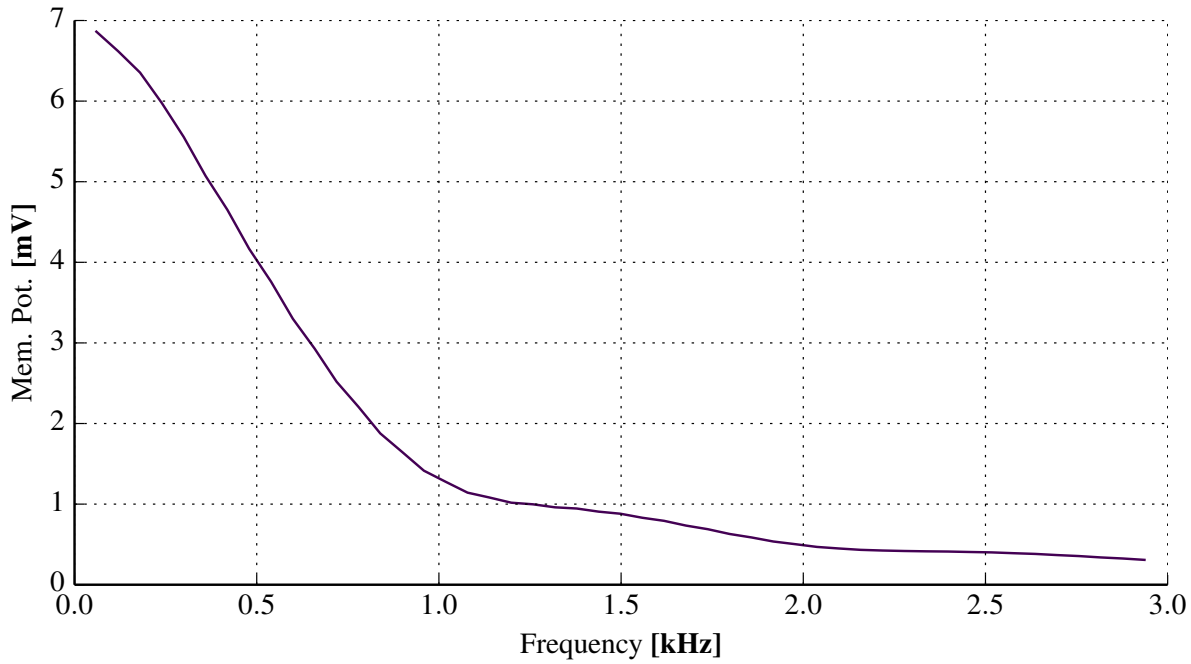

Figure 4.5: Magnitude specter of simulated somatic membrane potential.

The spike width increases with the distance from soma as seen in fig. 4.6. These results are lower than the widths reported in Pettersen & Einevoll [22].

Sudden changes in spike width was experienced with increased distance from soma. Above $200\mu V$ the most of the spikes shapes are not well defined. This was also reported in Pettersen & Einevoll [22].

Figure 4.7 shows the spike amplitude with logarithmic axes. The exact results from Pettersen & Einevoll [22] are not available but the approximate value can be seen from their plots and the exponential decay $1/r^n$ was reported as $n \sim 2$ at $20\,\mu\text{m}$ and $n \sim 2.5$ at $120\,\mu\text{m}$. Current results are not identical to those findings and have an exponent of $n = 2$ at $20\,\mu\text{m}$ and $n = 2.8$ at $120\,\mu\text{m}$. The value of the amplitude was about $350\,\mu\text{V}$ in Pettersen & Einevoll [22], but the current model only gives an amplitude of $120\,\mu\text{V}$ at $20\,\mu\text{m}$. How can I explain this descrepancy?

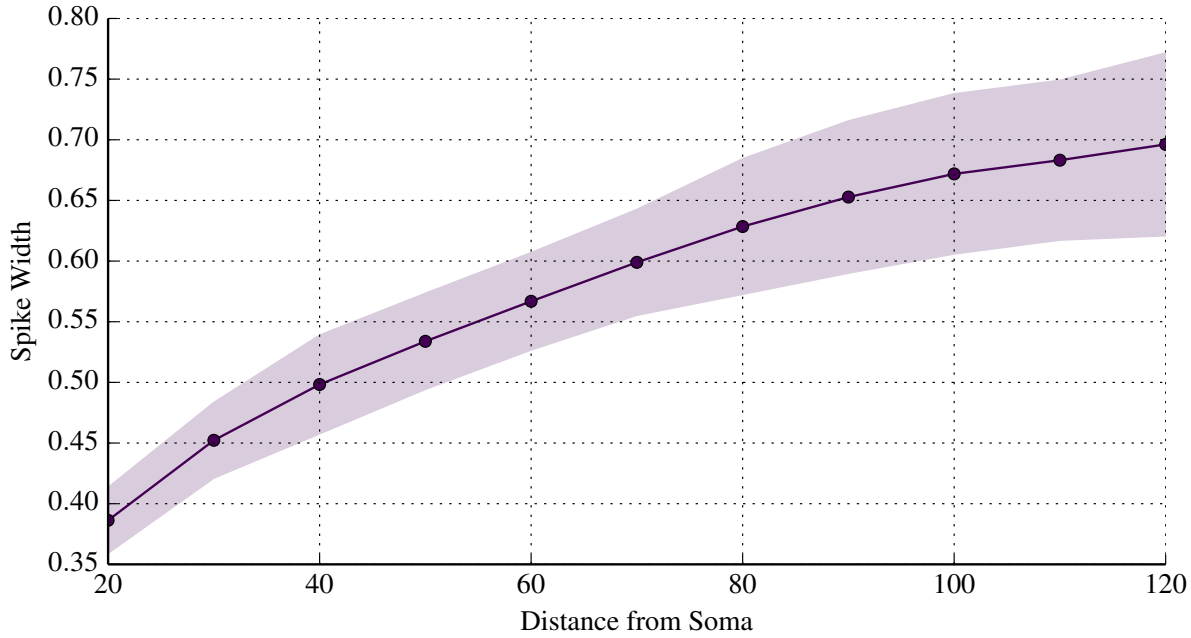

Figure 4.6: Spike width over distance. Mean +/- 1 std.

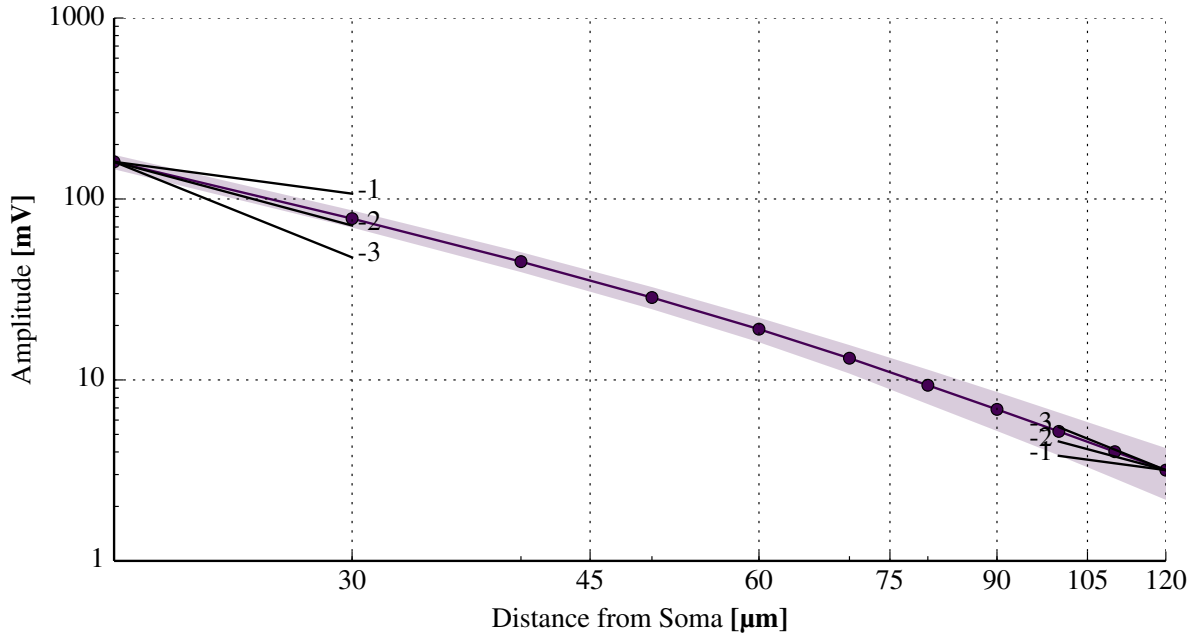

Figure 4.7: Spike amplitude over distance. Mean +/- 1 std. The power law decays $1/r$, $1/r^2$ and $1/r^3$ are shown at the leftmost and rightmost data points.

Optimal Width Definition

Many different definitions of spike width has been used to differentiate neurons, but to date it is not clear if some definitions are more suited for neuron classification than others. The two width definitions analyzed are the peak-to-peak spike definition and width at half amplitude from baseline, respectively referred to as Type I and Type II (fig. 4.8).

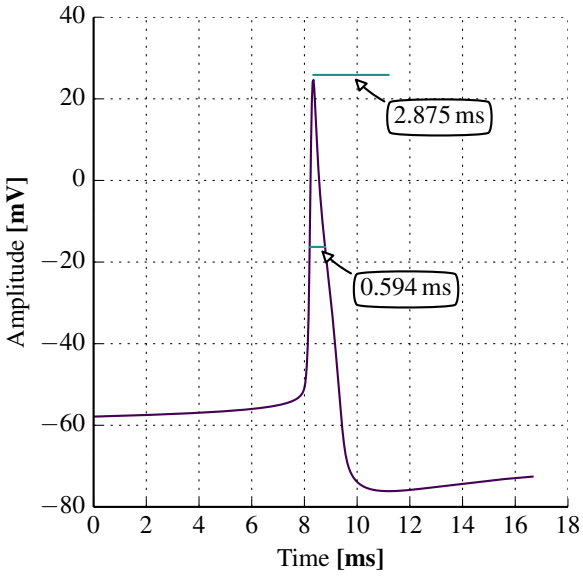

Figure 4.8: Width types.

Models: To investigate which of the two width definitions is more optimal for differentiating interneurons from pyramidal neurons, three classes of the most abundant neurons were selected from the blue brain models. Two classes of interneurons and one class of pyramidal neurons. There were two classes of pyramidal models available but as the one class had identical dendrites it was not analyzed. The number of inhibitory neuron classes were much greater, the two classes were selected by being the most abundant inhibitory neurons in the L5 area. In addition to these classes, one addition group was analyzed named "All Inter." which included all available interneuron models in L5. In this group many of the classes represent only a very small portion of the number of neurons in L5. The findings from Markram et al. [19] suggest that overall the ratio between excitatory and inhibitory neurons is around $87\% \pm 1\%$. Of these, 50% were classified as baskets cells (LBC and NBC). The interneurons from L5 were seperated into 9 m-types (morphological) and 10 e-types (electrophysiological) which gave a total number of 48 uniquie models. As such some small classes of interneurons are overrepresented in the grouped called "All Inter.".

The classes of neurons were Thick Thufted Pyramidal Cell with an early bifurcating apical thuft (TTPC2), Large Basket Cell (LBC) and Nest Basket Cell (NBC). Each of the three class had 5 seperate models where each model had different m-type but identical e-type. The e-type of TTPC2 class was continuous adapting (cAD), LBC was delayed stuttering (dSTUT) and NBC was continuous non-accommodating (cNAC) (Markram et al. [19, p. 463]). Simulations were ran using using the SphereRand (section 3.3.3) simulation .

Seperation: A good definition is recoqnized by having a better seperation between inhibitory and excitatory neurons. The results of the simulations can be seen in fig. 4.9. For both deifinitions the spike width of interneurons are smaller than the width of pyramidal neurons. These findings are in line with previously established findings. With the Type I definition the seperation between the two classes are greater in both absolute and relative value for all distances from soma. The seperation between the mean of pyramidal and interneurons for Type I were $0.40\,\mathrm{ms}$ at $30\,\mu\mathrm{m}$ and $0.60\,\mathrm{ms}$ at $100\,\mu\mathrm{m}$. For Type II the seperation was $0.15\,\mathrm{ms}$ at $30\,\mu\mathrm{m}$ and $0.35\,\mathrm{ms}$ at $100\,\mu\mathrm{m}$. These results suggests that using a type I deifinition of the spike width increases the

chance of correctly classifying the neuron class.

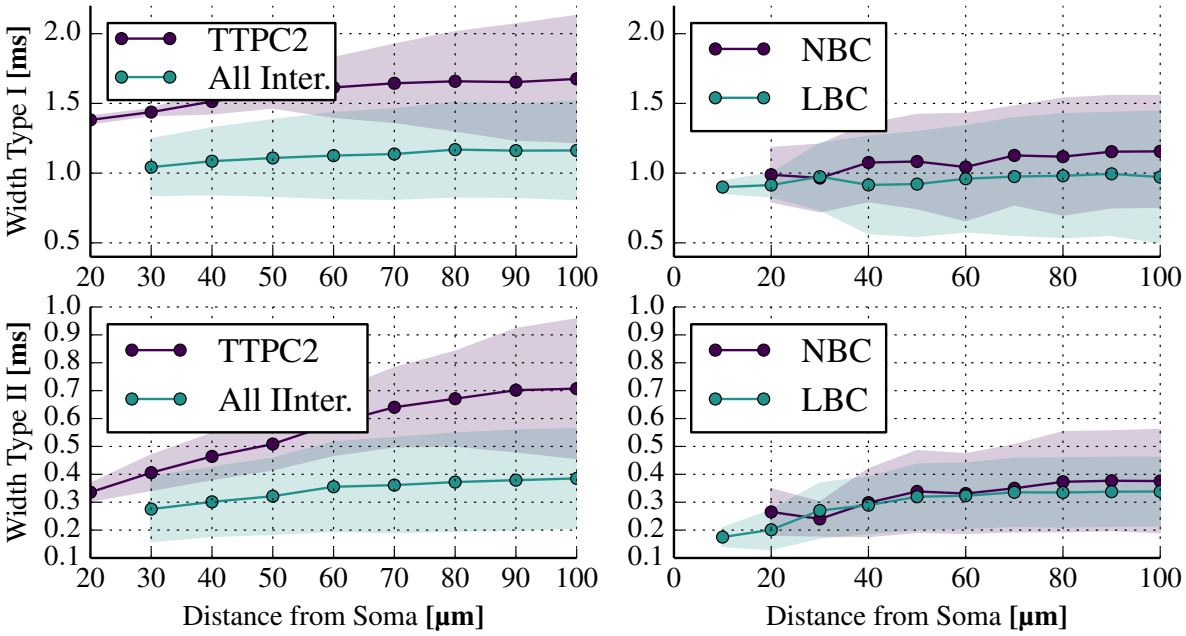

Figure 4.9: Nothing.

Variance: To evaluate the precision of the two width definitions it is not very useful to compare the raw variance because the mean values are different. An example is the σ at $50\,\mu\text{m}$ for the group "All Inter" from fig. 4.9. For Type I $\sigma = 0.25\,\text{ms}$ and $\mu = 1.1\,\text{ms}$, while for Type II $\sigma = 0.15\,\text{ms}$ and $\mu = 0.3\,\text{ms}$. The variance is lower for Type II but is also almost 50% of the mean value. The variance from Type I is higher, but is only about 25% of the mean value, which makes Type I more accurate than Type II.

The coefficient of variation, c_v, is the relative variance compared to the mean.

$$c_v = \frac{\sigma}{\mu} \tag{4.1}$$

Because the variance is of similar magnitude for both width definitions, the overall greater mean of the Type I definition results in a lesser c_v at all distances, for all groups except LBC (fig. 4.10). To minimalize errors when classifing neurons based on width it is best to minize the overlap between the two definitions. Because the seperation is higher and c_v is lower with Type I, this definition is better suited for classification.

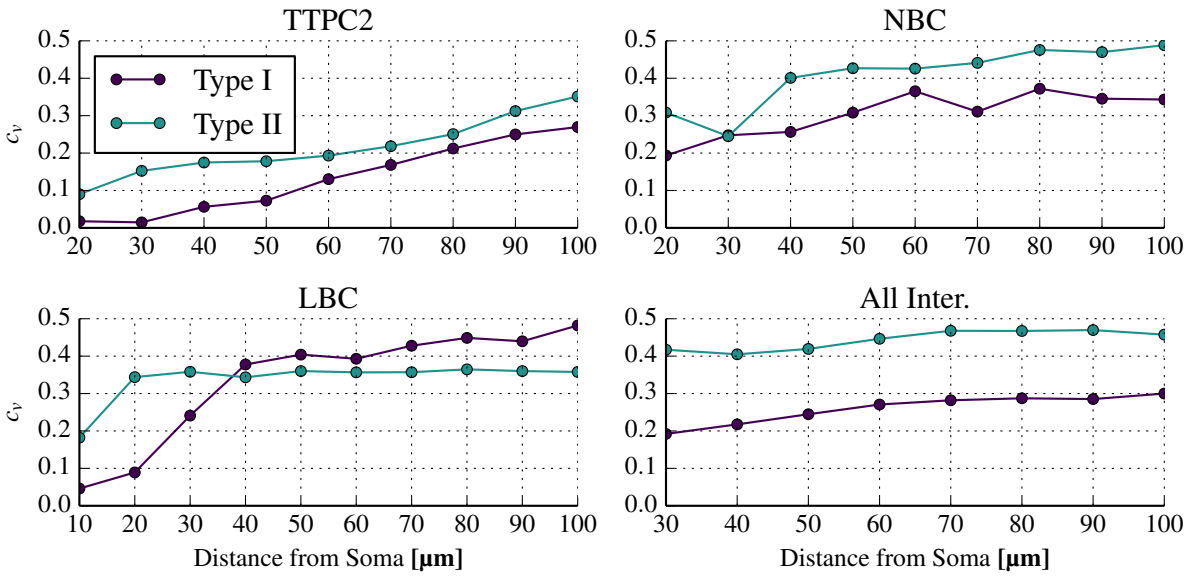

Figure 4.10: Nothing.

Discussion

Results are based on spike width and amplitude. The basis for all spike shapes are the types and concentrations of ion channels. This is the central factor that decides if the neuron has short or long action potentials. A number things has been observed that can change spike spike width. Factors that change action potentials width: * Firing frequency. * Input current. Higher current gives higher frequency. * Number of previous spikes. * Bursting behavior. * Backproagating action potentials.

Article anastassiou is consistent with Nature review, spike shape increases with frequency. This is not the case the models from Blue Brain.

When plotting spike amplitude over spike width will neurons look different.

Results: Spike width alone is often used for classifying neurons. They often have a bimodal distribution. This is the first time many simulations have been done of the extracellular potential. Previous research have only used a few models. With LFPy and LFPyUtil it will be easy to adapt new models to calculate extracellular potential.

The spike widths does not show dependency on the firing frequency or spike number. Some neurons show this feature and this should do it too.

Results: * Which width and amp definition is most suitable for differention. * Classify neurons based on spike width and amplitude. * Compare to other data. Hard to varify model, values from experiments are vary a lot. Models were based on result from their recordings. Cannot see adaption with frequency of any kind. * Is experimental distribution shape similar to mine, if so that is a good result. * Development of LFPyUtil. Future neuron models can be adapted for extracellular simulations if they include a morphology.

Appendix

```
import LFPy
import LFPy_util
import os
import numpy as np
import neuron
from glob import glob

# Location of the models.
DIR_FILE = os.path.dirname(os.path.realpath(__file__))
DIR_MODELS = os.path.join(DIR_FILE, 'res/')

def get_cell(neuron_name):
    """
    Load a spesific model based on a string and return a LFPy Cell object.

    :param string neuron_name:
        String to identify the neuron model that will be loaded.
    :return:
        Cell Object from LFPy.
    """
    original_cwd = os.getcwd()

    neuron.h.load_file('stdrun.hoc')
    neuron.h.load_file('import3d.hoc')

    # Use the neuron name to find the desired model.
    dir_nrn_model = os.path.join(DIR_MODELS, neuron_name)

    # Load mod files of the neuron.
    mechanism_mod_dir = os.path.join(dir_nrn_model, 'mechanisms')
    LFPy_util.other.nrnivmodl(mechanism_mod_dir)

    # The following .hoc files are spesific for the blue brain models.
    os.chdir(dir_nrn_model)
    #get the template name
    tmp_file = file("template.hoc", 'r')
    templatename = get_templatename(tmp_file)
    tmp_file.close()
    #get biophys template name
    tmp_file = file("biophysics.hoc", 'r')
    biophysics = get_templatename(tmp_file)
    tmp_file.close()
    #get morphology template name
    tmp_file = file("morphology.hoc", 'r')
    morphology = get_templatename(tmp_file)
    tmp_file.close()
    #get synapses template name
    tmp_file = file(os.path.join("synapses", "synapses.hoc"), 'r')
    synapses = get_templatename(tmp_file)
    tmp_file.close()
    neuron.h.load_file('constants.hoc')

    if not hasattr(neuron.h, templatename):
        # Load main cell template
        neuron.h.load_file(1, "template.hoc")
    if not hasattr(neuron.h, morphology):
        # Load morphology
        neuron.h.load_file(1, "morphology.hoc")
    if not hasattr(neuron.h, biophysics):
        # Load biophysics
        neuron.h.load_file(1, "biophysics.hoc")
    if not hasattr(neuron.h, synapses):
        # load synapses
        neuron.h.load_file(1, os.path.join('synapses', 'synapses.hoc'))


    for morphologyfile in glob('morphology/*'):
        # Instantiate the cell(s) using LFPy
        cell = LFPy.TemplateCell(
            morphology=morphologyfile,
            templatefile=os.path.join(neuron_name, 'template.hoc'),
            templatename=templatename,
            templateargs=0,
            tstartms=0,
            tstopms=300.,
            pt3d=True,
            timeres_NEURON=2 ** -5,
            timeres_python=2 ** -5,
            passive=False,
            v_init=-70,
            )
        # Reset back to the previous working directory.
```

Bibliography

[1] *Allen Cell Types Database Technical White Paper: Overview*. Tech. rep. Allen Brain Institute, 2015.

[2] Rubén Armañanzas & Giorgio A. Ascoli. "Towards the automatic classification of neurons." In: *Trends in Neurosciences* 38.5 (2015), pp. 307–318. ISSN: 0166-2236. (Visited on 01/13/2016).

[3] Romain Brette & Alain Destexhe. *Handbook of neural activity measurement*. 2012.

[4] WILLIAM H. Calvin & GEORGE W. Sypert. "Fast and slow pyramidal tract neurons: an intracellular analysis of their contrasting repetitive firing properties in the cat." In: *Journal of neurophysiology* 39.2 (1976), pp. 420–434. (Visited on 12/24/2015).

[5] J. A. Connor & C. F. Stevens. "Prediction of repetitive firing behaviour from voltage clamp data on an isolated neurone soma." In: *The Journal of Physiology* 213.1 (1971), pp. 31–53.

[6] J. A. Connor, D. Walter, & R. McKown. "Neural repetitive firing: modifications of the Hodgkin-Huxley axon suggested by experimental results from crustacean axons." In: *Biophysical Journal* 18.1 (1977), pp. 81–102. ISSN: 0006-3495. (Visited on 11/13/2015).

[7] Barry W. Connors & Michael J. Gutnick. "Intrinsic firing patterns of diverse neocortical neurons." In: *Trends in neurosciences* 13.3 (1990), pp. 99–104.

[8] Diego Contreras. "Electrophysiological classes of neocortical neurons." en. In: *Neural Networks* 17.5-6 (2004), pp. 633–646. ISSN: 08936080. (Visited on 11/12/2015).

[9] Peter Dayan & Laurence F. Abbott. *Theoretical neuroscience*. Vol. 806. Cambridge, MA: MIT Press, 2001. (Visited on 11/12/2015).

[10] M. Deschênes, A. Labelle, & P. Landry. "A comparative study of ventrolateral and recurrent excitatory postsynaptic potentials in large pyramidal tract cells in the cat." In: *Brain research* 160.1 (1979), pp. 37–46. (Visited on 12/24/2015).

[11] *Documentation NEURON*. URL: `https://www.neuron.yale.edu/neuron/docs` (visited on 03/17/2016).

[12] Espen Hagen et al. "ViSAPy: A Python tool for biophysics-based generation of virtual spiking activity for evaluation of spike-sorting algorithms." In: *Journal of neuroscience methods* 245 (2015), pp. 182–204. (Visited on 03/02/2016).

[13] Matti Hämäläinen et al. "Magnetoencephalography- theory, instrumentation, and applications to noninvasive studies of the working human brain." In: *Reviews of modern Physics* 65.2 (1993), p. 413.

[14] *HOC Syntax — NEURON documentation*. URL: https://www.neuron.yale.edu/neuron/static/new_doc/programming/hocsyntax.html (visited on 03/17/2016).

[15] Alan L. Hodgkin & Andrew F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of physiology* 117.4 (1952), pp. 500–544.

[16] Satoshi Katai et al. "Classification of extracellularly recorded neurons by their discharge patterns and their correlates with intracellularly identified neuronal types in the frontal cortex of behaving monkeys." en. In: *European Journal of Neuroscience* 31.7 (2010), pp. 1322–1338. ISSN: 0953816X, 14609568. (Visited on 11/12/2015).

[17] Henrik Lindén et al. "LFPy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons." In: *Frontiers in neuroinformatics* 7 (2013).

[18] Zachary F. Mainen & Terrence J. Sejnowski. "Influence of dendritic structure on firing pattern in model neocortical neurons." In: *Nature* 382.6589 (1996), pp. 363–366. (Visited on 11/13/2015).

[19] Henry Markram et al. "Reconstruction and simulation of neocortical microcircuitry." In: *Cell* 163.2 (2015), pp. 456–492.

[20] Vernon B. Mountcastle et al. "Cortical neuronal mechanisms in flutter-vibration studied in unanesthetized monkeys: Neuronal periodicity and frequency discrimination." In: *Journal of neurophysiology* (1969).

[21] Paul L. Nunez & Ramesh Srinivasan. "Electric fields of the brain: the neurophysics of EEG." In: (2006).

[22] Klas H. Pettersen & Gaute T. Einevoll. "Amplitude variability and extracellular low-pass filtering of neuronal spikes." In: *Biophysical journal* 94.3 (2008), pp. 784–802.

[23] Wilfrid Rall & Gordon M. Shepherd. "Theoretical reconstruction of field potentials and dendrodendritic synaptic interactions in olfactory bulb." In: *J. Neurophysiol* 31.6 (1968), pp. 884–915. (Visited on 01/08/2016).

[24] David Sterratt et al. *Principles of computational modelling in neuroscience*. 2011.

[25] G. Vigneswaran, A. Kraskov, & R. N. Lemon. "Large Identified Pyramidal Cells in Macaque Motor and Premotor Cortex Exhibit "Thin Spikes": Implications for Cell Type Classification." en. In: *Journal of Neuroscience* 31.40 (2011), pp. 14235–14242. ISSN: 0270-6474, 1529-2401. (Visited on 11/12/2015).