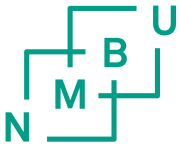


Norwegian University of Life Sciences
Faculty of Environmental Science and Technology
Department of Mathematical Sciences and Technology

Master Thesis 2015
60 credits

Neuron model reduction: From spatially extended biophysical models to simplified point models

Torgunn Halvorsen

1 Preface

This master thesis represents the final stage in my master's degree in Mathematical, Physical and Computational Sciences at the Norwegian University of Life Science (NMBU) at the Department of Mathematical Sciences and Technology (IMT).

I would like to thank my supervisors, Geir Halnes, Gaute Einevoll and Espen Hagen for useful comments, great ideas and a lot of patience throughout the process of this master thesis.

Ås, 12.03.15

Torgunn Halvorsen

Contents

2 Abstract

The response of a real neuron to a single synaptic input will depend on where on its dendritic tree the synapse is placed. The synaptic response is a voltage deflection generated at the synapse location in a synapse and then propagates from the synapse to the soma, where the membrane potential is typically measured. If the voltage deflection in the soma reaches a certain threshold value the neuron may generate an action potential, which is an output signal to other neurons. The distance from where the synaptic response is generated and propagates to the soma, decides the amount of weakening and change in shape the signal experienced before measured in soma. Real neurons will therefore have a diversity of somatic responses to a single synaptic input, depending on the synaptic location.

In network simulations the use of single compartment models (point models), are customary [1]. A point model lacks the spatial aspect of a morphological spacious neuron model. A point model has only one option for synapse placement, directly at its soma (i.e., its single compartment). The diversity of synapse responses a spacious neuron model can achieve in its soma is not captured by the point model in a trivial way.

The goal of this project is to create a synapse model for a passive point neuron, that can account for the diversity of somatic responses obtained in spatially explicit neurons.

"Realistic" somatic responses were obtained by placing synapses at different locations in a previously published multicompartmental neural model which was taken to represent a "real" neuron. The synaptic response obtained for a given synapse location was then recreated in a point model. This was done by using curve fitting methods: Four parameters in the point neuron synapse were varied to obtain an optimal fit from the response in the "real" neuron. This was repeated for all possible synapse locations in the realistic neuron's morphology.

The result was two synapse models for a point neuron. One easily implemented and applied, and one more accurate, but containing more results. The synapse was described by four parameters, each of which were functions of a single variable d, representing the synaptic distance from the soma in the realistic neural model. In this way, the variability in somatic response that in real neurons follow from variability in synaptic position, could be captured by varying a single parameter in a synapse placed at a point model. These synapse models will contribute to more realistic modelling of sub-threshold dynamics for point neurons.

3 Sammendrag

Et realistisk nevrons respons til et synaptisk innput vil avhenge av hvor på dens dendrittre synapsen er plassert. Synapseresponsen er en potensialendring som genereres i synapsen og deretter propagerer til soma, hvor membranpotesialet vanligvis måles. Hvis membranpotensialet i soma oversiger en gitt terskelspenning, kan nevronet generere et aksjonspotensiale, som er nevronets måte å sende signaler til andre nevroner. Avstanden fra synapseresponsen genereres og propagerer til soma, bestemmer graden av svekking og endring i form signalet opplever før det måles i soma. Realistiske nevroner vil derfor ha et mangfold av responser målt i soma for et synaptisk input, avhengig av synapsens plassering.

I nettverks-simuleringer er det vanlig å bruke én-compartment-modeller (punktmodeller) [1]. En punktmodell mangler det romlige aspektet til en morfologisk utstrakt nevronmodell. En punktmodell har kun én mulighet for synapseplassering, direkte på dens soma (dens eneste compartment). Mangfoldet av somaresponser en romlig kompleks nevronmodell kan måle i soma er ikke en del av punktmodellen.

Målet for denne oppgaven er å lage en synapsemodell for en passiv punktmodell, som gjør punktmodellen egnet til å modellere somaresponser som en romlig kompleks nevronmodell.

"Realistiske" somaresponser ble funnet ved å plassere en synapse på forskjellige steder i en tidligere publisert fler-compartment nevronmodell som ble valgt til å representere et "realistisk" nevron. Synapseresponsen for en gitt synapseplassering ble deretter gjenskapt i en punktmodell. Dette ble gjort ved kurvetilpasning: Fire parametre i punktnevronets synapse ble variert for å finne den optimale tilpasningen til responsen fra det "realistiske" nevronet. Dette ble gjentatt for alle synapseplasseringer mulig på det realistiske nevronets morfologi.

Resultatet var to synapsemodeller for et punktnevron. Én som var lett å implementere og anvende og én mer nøyaktig, men med flere resultater som følge. Synapsen var beskrevet av fire synapseparametre, som hver var funksjoner av en variabel d, som representerte den synaptiske avsanden fra soma i den realistiske modellen. På denne måten ble mangfoldet av somareponser fra et "realistisk" nevron gjenskapt ved å variere parameteren d i en synapse plassert på en punktmodell. Disse synapsemodellene vil bidra til mer realistisk modellering av dynamikken til et punktnevron under terskelspenning.

4 Introduction

A realistic neuronal morphology has a complex spacious structure consisting of a cell core, the soma, with branches called dendrites growing out of it. The input-terminals of a neuron, the synapses are located at the dendrites, while the output-terminals lie on a branch growing out of soma called the axon.

When a synapse receives a signal from another neuron, the synapse responds by an increase or decrease in the membrane potential, called a synaptic response. The synaptic response is generated in the synapse and then propagates along the dendrites to the soma, where it is typically measured. If the membrane potential in soma reaches a threshold value, a rapid increase in the membrane potential, called an action potential (AP) is sent to the synapses at the axon communicating with other neurons.

Neuron models have different degrees of complexity, depending on what one wishes to model. If it is the concrete shape and values for a synaptic response one wishes to capture in a model, a complex model can be a good choice. A model can be complex in both its membrane mechanisms and morphology. Complexity in membrane mechanisms can be a model containing several active ion channels, while a complex morphology is a complex model of the neurons structure.

In a complex morphology model the dendrites and soma are modelled as cylinders that are divided into several compartments where the membrane potential is evaluated at every compartment when a signal propagates from a synapse at the dendrites to the soma.

When simulating large networks of neurons, one wishes to recreate the trends in signals and not so much the concrete values. The models of neurons' structure in networks are often simplified. Usually every neuron in a network simulation is represented by a one-compartment model, called a point model, to keep the model's conceptual complexity and computational cost low. [1].

A point model lacks the spatial aspect of a real neuron. The structure of the neuron is simplified to a point, which represents dendrites, axon and soma.

The only option to place a synapse on a point model is at its only compartment, as opposed to a complex morphology model, where synapses can be placed at arbitrary locations in the dendrites at various distances from the soma.

A spatial morphology can produce a variety of responses measured in soma, by changing the location of the synapse. The spread and weakening of the signal as it propagates along the dendrites to soma from different locations will be a dynamical aspect of a real neuron.

The point model is a different story. The only synaptic response possible to obtain is found by placing the synapse at its only compartment, the soma.

How can a point model capture the diversity in signals a complex morphology model can produce? My goal is to create a synapse model for a point neuron, able to capture the variety of excitatory post synaptic responses [EPSPs], an active, complex morphology model can produce.

I used a model by Hay et. al (the Hay-model) [2] to be my realistic, active and complex structured model. I measured all the possible EPSPs the Hay-model could produce in its soma, by running a simulation with a conductance based, two-exponential synapse for each synapse placement possible on its morphology. The somatic responses obtained from the Hay-model were used as my "realistic" data.

An investigation with many similarities was performed by Wybo et. al in 2013 [1]. They also used a synapses to reinstate the spatial aspect for a point model, but the model they simplified was passive, as opposed to the active Hay-model, and the synaptic integration was therefore a more analytical matter.

The synapse I used on the point model was current based. This made it possible to derive an analytical solution for its synaptic response.

The analytical solution for the point model's synaptic response contained four synapse parameters, describing the shape of the EPSP. The four synapse parameters were: A, t_0, τ_1 and τ_2; peak value, delay, decay time constant and rise time constant, respectively.

By letting the four synapse parameters in the expression for synaptic response for the point model vary and fitting it to the different EPSPs measured for the Hay-model, I obtained values for the point model's synapse parameters, A, t_0, τ_1 and τ_2, for each EPSP the Hay-model produced.

To be able to present my results in an easily implemented and applied way, I wanted to find equations describing the synapse parameters fitted for the point model. I was interested in finding a trend in these four synapse parameters, dependent on the somatic distance the EPSP from the Hay-model had travelled (distance from soma).

I separated the four types of fitted synapse parameters for the point model and indexed each value with the corresponding distance from soma.

I captured the trend in the fitted synapse parameters through curve fitting with the sigmoid function, by letting parameters describing the shape of the sigmoid function vary.

I obtained four sigmoid functions, dependent on a single variable d. The variable d represents the somatic distance for the EPSPs from the Hay-model. When the four sigmoid functions are evaluated for a given somatic distance, they give the four synapse parameters (A, t_0,τ_1 and τ_2) to build a synapse for the point model, able to recreate the EPSP from the Hay-model, with the synapse actually placed at the given distance from soma.

My resulting synapse model can be put on point models and make them model sub-threshold dynamics and capture dendritic integration in a more realistic way.

My synapse model is mainly meant for network simulations where the use of point models is common [1].

This thesis has the following structure: I will first look at theory background where neuronal structure and mechanisms and well known ways to model neurons, and synapses are elaborated. Then I look at methods used in the project, including the Hay-model implemented in Python and the methods for the fitting processes for finding the values of the synapse parameters for the point model, and the curve fitting with the sigmoid function to capture the trends in the fitted parameter to obtain my result functions. Then follows the Result chapter, where I look at the derivation of the equation for the point model's response to a synaptic input, how I decided the values for a cell parameter and the results for the fitting processes, with values and error estimates. I will discuss some of my results in the result section. Then I summarize my results in the conclusion. Then I discuss my project's limitations and future prospects among other topics worthy of being discussed, followed by Appendix A, containing some sidelined results from the curve fitting of sigmoid functions and Appendix B containing Python codes used in the project.

5 Theory background

The human brain is an extremely complex organ, with responsibilities from coordinating physical movement of our body to vital unconscious processes as breathing and digesting food. Our brain's uniqueness gives us our personality and abilities such as language, moral judgements, rational thoughts and memories [3]. The brain consists of about 100 billion [3] nerve cells and some trillion synapses [4].

The nerve cells in the brain are called neurons and they have rich external anatomy. The anatomy consists of a cell core, called the soma. Branches, called dendrites grow out of the soma. The input terminals of the neuron are called synapses, and are where the neurons receive signals from other cells. The synapses are located at the dendrites. The neuron has one output terminal, called the axon, where the neuron sends signals to other neurons. [4]. Figure 1 shows the schematic structure of a neuron.

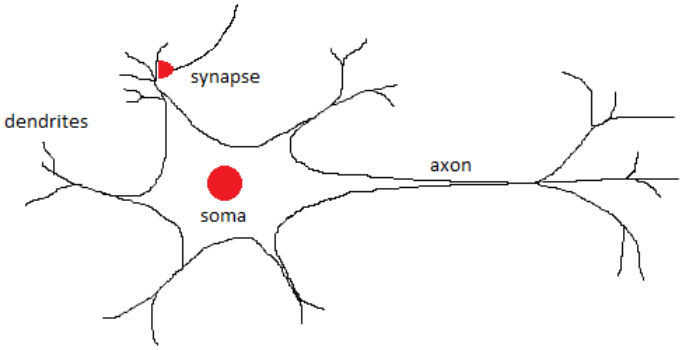


Figure 1. A schematic neuron. The soma is the cell body. Dendrites with their branches grow out of soma and have synapses which receive signals from other neurons. The axon transfers signals from soma and sends signals to other neurons.

A neuron receives signals from other neurons in their synapses located at the dendrites. A synapse input leads to a synaptic response that either increases or decreases the membrane potential of the cell, dependent of the type of neurotransmitter the synapse received. Different neurotransmitters can either lead to an excitatory or inhibitory response, increasing or decreasing the membrane potential respectively.

A neuron has a membrane resting potential which lies around -65 mV [5]. Sufficiently strong synapse inputs can elevate the membrane potential to above the threshold value and trig mechanisms in the ionic channels, which initiate an action potential (AP). The AP is the neuron's mechanism for sending signals to other neurons. It is characterized by a steep increase of the membrane potential, about 100 mV above the threshold value, followed by a less steep decrease to below the neuron's resting potential, before stabilizing.

The main goal for a neuron model is to recreate the behavior of the real neurons. Typical one uses recordings from real neurons and want the model to capture the most important characteristics in the data. What the important characteristics are depends on the level of detail on which one chooses to simulate, and what one considers to be important. In complex one-cell models one typically wishes to recreate the neuron's potential response, how the membrane potential, V(t), varies with time , with great detail. In simplified network models one typical settles to recreate some characteristics of the data. Most common is recreating the average firing rate of action potentials (APs), without thinking about the shape of the APs or recreating the signal V(t) between the them.

Models differ in complexity with morphology and membrane-mechanisms. The most complex morphology models consist of spatial 3D-reconstructions of the real neuron's morphologies. The structure of the neuron is split up into a number of parts, called compartments. The number of compartments decides

the spatial resoluteness of the model. The simplest morphology model consists of one single compartment and is called a point model.

A model with only one compartment gives the neuron infinitely large inner conductance which makes the whole neuron lie on the same potential ground and the information about how potential can vary throughout the dendrite-tree and axon is not captured by the model.

For a model with several compartments, the dynamics between the compartments is decided by the cable equation, see section 5.4. [5]

To model the neurons one uses different neuro-simulators. I used NEURON and Python, described in section 5.7.

5.1 The RC-circuit

A simple model for a membrane and its potential is an RC-circuit known from electronics [5], shown in figure 2. An RC-circuit describes a passive one-compartment model's membrane potential and is suitable to describe a passive point model's sub-threshold dynamics. The current source, I_e models an electrode

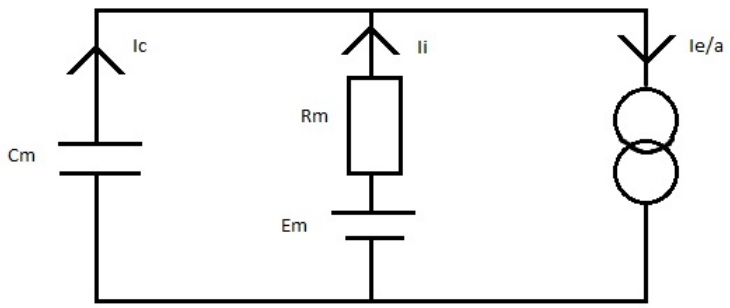


Figure 2. The RC-circuit consists of a capacitor, C_m, that models the aspect that a membrane can store and separate charge. The resistor, R_m describes the conductance of the passive ion channel, where lower resistance gives higher conductance. The battery, E_m models the net difference between ion density inside and outside the cell and sets up the resting potential. The current source, I_e represents an injected current. I_i is the current flowing through the ionic channel, in and out of the cell. I_c is the capacitive current due to the change in membrane potential [6].

where current can be injected. Injected current is divided by area of the cell, because this model has infinite inner conductance and the current will be homogeneously divided in the cell. [5]

$$I_{\frac{e}{a}} = I_c + I_i \tag{1}$$

Injected current $I_{\frac{e}{a}}$ gives rise to both a capacitive current I_c and an ionic current I_i, described in equations 2 and 3 respectively.

$$I_c = Cm\frac{dV}{dt} \tag{2}$$

$$I_i = \frac{V - Em}{Rm} \tag{3}$$

By substituting equation 2 and 3 into equation 1 we get the equation for an RC-circut in equation 4.

$$Cm\frac{dV}{dt} = \frac{Em - V}{Rm} + \frac{I_e}{a} \tag{4}$$

Equation 4 describes the change in an isolated neuron's membrane potential when a current I_e is injected. This neuron model is isolated, because it does not receive signals from other neurons. Equation 4 is suitable for modelling sub-threshold membrane potential changes for a passive point model. [5]

5.2 The Hodgkin-Huxley model of action potential

The Hodgkin-Huxley (HH-model) model is a one-compartment model, like the RC-circuit and describes a single compartment's membrane potential

The HH-model includes three types of ionic channels; sodium- (Na^+), potassium- (K^+), and leak channel. The sodium and potassium channels' conductances are voltage dependent, so the conductances' change is dependent on the membrane potential of the neuron, and the model is active.

The HH-model also includes some state variables for the ionic channels, and together with its active properties it does not only make the modelling of action potentials possible, but also explains the mechanisms behind it. The HH-model was the first to describe the active ion channel mechanisms quantitatively.

The way of modelling ionic flow through the membrane with the state variables described in this model is often referred to as HH-formalism and is widely applied on more complex morphology models. Then the HH-formalism is used on every compartment of the cell's morphology, as in the Hay-model [2].

Hodgkin and Huxley simulated the action potential in the squid giant axons, because these are big end easy to work with. Their work proceeded in three main stages: They recorded intracellularily in the squid giant axon with a spacial clamp and found the current-voltage connection. Then they changed the extracellular sodium concentration and found the amount of current carried by the Na^+ and other ions such as K^+. They then fitted these results to a mathematical model containing a circuit and two active ion-channels (Na^+ and K^+). They then described the membrane potential in different situations and found a numerical solution. What the model predicted matched the recorded values.

To decide whether an ionic channel is in open or closed state, Hodgkin and Huxley introduced gating particles, see figure 3. [5]

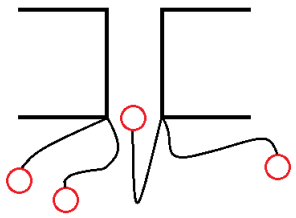


Figure 3. Abstraction of the gating particles. The black boxes describe a membrane with an ion channel. For the ionic channel to be in open state, neither of the gating particles (red balls) can block the channel. Here one gating particle is blocking the channel and the channel is in closed state.

5.2.1 The gating particles

Gating particles were meant to decide whether an ionic channel was in open or closed state. A gating particle has the probability n for not blocking the ionic channel. With x number of gating particles, the channel has a probability of n^x for being in open state, because none of the gating particles can block the channel for it to be in open state, as shown in figure 3.

The movement of the gating particles between the two states, open and closed, is described as a reversible chemical reaction in equation 5.

$$closed \ \overrightarrow{\alpha_n(V)} \ \overleftarrow{\beta_n(V)} \ open \tag{5}$$

α_n and β_n are rate coefficients depending on the membrane potential. The fraction of the gating particles being in open state is n, and the fraction 1-n are in closed state. The change in n per time is described

by equation 6.

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n \tag{6}$$

$\frac{dn}{dt}$ is the change in the fraction of gating particles being in open state per time. α_n is the rate for gating particles being in closed state changing to open state. β_n is the rate of gating particles being in open state changing to closed state. In other words the change in n per time is the rate particles change from closed to open state times the particles in closed state minus the rate particles that change from open to closed state times the particles in open state. [5]

5.2.2 The potassium channel

The dynamics for the active potassium channel is described in equation 7.

$$I_k = g_k n^4 (V(t) - E_k) \tag{7}$$

With g_k given by equation 8.

$$g_k = \overline{g_k} n^4 \tag{8}$$

I_k is the current through the potassium channel. g_k is the potassium channel's conductance. $\overline{g_k}$ is a constant found experimentally. n is the state variable describing the probability for a potassium gating particle for being in open state. n^4 gives the probability for the potassium ionic channel being in open state, with its four gating particles. V(t) is the membrane potential and E_k is the resting potential for the potassium channel and its electromotive force.

The dynamics for state variable n is described by equation 9

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n \cdot n \tag{9}$$

The rate coefficients α_n, describing the rate potassium gating particles move from closed to open state and β_n, describing the rate potassium gating particles move from open to closed state, follow the equations 10 and 11 respectively. [5]

$$\alpha_n = 0.01 \frac{V+55}{1 - e^{-\frac{v(t)+55}{10}}} \tag{10}$$

$$\beta_n = 0.123 e^{-\frac{V(t)+65}{80}} \tag{11}$$

5.2.3 The sodium channel

The sodium channel has two state variables, m and h. m is probability for a sodium gating particle for being in open state, equivalent to the potassium channel's state variable n. h is an inactivation state variable. The dynamics for the active sodium channel is described in equation 12.

$$I_{Na} = g_{Na}(V(t) - E_{Na}) \tag{12}$$

With g_{Na} given by equation 13.

$$g_{Na} = \overline{g_{Na}} m^3 h \tag{13}$$

I_{Na} is the current through the sodium channel. g_{Na} is the sodium conductance. $\overline{g_{Na}}$ is a constant found experimentally. m is the state variable for the fraction of sodium gating particles being in open state. m^3 gives the probability of the sodium ionic channel being in open state, with its three gating particles. V(t) is the membrane potential and E_{Na} is the resting potential for the sodium channel and its electromotive

force.

The dynamics for the state variable m is described in equation 14.

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m \cdot m \tag{14}$$

The rate coefficients α_m, describing the rate sodium gating particles move from closed to open state and β_m, describing the rate sodium gating particles move from open to closed state, follow the equations 15 and 16 respectively.

$$\alpha_m = 0.1 \cdot \frac{V+40}{1-e^{-\frac{V-40}{10}}} \tag{15}$$

$$\beta_m = 4 \cdot e^{-\frac{v+65}{18}} \tag{16}$$

The state variable h describes the level of inactivation for the sodium ionic channel. The dynamics of the inactivation state variable h is described in equation 17.

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h \cdot h \tag{17}$$

The rate coefficients α_h, describing the rate h move from inactivating to activating the sodium channel, and β_h, describing the rate h move from activating to inactivating the sodium channel, follow equations 18 and 19 respectively. [5]

$$\alpha_h = 0.07 \cdot e^{-\frac{V+65}{20}} \tag{18}$$

$$\beta_h = \frac{1}{e^{-\frac{V+35}{10}}+1} \tag{19}$$

5.2.4 The leak channel

Besides the active potassium and sodium channels, there are a lot of other ionic currents flowing through a realistic neuron's membrane. The extra flow of ions through the membrane is mostly caused by chloride ions but also calcium ions and other potassium and sodium ions. These currents contribute to the cell's negative resting potential. All of these currents are gathered in one ionic channel in the Hodgkin-Huxley model, called the leak ionic channel. The leak ionic channel follows the quasi- ohmic current-voltage relationship described in equation 20.

$$I_L = \overline{g_L}(V(t) - E_L) \tag{20}$$

I_L is the current through the cell membrane due to the other ions besides Na^+ and K^+. $\overline{g_L}$ is the leak channel's conductance, a constant found experimentally. It differs from the sodium and potassium conductances by not depending on the cell's membrane potential and is a passive element in the model. V(t) is the membrane potential and E_L is the leak channel's resting potential and electromotive force. [5]

5.2.5 The total dynamics

The complete Hodgkin-Huxley model is described by equation 21.

$$C_m \frac{dV}{dt} = -\overline{g_L}(V(t) - E_L) - \overline{g_{Na}} m^3 h (V(t) - E_{Na}) - \overline{g_K} n^4 (V(t) - E_K) + I \tag{21}$$

C_m is the membranes conductance, V(t) is the membrane potential at time t. I is the local circuit current; the net contribution of axial current from neighbour compartment. I can be modelled by the cable equation, see equation 25. The other terms are specified in the equations 20, 12 and 7 respectively.

The Hodgkin-Huxley model describe the mechanisms behind the action potential; When the membrane is depolarized to above its threshold value the sodium current is activated, and the state variable m and the sodium conductance increase. Because of the sodium channel's high reversal potential, both the sodium conductance and membrane potential continue to increase. Potassium activates after sodium and cause a re-polarization of the membrane, because positive ions are flowing out of the cell due to its low reversal potential. The in-activation variable h, for sodium also contributes to the re-polarization, because it decreases when the membrane potential increases. Sodium deactivates and the cell is under resting potential. The active mechanisms are no longer present and the cell re-establish its resting potential. [5]

5.3 Multicompartmental model

Differing from the RC-neuron and the HH-model, which are one-compartment models, where the cell is assumed to be isopotential, a multicompartmenal model splits the cell's morphology into cylindrical parts of length l and diameter d. Each compartment is isopotential, but different compartments can have different membrane potentials. Current can flow through the cell's membrane and longitudinally (inside the cell) in each compartment as shown in figure 4. How the membrane potential changes from one compartment to the next is modelled by the fundamental equation for a compartmental model, in equation 22. [5]

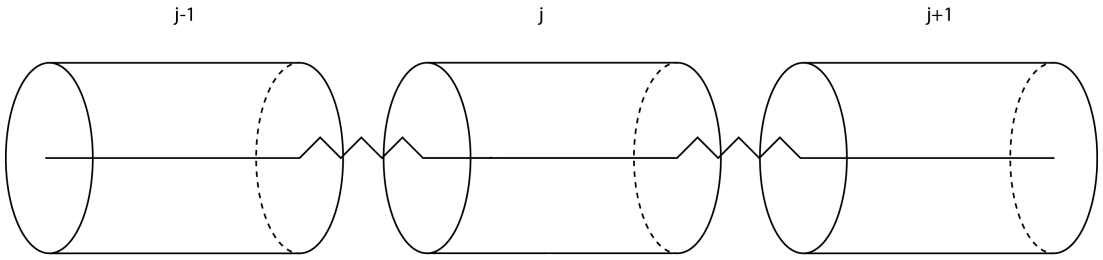


Figure 4. The idea behind a multicompartmental model. The neuron's morphology is split into cylinders of length l and diameter d, called compartments. Each compartment is isopotential, but different compartments can have different membrane potentials. Current can flow through the membrane and longitudinally in each of the compartments.

$$C_m \frac{dV_j}{dt} = \frac{E_m - V_j}{R_m} + \frac{d}{4 \cdot R_a}\left(\frac{V_{j+1} - V_j}{l^2} + \frac{V_{j-1} - V_j}{l^2}\right) + \frac{I_{e,j}}{\pi d \cdot l} \tag{22}$$

C_m is the cell's membrane capacitance. V_j is the membrane potential for compartment with index j, V_{j+1} is the membrane potential for compartment with index j+1. V_{j-1} is the membrane potential for compartment with index j-1. E_m is the cell's reversal potential. R_m is the cell's membrane resistance. R_a is the cell's axial resistance. d is the diameter of each compartment. l is the length of each compartment. $I_{e,j}$ is the current injected into compartment with index j. [5]

5.4 The cable equation

The cable equation is a partial differential equation which gives an analytical solution to the multicompartmental model in section 5.3. The idea is to split the neurite into infinitely many infinitesimally small compartments. The membrane is modelled as a cylinder as shown in figure 5.

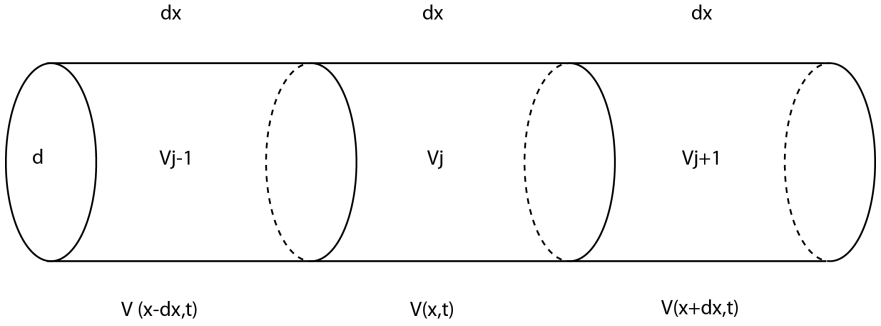


Figure 5. The idea behind the cable equation is to split a neurite into infinitely many infinitesimally small compartments of length δx and diameter d

$$C_m \frac{\partial V(x,t)}{\partial t} = \frac{E_m - V(x,t)}{R_m} + \frac{d}{4R_a} \cdot [\frac{1}{\delta x}(\frac{V(x+\delta x,t) - V(x,t)}{\delta x} - \frac{V(x,t) - V(x-\delta x,t)}{\delta x})] + \frac{I_e(x,t)}{\pi d} \quad (23)$$

$$\frac{1}{\delta x}(\frac{V(x+\delta x,t) - V(x,t)}{\delta x} - \frac{V(x,t) - V(x-\delta x,t)}{\delta x}) = \frac{\partial^2 V(x,t)}{\partial x^2} \quad (24)$$

when $\delta x \to 0$

$$C_m \frac{\partial V(x,t)}{\partial t} = \frac{E_m - V(x,t)}{R_m} + \frac{d}{4R_a} \frac{\partial^2 V(x,t)}{\partial x^2} + \frac{I_e(x,t)}{\pi d} \quad (25)$$

The membrane potential V(x,t) is a function of x; position along the cable and time; t . E_m is the membrane resting potential and electromotive force. R_m is the membrane resistance. $I_e(x,t)$ is the injected current per unit length at position x at time t. The term $\frac{E_m - V(t,x)}{R_m}$ describes that the neuron works to keep itself at equilibrium. The second term at the right side of the equation describes the local circuit current; the net contribution of axial current from neighbour compartments. [5]

5.5 Synapses

Synapses are the neuron's input-terminals and are where the communication with other neurons happens. Synapses can be chemical or electrical. An electrical synapse is a direct electrical contact between cells through channels which span the membrane of both cells. A chemical synapse has a pre- and post-synaptic terminal separated by a synaptic cleft, see figure 6.

The mechanisms of synaptic transmission in a chemical synapse are well established. An action potential at the pre-synaptic side of the synapse depolarizes the synaptic terminal which this makes the calcium-ion channels of the synaptic terminal open and causes a flow of calcium-ions through the membrane. The calcium ion flow leads to a release of neurotransmitters into the synaptic cleft. The neurotransmitters diffuse to the post-synaptic side of the synapse and are temporarily bound to post-synaptic receptors. This opens the ion channels, allowing ions to flow in and out of the cell, initiating a synaptic response. [7]. The synaptic response is either excitatory or inhibitory, increasing or decreasing the membrane potential respectively, dependent on the type of neurotransmitter the respective synapse uses [5].

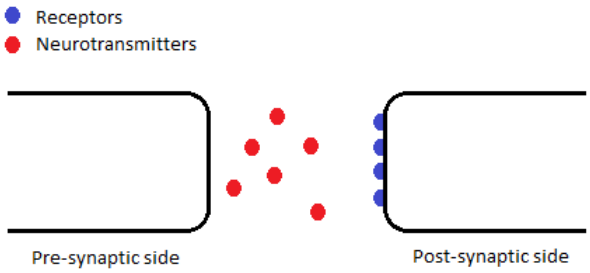


Figure 6. An abstraction of a chemical synapse. It consists of a pre- and a post-synaptic side separated by a synaptic cleft where neurotransmitters (red dots) diffuse from the pre-synaptic side to receptors (blue dots) at the post-synaptic side.

There are two types of models for chemical synapses: current based synapses and conductance based synapses. For current based synapses the synapses current response is described by equation 26. $I_{syn}(t)$ is the synaptic current due to to a synaptic input. $\overline{I_{syn}}$ is the maximum current. $f(t)$ is a function of time, describing the time course of the synaptic current.

$$I_{syn}(t) = \overline{I_{syn}} \cdot f(t) \tag{26}$$

For conductance based synapses the synapse current is described in equation 27.

$$I_{syn}(t) = \overline{g_{syn}} \cdot f(t) \cdot (V(t) - E_{syn}) \tag{27}$$

$I_{syn}(t)$ is the synaptic current due to to a synaptic input. $\overline{g_{syn}(t)}$ is the maximum synaptic conductance. $f(t)$ is the time course for the synaptic current. E_{syn} is the synapse's resting potential. The conductance based synapse is dependent on the membrane potential because of the factor $V(t) - E_{syn}$.

There are several different ways to model the time course, f(t), for the synaptic conductance for the conductance based synapses. The time course, $f(t)$ is often modelled using simple waveforms. The three most commonly used waveforms are; single exponential decay, described in equation 28, the alpha

function, described in equation 29 and the beta function, also called two exponential function, described in equation 30. [7].

$$f(t) = \overline{g_{syn}} \cdot e^{\left(-\frac{t-t_0}{\tau}\right)} \tag{28}$$

The single exponential waveform, see figure 7 (a) and equation 28.
The equation is valid for t$\geq t_0$. At t=t_0 the conductance goes straight up to its amplitude: $\overline{g_{syn}}$. τ describes the rate of decay [7]. At t=τ the signal has dropped to $\sim$ 63 % of its starting value and continues to drop by the factor e^{-1} when t goes an interval of τ. At t=3τ the signal has dropped 95 %. [8]

$$f(t) = \overline{g_{syn}} \frac{t - t_0}{\tau} \cdot e^{1-\left(\frac{t-t_0}{\tau}\right)} \tag{29}$$

The alpha function, see figure 7 (b) and equation 29. The alpha function describes the rising process better than the single exponential decay using the coefficient $\frac{t-t_0}{\tau}$. At t=t_0 the signal rises to its amplitude $\overline{g_{syn}}$ at time $t = \tau + t_0$ and then drops by a factor e^{-1} when t goes an interval of τ [7]. One often see the alpha function without the term 1 in the exponent [5], this term normalizes the equation and assures $\overline{g_{syn}}$ to be the amplitude.

$$f(t) = \overline{g_{syn}} \frac{\tau_1 \cdot \tau_2}{\tau_1 - \tau_2} \left(e^{\left(-\frac{t-t_0}{\tau_1}\right)} - e^{\left(-\frac{t-t_0}{\tau_2}\right)} \right) \tag{30}$$

The beta function or the two exponential function, see figure 7 (c) and equation 30, is the most complex waveform of the three I look at. The beta function describes both the decay and rise with separate constants, τ_1 and τ_2 respectively. [5]
It is easier to see the physics behind the normalized beta function, see equation 31, with the normalization coefficient C in equation 33 and t_{peak} in equation 32.

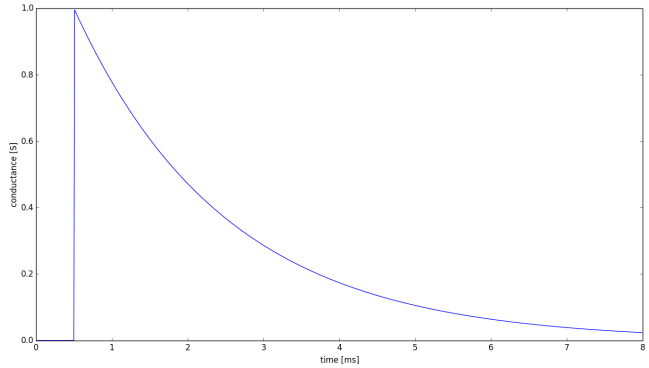
$$f(t) = \overline{g_{syn}} \cdot C \left(e^{\left(-\frac{t-t_0}{\tau_1}\right)} - e^{\left(-\frac{t-t_0}{\tau_2}\right)} \right) \tag{31}$$

$$t_{peak} = t_0 + \frac{\tau_1 \cdot \tau_2}{\tau_1 - \tau_2} \cdot ln\left(\frac{\tau_1}{\tau_2}\right) \tag{32}$$

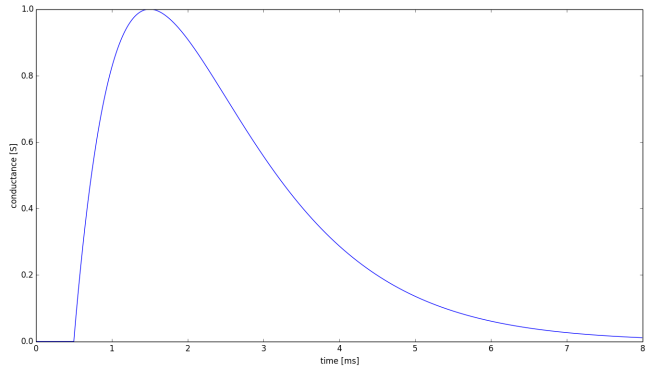
$$C = \frac{1}{-e^{-\left(\frac{t_{peak}-to}{\tau_2}\right)} + e^{-\left(\frac{t_{peak}-to}{\tau_1}\right)}} \tag{33}$$

Equation 31 peaks at $t = t_{peak}$ and has an amplitude equal to $\overline{g_{syn}}$ [7].

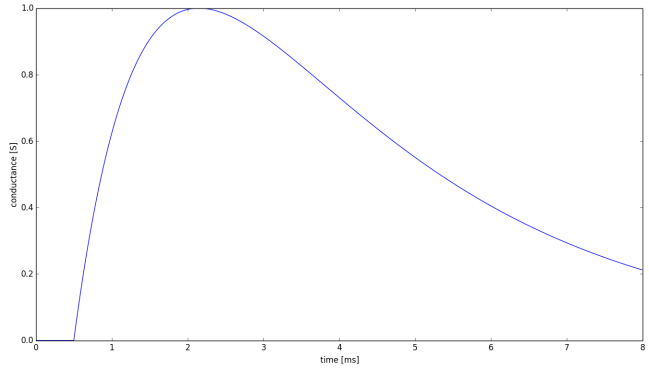
The time course for the synaptic current for current based synapses, are the same waveforms as the conductance based synapses use for their conductance.



(a) single exponential decay



(b) alpha function



(c) betha function

Figure 7. The waveforms used to model the synapse's conductance. All graphs are normalized and has a 0.5 ms delay. For figure (a) $\tau = 2$ ms, (b) $\tau = 1$ ms , (c) $\tau_1 = 3$ ms and $\tau_2 = 1$ ms

5.6 The Hay-model

The model used in this study is the Hay-model [2]. This is a model for the layer 5b pyramidal cells (L5b PCs).

The L5b PCs have been subjects of many experimental and modelling studies, because they are major building blocks in the mammalian neocortex and extend their dendritic trees to all its six layers.

The Hay-model is the first model that reproduces perisomatic Na^+-spiking behavior and active dendritic properties such as Ca^{2+} spikes and the interaction between these two spiking regions, back propagating action potentials and experimental variability.

The model was based on recordings in adult rats. Using "an evolutionary algorithm" 21 free parameters were fitted, among them channel densities for nine ionic channels. The channels follow the Hodgkin-Huxley formalism (as described in the subsection 5.2) and the ionic currents are modelled on the form shown in equation 34.

$$I = \overline{g} m^x h^y (V(t) - E_{syn}) \tag{34}$$

I is the ionic current. E_{syn} is the ionic channel's reversal potential. V(t) is the membrane potential. x is the number of gating particles. y is the number of gating inactivation particles. m is the probability of a gating particle being in open state. h is the level of activation of an inactivation particle.

The model was made by looking at other models that optimized either of the two target behaviors (perisomatic Na^+-spiking and active dendritic Ca^{2+} spiking). Hay et. al looked at the parameters that differed between the models in range, size or range values, because these differences pointed in the direction that the given ionic channel had to change and was important to the target behavior. Statistics of electrophysiological features such as spike frequency, spike width and adaptation index were grouped into separate objectives and fitted to a model of a L5b PC by the "evolutionary algorithm".

The morphology was split into compartments up to 200 μ m long and the cells had an average of 200 compartments. The optimization and simulations were conducted in NEURON and runtime was between 2 and 5 days. [2]

The Hay-model implemented on the layer 5b pyramidal cell was used as my "realistic neuron". The Pyramidal cell has long apical dendrites at the top, that differs from the shorter proximal dendrites around soma at the bottom. When an action potential is fired at the soma it can propagate all the way to the top of the apical dendrites. The apical dendrites can fire action potential locally themselves, if they get enough synaptic input [9].

It has been suggested that L5b PCs could be important in the learning process. The apical and proximal dendrites get input from different parts of the brain, and can therefore serve as a detecting device for events happening at the same time, since the response in soma gets larger when the signals received in the apical- and proximal dendrites occur with short time between them [10].

Similar phenomena can occur locally at dendrites. For example synapse activation on more than one synapse happening at the same time can give a response higher than linear summation of single responses [11].

Recordings show that pyramidal neurons in awake animals are bombarded by synaptic input and are in "high conductance state". This bombarding of input leads to a depolarization of the membrane. Therefore it lies on a higher potential than the resting potential and more ion channels are in open state than for an isolated cell. Open ion channels means higher conductance (or lower resistance). In this state the membrane potential of the cell will differ ($\pm$4mV) and the response to a signal will be dependent on how the membrane potential was when the input arrived. [9]

5.7 NEURON and Python

NEURON is a simulation environment for modelling single neurons or networks.

Expertise in numerical methods or programming is not necessary when using NEURON, given its efficient tools for constructing, running and managing models.

NEURON uses the programming language hoc. A hoc-file specifies the morphology and biophysical features as ionic channels. It is also possible to implement models directly in the GUI (graphical user interface).

NEURON uses the cable equation to describe the membrane potential for each segment of a cell, and evaluates it at the mid-point of each segment. NEURON uses linear interpolation of the potential between the midpoints of two neighbor compartments and sets each compartment on the same potential ground [12].

For multicompartmental models the most common formalism used for modelling ion channels and membrane mechanisms is the Hodgkin-Huxley formalism (HH-formalism), see section 5.2. The HH-formalism is easily implemented in a NEURON model.

Python has a wide range of analysing tools built for scientists and engineers. Adding Python to NEURON makes it possible to create complex programs using Python's wide spread of built-in functions from fitting algorithms to statistical analyses [13]

I am going to use the open source Python package LFpy (Local Field Potentials in Python), which runs on top of NEURON. LFPy contains classes for defining cells, synapses and point processes as Python objects. [14]

6 Method

6.1 The Hay-model implemented in Python

The Hay-model was chosen as my realistic neuron. Therefore its EPSPs measured in soma were used as data in the same way others use recorded data from ex. rat brains.

The Hay-model was implemented in Python and NEURON by Espen Hagen, see section 11.1 in Appendix B for code. The Hay-model implemented in Python has 642 compartments and therefore 642 options for placing a synapse. The synapse gets an input at time t= 500 ms which generates an EPSP that travels from the synapse to the soma where the membrane potential is recorded.

The model's cell and synapse parameter values are listed in table 1, and figure 8 show its morphology.

For use in the following sections, 6.2 and 6.3 I split the Hay-models morphology into four parts; apical, stem, basal and branches, see figure 9.

Cell parameter	value	unit	Explanation
R_m	30000	Ωcm^2	membrane resistivity
C_m	1	S/cm^2	membrane capacitance
Ra	150	Ωcm	axial resistance
a	31200	μm^2	area of the cell
V_{init}	-75	mV	initial potential
Synapse parameter	value	unit	Explanation
e	0	mV	reversal potential of synapse
τ_1	0.25	ms	time constant rise
τ_2	1.0	ms	time constant decay
weight	0.01	μS	synaptic weight

Table 1. The cell and synapse parameters used for the Hay-model implemented in Python by Espen Hagen.

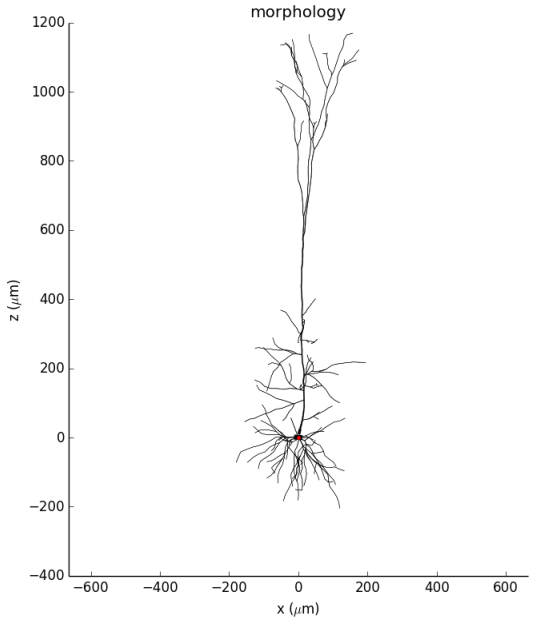


Figure 8. The morphology for the cell the Hay-model was implemented on. The red represents a synapse. Here it is placed in soma at index = 0.

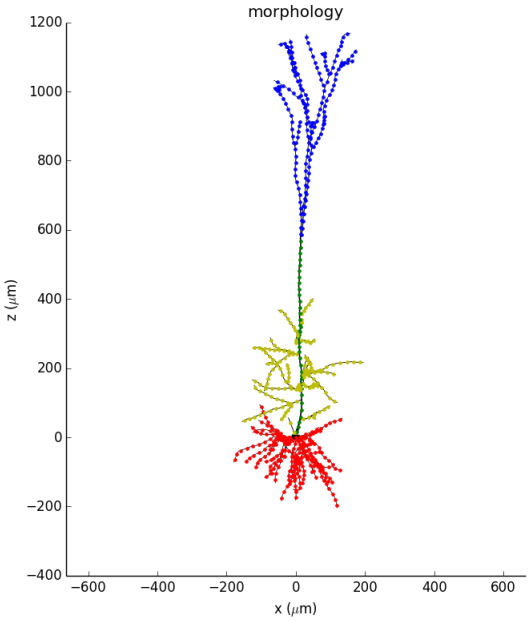


Figure 9. The morphology for the Hay-neuron split into four parts. The coloured dots represent synapse indexes. Red dots are in the basal region, yellow dots are forks growing out of the stem, called branches, the stem is green and the apical part is blue.

6.2 Fitting point model EPSPs to EPSPs obtained with the Hay-model

I derived an analytical solution for the point model's synapse response due to a synapse input, in section 7.1, equation 58. Equation 58 contains some synapse parameters deciding the strength and shape of the EPSP for the point model. They are: A, t_0, τ_1 and τ_2.

A is the amplitude [mV] and the maximum fluctuation of the EPSP above resting potential. t_0 is the delay [ms], which decides the starting time of the EPSP. τ_1 is the decay time constant [ms] which decides the rate the EPSP decays towards resting potential after it peaks and τ_2 is the rise time constant [ms] and decides the time rate for the rise of the signal towards the peak value.

I let these four synapse parameters vary in equation 58 and fitted it to the EPSPs obtained from the Hay-model. The results were sets of the four synapse parameters for the point model for each EPSP the Hay-model produced.

To link the fitted synapse parameters for the point model to the Hay-model, I calculated the distance each EPSP from the Hay-model travelled to soma and pinned this value to the respective set of fitted synapse parameters.

The fitting process was conducted in Python using the package scipy.optimize.minimize [16] with the fitting method TNC. TNC (Trunctated Newton method) [15] is a non-linear fitting algorithm that makes it possible to use bounds on the parameters [16]. For Python code, see section 11.2 in Appendix B. I chose to fit the synapse parameters for the point model in four different ways using different bounds on the parameters and length for the interval for the curve fitting:

1. Fit 1, The fitting process was done with bounds on the delay, it could range from 500 ms to the time where the Hay signal has reached 3% of its amplitude.
   The interval where the curves were fitted was from 500 ms to the end time of the simulation.

2. Fit 2, The fitting process was done with bounds on both delay and amplitude. The delay could range from 500 ms to the time where the hay signal has reached 3% of its amplitude and the amplitude was fixed at the Hay-model's EPSPs amplitude value.
   The interval where the curves were fitted was from 500 ms to the end time of the simulation.

3. Fit 3, The fitting process was done with equal bounds on the delay and amplitude as Fit 2.
   The interval where the signals were fitted where shortened. The signals were fitted from 500 ms to the time where the EPSP from the Hay-model reached its resting potential after peaking, to avoid fitting the part of the Hay-models EPSP going below the resting potential before stabilizing.

4. Fit 4, The fitted parameters for the point model were obtained by calculating the mean values of each of the fitted synapse parameters from Fit 1, 2 and 3.

The synapse parameters for the point model were fitted for all the EPSPs from the Hay-model by the four methods listed above.

I also chose to split the EPSPs from the Hay-model into four groups, dependent on which part of the Hay-models morphology they were generated (apical, stem, basal and branches). The four parts of the morphology for the Hay-model is showed in figure 9 on page 23. The idea was that EPSPs generated in the same part of the morphology would encounter about equal change in shape and wakening in strength, because they propagated about the same distance to soma were they were recorded.

In a fitting process the cost function is what one wishes to minimize. The cost function used for my fitting process was the sum of least squares in equation 35. The minimized cost function value was used as error estimates for Fit 1 and 2. The synapse parameters in Fit 3 are fitted on a shorter interval than Fit 1 and 2 and the cost function's minimized value would naturally have a lower value than the cost function ranging over a longer interval. I used the fitted synapse parameters from Fit 3 and estimated the errors on the interval 500 [ms] to the end time of the simulation, using equation 35. The same was done for the synapse parameters from Fit 4.

Cost function / Error estimate

$$\sum_{t_s}^{te}(V(t)_{pointmodel} - V(t)_{soma}^{Hay})^2 \tag{35}$$

t_s is the time for the synapse stimuli [ms](I did not start the calculation at t_0, because I wanted equal length intervals to evaluate the error estimates/ cost function for the different EPSPs).
t_e is the end time of the signal [ms].
$V(t)_{pointmodel}$ is the point models synapse response from equation 58 at time t [mV].
$V(t)_{soma}^{Hay}$ is the Hay-model's EPSP recorded in soma at time t [mV].

6.3 Variation in parameter values as a function of distance from soma

In section 6.2 I went through the method for fitting synapse parameters for the point model. The result was sets of four synapse parameters for the point model (amplitude, t_0, τ_1 and τ_2) for each EPSP the Hay-model generated. The sets of synapse parameters for the point model were linked to the Hay-model by the somatic distance for each EPSP from the Hay-model.

I wanted to present my results as functions for each of the four different synapse parameters, dependent on distance from soma. If you wanted a synapse for the point model able to recreate the EPSP from the Hay-model generated at a given distance from soma, you use this somatic distance as a variable in the four equations for the synapse parameters and evaluate them. The result would be four synapse parameters for the point model able to build a synapse that could recreate the EPSP from the Hay-model actually generated at the given somatic distance.

I needed a flexible function that could capture different trends in the parameters, dependent on the distance from soma. The choice fell on the sigmoid function, see equation 36.

The sigmoid function can capture linear, decaying, rising and S-shaped relationships between the parameters and the distance from soma. I modified the sigmoid function to be able to flip and scale it more, see equation 37.

Figure 10 shows a special case for equation 37, where it acts like the simple sigmoid function in equation 36, because of the choice of parameters (x_0, x_1, x_2, x_3, and d_0).

Figure 11 shows the flexibility of the sigmoid function and how one easily can manipulate its shape through parameter choices.

Through curve fitting with the sigmoid function and the four sets of synapse parameters dependent on distance from soma, I obtained fitted values for the sigmoid function's parameters,x_0, x_1, x_2, x_3, and d_0. The fitting process used the same package in Python as section 6.2. For Python code, see section 11.3 in Appendix B. The cost function was the sum of least squares in equation 38, which was used as error estimate when it was minimized.

$$s(d) = \frac{1}{1 + e^{-d}} \tag{36}$$

$$s(d) = x_0 + \frac{x_1}{x_2 + e^{-x_3(d-d_0)}} \tag{37}$$

$$\sum_{d_0}^{d_e} (S(d) - Syn_{param}(d))^2 \tag{38}$$

d_0 is the starting point of the interval for distances from soma [μm]. d_e is the end point of the interval for distances from soma [μm]. $S(d)$ is the sigmoid function's value at distance from soma d [unit for the synapse parameter]. $Syn_{param}(d)$ is the parameter value at distance from soma d [unit for the synapse parameter].

The parameters in the sigmoid function, x_1, x_2, x_3, x_4 and d_0 describe different aspects of its shape. d_0 is the delay and shifts the graph horizontally. When d=d_0 the function changes its slope. x_3 makes it possible to let the sigmoid function be both decreasing and increasing. A positive x_3 makes the exponential term in the denominator a decaying exponential and the sigmoid function rises with t. A negative x_3 makes the exponential term a rising exponential and the sigmoid function decays with time. In the case of a positive x_3 the sigmoid function will approach the value x_0 when t$\rightarrow -\infty$ and the value $x_0 + \frac{x_1}{x_2}$ when t$\rightarrow \infty$. For the case of a negative x_3 the function will approach the value $x_0 + \frac{x_1}{x_2}$ when t$\rightarrow -\infty$ and the value x_0 when t$\rightarrow \infty$. The value of x_3 describes how fast the sigmoid function changes in other words how steep the slopes are. The larger x_3, the steeper curve.

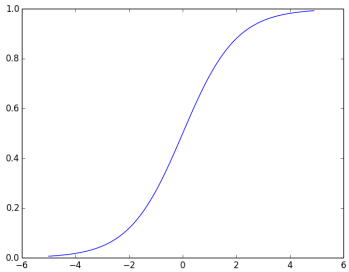
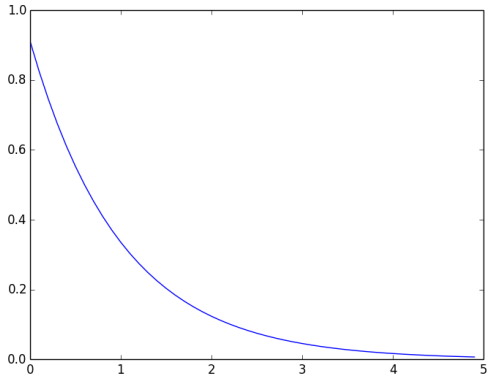
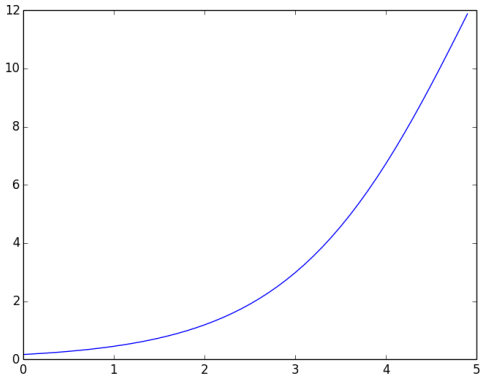


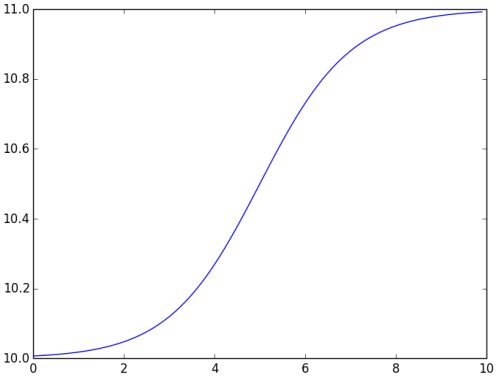
Figure 10. Sigmoid function from equation 37, with parameters: x_0=0, x_1= 1, x_2=1, x_3=1, and d_0=0, gives the special case where the modified sigmoid function behaves like the original sigmoid function from equation 36.



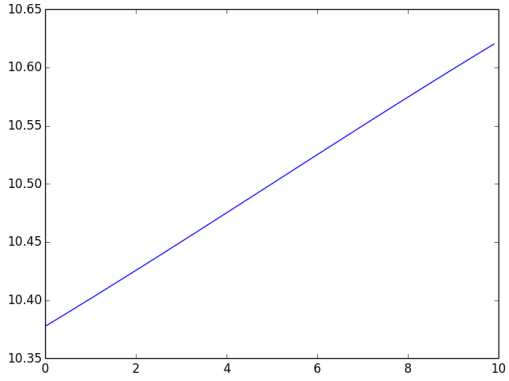
(a) sigmoid function behaving like a decaying exponential



(b) sigmoid function behaving like a rising exponential



(c) sigmoid function on a positive interval



(d) sigmoid function behaving linear

Figure 11. Examples of the sigmoid function in equation 37 with different parameters: $[x_0,x_1,x_2,x_3,x_4,t_0]$.
(a): [0,1000,1,1,-7], (b): [0, 25,1,1,5], (c): [10,1,1,1,5], (d): [10,1,1,0.1,5]

7 Results

7.1 Modelling synaptic responses in the point model

To be able to fit the point neuron's response to the Hay-models EPSPs, I needed an analytical solution for its response to a synapse input.
I started with the equation for a simple RC-neuron, from section 5.1.

$$C_m \cdot \frac{dV}{dt} = -i_i - i_{syn} \tag{39}$$

I write the quasi-ohmic relation for the ionic current, i_i. The synapse used on the point model is current based, which means that the synaptic current response, i_{syn} to a synaptic input is proportional to the chosen time course function f(T) for the synaptic current. i_{syn} can be expressed as the time course of the synaptic current f(t) multiplied with a scaling constant K.

$$C_m \cdot \frac{dV}{dt} = -g(V - V_{rest}) - K \cdot f(t) \tag{40}$$

$g = \frac{1}{R_m}$ and the membrane time constant $\tau_m = C_m \cdot R_m$

$$\frac{dV}{dt} = -\frac{(V - V_{rest})}{\tau_m} - \frac{K \cdot f(t)}{C_m} \tag{41}$$

Multiply both sides by $e^{\frac{t}{\tau_m}}$

$$\frac{dV}{dt} e^{\frac{t}{\tau_m}} = -\frac{V - V_{rest}}{\tau_m} e^{\frac{t}{\tau_m}} - \frac{K \cdot f(t)}{C_m} e^{\frac{t}{\tau_m}} \tag{42}$$

$$(\frac{dV}{dt} + \frac{V}{\tau_m}) e^{\frac{t}{\tau_m}} = \frac{V_{rest}}{\tau_m} e^{\frac{t}{\tau_m}} - \frac{K \cdot f(t)}{C_m} e^{\frac{t}{\tau_m}} \tag{43}$$

The left side is the derivative of $Ve^{\frac{t}{\tau_m}}$.

$$(Ve^{\frac{t}{\tau_m}})' = \left(\frac{V_{rest}}{\tau_m} - \frac{K \cdot f(t)}{C_m} \right) \cdot e^{\frac{t}{\tau_m}} \tag{44}$$

Integrate both sides from 0 to T.

$$Ve^{\frac{T}{\tau_m}} - V(0) = \int_0^T ((\frac{V_{rest}}{\tau_m} - \frac{K \cdot f(t)}{C_m}) e^{\frac{t}{\tau_m}}) dt \tag{45}$$

Use the initial condition: $V(0) = V_{rest}$.

$$Ve^{\frac{T}{\tau_m}} - V_{rest} = \int_0^T \frac{V_{rest}}{\tau_m} e^{\frac{t}{\tau_m}} dt - \int_0^T \frac{K \cdot f(t)}{C_m} e^{\frac{t}{\tau_m}} dt \tag{46}$$

$$Ve^{\frac{T}{\tau_m}} - V_{rest} = \frac{V_{rest}}{\tau_m} \tau_m (e^{\frac{T}{\tau_m}} - 1) - \int_0^T \frac{K \cdot f(t)}{C_m} e^{\frac{t}{\tau_m}} dt \tag{47}$$

$$(V - V_{rest}) e^{\frac{T}{\tau_m}} = -\int_0^T \frac{K \cdot f(t)}{C_m} e^{\frac{t}{\tau_m}} dt \tag{48}$$

Switch places for T and t.

$$V \cdot e^{\frac{t}{\tau_m}} = V_{rest} e^{\frac{t}{\tau_m}} - \int_0^t \frac{K \cdot f(T)}{C_m} e^{\frac{T}{\tau_m}} dT \tag{49}$$

$$V(t) = V_{rest} - \int_0^t \frac{K \cdot f(T)}{C_m} e^{\frac{(T-t)}{\tau_m}} dT \tag{50}$$

I chose the beta function to describe the synapse's current time course. It is valid for $T \geq t_0$ and zero for $T < t_0$. The beta function is given by equation 51.

$$I_{syn}(T) \propto e^{\frac{-(T-t_0)}{\tau_1}} - e^{\frac{-(T-t_0)}{\tau_2}} = f(T) \tag{51}$$

The synaptic current is given per area of the point neuron, denoted a. I divide f(t) by a.

$$f(T) = \frac{K \cdot I_{syn}(T)}{a} = \frac{K(e^{\frac{-(T-t_0)}{\tau_1}} - e^{\frac{-(T-t_0)}{\tau_2}})}{a} \tag{52}$$

I use the expression for f(T) in equation 52 in equation 50 and get equation 53.

Because the conductance for the synapse is zero for $T < t_0$, the limits for the integral were changed to from t_0 to t.

$$V(t) = V_{rest} - \frac{K}{C_m \cdot a} \cdot e^{\frac{-t}{\tau_m}} \int_{t_0}^t (e^{\frac{-(T-t_0)}{\tau_1}} - e^{\frac{-(T-t_0)}{\tau_2}}) e^{\frac{T}{\tau_m}} dT \tag{53}$$

$$V(t) = V_{rest} - \frac{K}{C_m \cdot a} \cdot e^{\frac{-t}{\tau_m}} \left(\int_{t_0}^t e^{\frac{-(T-t_0)}{\tau_1}} \cdot e^{\frac{T}{\tau_m}} dT - \int_{t_0}^t e^{\frac{-(T-t_0)}{\tau_2}} \cdot e^{\frac{T}{\tau_m}} dT \right) \tag{54}$$

$$V(t) = V_{rest} - \frac{K}{C_m \cdot a} \cdot e^{\frac{-t}{\tau_m}} \left(\frac{t_0}{\tau_1} \int_{t_0}^t e^{T(\frac{\tau_1 - \tau_m}{\tau_1 \cdot \tau_m})} dT - e^{\frac{t_0}{\tau_2}} \int_{t_0}^t e^{T(\frac{\tau_2 - \tau_m}{\tau_2 \cdot \tau_m})} dT \right) \tag{55}$$

$$V(t) = V_{rest} - \frac{K}{C_m \cdot a} \cdot e^{\frac{-t}{\tau_m}} \left(\frac{\tau_1 \cdot \tau_m}{\tau_1 - \tau_m} \cdot e^{\frac{t_0}{\tau_1}} \left[e^{T(\frac{\tau_1 - \tau_m}{\tau_1 \cdot \tau_m})} \right]_{T=t_0}^{T=t} - \frac{\tau_2 \cdot \tau_m}{\tau_2 - \tau_m} \cdot e^{\frac{t_0}{\tau_2}} \left[e^{T(\frac{\tau_2 - \tau_m}{\tau_2 \cdot \tau_m})} \right]_{T=t_0}^{T=t} \right) \tag{56}$$

Equation 57 was normalized before implemented in Python, to make the constant $\frac{K}{C_m \cdot A}$ the maximum

$$V(t) = V_{rest} + \frac{K}{C_m \cdot a} \cdot e^{\frac{-(t-t_0)}{\tau_m}} \left(\frac{\tau_1 \cdot \tau_m}{\tau_1 - \tau_m} - \frac{\tau_2 \cdot \tau_m}{\tau_2 - \tau_m} \right) + \frac{K}{C_m \cdot a} \left(\frac{\tau_2 \cdot \tau_m}{\tau_2 - \tau_m} \cdot e^{\frac{-(t-t_0)}{\tau_2}} - \frac{\tau_1 \cdot \tau_m}{\tau_1 - \tau_m} \cdot e^{\frac{-(t-t_0)}{\tau_1}} \right) \tag{57}$$

Equation 57 is valid for t$\geq t_0$ and has the value V_{rest} for t$<t_0$.

Figure 12. The synaptic potential response for a point neuron with a current based synapse with conductance modelled by the beta function.

value of the function and the amplitude of the synaptic response.
The normalized synaptic response for the point model is given by equation 58 in figure 13.

$$V(t) = V_{rest} - V_{max} \cdot \left(\frac{p(t)}{p(t)_{max}} \right) \tag{58}$$

where:

$$p(t) = \left(\frac{\tau_1 \cdot \tau_m}{\tau_1 - \tau_m} - \frac{\tau_2 \cdot \tau_m}{\tau_2 - \tau_m} \right) e^{\frac{-(t-t_0)}{\tau_m}} - \frac{\tau_1 \cdot \tau_m}{\tau_1 - \tau_m} \cdot e^{\frac{-(t-t_0)}{\tau_1}} + \frac{\tau_2 \cdot \tau_m}{\tau_2 - \tau_m} \cdot e^{\frac{-(t-t_0)}{\tau_2}} \tag{59}$$

$$p(t)_{max} = |p(t)|_{max} \tag{60}$$

V_{rest} is the cells resting potential and V_{max} is the amplitude taken from V_{rest}.

Figure 13. The normalized synaptic response for a current based synapse with conductance modelled by the beta function, for a point neuron.

7.2 Estimation of the cell parameter τ_m for the point model

The analytical solution for the point model's response to synaptic input in equation 58 on page 30 contains a cell parameter, τ_m, this is the membrane time constant. τ_m decides how fast the membrane potential responds to an increase or decrease in potential differences across the membrane [5].
I have to decide the value for τ_m for the point model. Because I wanted the point model to act the same way as the Hay-model as far as possible, I wanted to fit the cell parameter τ_m for the point model through curve fitting with the Hay-model.

I need an equation for the membrane potential's response due to a current stimuli, containing τ_m for the point model.

I could have used the analytical solution for the point model's synaptic response, from section 7.1, but a simpler equation can be obtained by using a current injection instead of a synapse input. With a current injection instead of a synapse input I get an equation without the synapse parameters, and do not have to use their fitted values.
I began with the equation for the RC-circuit, from section 5.1 [5].

$$C_m \frac{dV}{dt} = \frac{E_m - V}{R_m} + \frac{I_e}{a} \tag{61}$$

This is an ordinary differential equation that can be solved using the method of integrating factors [17]

$$\frac{dV}{dt} + p(t) \cdot V = q(t) \tag{62}$$

$p(t) = \frac{V}{\tau_m}, q(t) = \frac{E_m}{\tau_m} + \frac{I_e}{a}$ The integrating factor is $e^{\int p(t)dt}$. In my case this gives $e^{\frac{t}{\tau_m}}$. Next step is to multiply both sides by the integrating factor.

$$e^{\frac{t}{\tau_m}}[\frac{dV}{dt} + \frac{V}{\tau_m}] = e^{\frac{t}{\tau_m}}[\frac{E_m}{\tau_m} + \frac{I_e}{a}] \tag{63}$$

The left side of the equation can be written as a derivative.

$$(e^{\frac{t}{\tau_m}} \cdot V)' \tag{64}$$

I look at the case for t<t_e and integrate both sides of the equation from 0 to t with respect to t.

$$e^{\frac{t}{\tau_m}} \cdot V - V(0) = \int_0^{t_0} e^{\frac{t}{\tau_m}}[\frac{E_m}{\tau_m}]dt + \int_{t_0}^{t} e^{\frac{t}{\tau_m}}[\frac{E_m}{\tau_m} + \frac{I_e}{C_m \cdot a}]dt \tag{65}$$

I use the initial condition V(0)=V_{rest} and the condition $\tau_m = R_m \cdot C_m \rightarrow \frac{\tau_m}{C_m} = R_m$

$$V(t)e^{\frac{t}{\tau_m}} - V_{rest} = E_m(e^{\frac{t_0}{\tau_m}} - 1) + E_m(e^{\frac{t}{\tau_m}} - e^{\frac{t_0}{\tau_m}}) + \frac{I_e \cdot R_m}{a}(e^{\frac{t}{\tau_m}} - e^{\frac{t_0}{\tau_m}}) \tag{66}$$

I use that for a point model with one passive ion channel, the resting potential equals the membrane's reversal potential E_m. Equation 69 in figure 14 describe the point model's membrane potential for $t_0 \leq t \leq t_e$.
For the equation describing the decay back to resting potential for t>t_e I start by changing limits in the integrals from equation 65 to from t_e to t, getting equation 67.

$$e^{\frac{t}{\tau_m}} \cdot V(t) - V(t_e)e^{\frac{t_e}{\tau_m}} = \int_{t_e}^{t} e^{\frac{t}{\tau_m}}[\frac{E_m}{\tau_m}]dt \tag{67}$$

$$e^{\frac{t}{\tau_m}} \cdot V(t) = V(t_e)e^{\frac{t_e}{\tau_m}} + E_m(e^{\frac{t}{\tau_m}} - e^{\frac{t_e}{\tau_m}}) \tag{68}$$

$$V(t) = V_{rest} + \frac{I_e \cdot R_m}{a}(1 - e^{\frac{t_0 - t}{\tau_m}}) \tag{69}$$

Figure 14. Equation 69 describes the point model's response due to a constant injected current I_e and is valid for the $t_0 \leq t \leq t_e$.

$$V(t) = V_{rest} + (V(t_e) - V_{rest})e^{\frac{t_e - t}{\tau_m}} \tag{70}$$

$$V(te) = V_{rest} + \frac{I_e \cdot R_m}{a}(1 - e^{\frac{t_0 - t_e}{\tau_m}}) \tag{71}$$

Figure 15. Equation 70 describes how the point model decays towards its resting potential after the current injection stops at $t = t_e$. The membrane potential starts at the value V_{te}, described by equation 71 at time $t = t_e$ and then decays towards V_{rest} as t increases. The equation is valid for $t > t_e$. V(t_e) is the maximum response for the point model due to the injected current. Evaluating equation 69 for t=t_e gives an expression for V(t_e) in equation 71 for use in equation 70.

I use that for a point model with one ion passive channel, the resting potential equals the membranes reversal potential E_m. The point models decay towards resting potential for $t > t_e$ is described in equation 70 in figure 15.

The equations 69 and 70 describe the membrane responses for the rise of the membrane potential and the decay respectively. The cell parameters R_m, specific membrane resistance and a, the area of the point model are present in the two equations and I have to decide their values as well.

To be able to decide the value of the cell parameter τ_m through curve fitting with the Hay-model, I need a response for the Hay-model to an injected current.

I placed an electrode in its soma, and gave it a current injection of 0.1 n amp for 200 ms with a delay of 200 ms. The current injection, $\tilde{I_e}$, is a heavy side function, because it rises and decays instantaneously.

$$\begin{cases} \tilde{I_e} = 0 & t < t_0 \\ \tilde{I_e} = I_e & t_0 < t < t_e \end{cases}$$

The parameters for my case are listed below.

I_e	0.1 nA	The current injected
t_0	200 ms	The start time for the current injection
t_e	400 ms	The end time for the current injection
t_{end}	600 ms	The end time of the simulation

I chose to fit τ_m for the point models decay process towards resting potential after the current injection, using equation 70. The equation 70 was implemented in Python with τ_m and the ratio R_m/a as unknown constants and fitted against the Hay-models response to the current injection for the time interval t_e to t_{end}. Figure 16 show the fitted equation 70 for the point model's decay process and the Hay-model's response to the current injection.

The result was τ_m=9.67 [ms] and the ratio $R_m/a = 44.34$ [Ω]

The result $\tau_m = 9.69$ [ms] was used in the derived expression for the point model's response to a synapse input in equation 58 on page 30.

The reason why I can compare the response in the active Hay-Model with the passive point model's

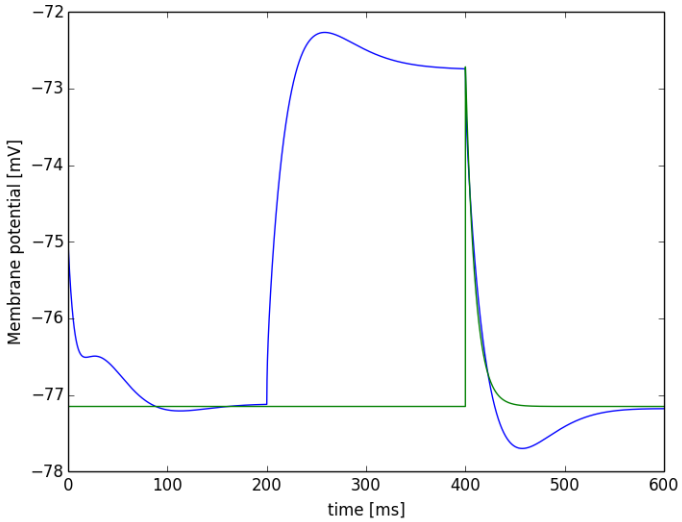


Figure 16. The point models decay towards resting potential after a current injection has stopped, at time t_e=400 ms, described by equation 70 (green) fitted against the Hay-models response to an equal injected current (blue). The interval for the curve fitting was from t_e=400 ms to t_{end}= 600 ms. The signals were fitted to decide the point models membrane time constant, τ_m. The result was a τ_m for the point model with value 9.67 ms.

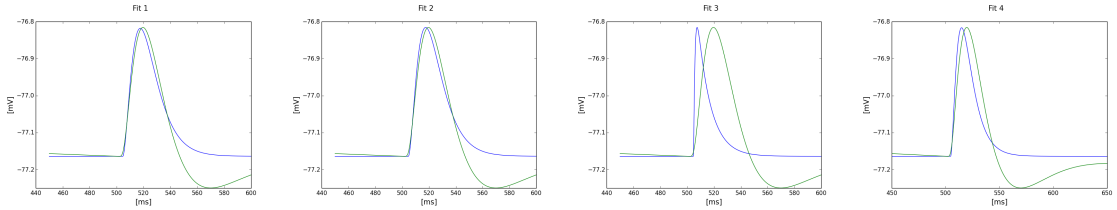
response to the injected current, is because of the low amount of current I injected. The active channels will have a low activation for these changes in the membrane potential and can be neglected.

7.3 Fitting the synapse parameters for the point model

In this section I will go through the results for fitting the synapse parameters for the point model.

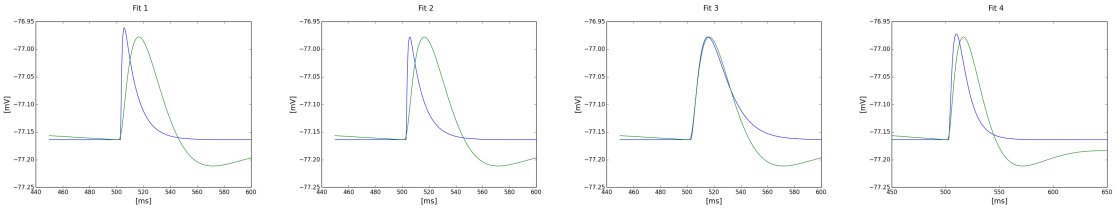
I fitted the synapse parameters for the point model in four different ways, described in section 6.2. The four ways for fitting synapse parameters resulted in four sets of synapse parameters for each EPSP generated by the Hay-model. The results for the four fitting methods are called Fit 1, 2,3 and 4.

Figure 17, 18 and 19 show that the different fitting methods could fail fitting EPSPs from the Hay-model, and that Fit 4, which used mean parameter values from Fit 1, 2 and 3 smooths the error. Error estimates are given in mV. The EPSPs from the Hay-model are indexed by their compartment number on the morphology. As mentioned in section 6.2, I divided the EPSPs from the Hay-model into four



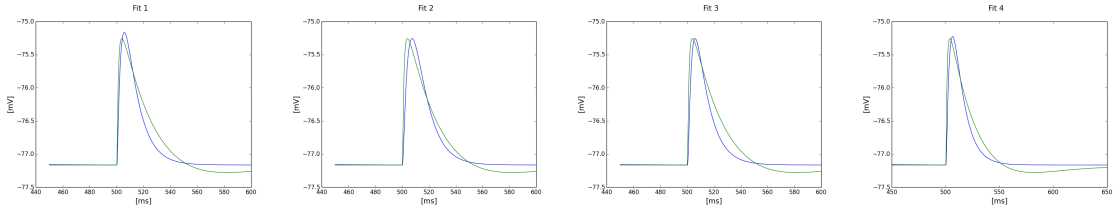
(a) Fit 1, error: 119 (b) Fit 2, error: 119 (c) Fit 3, error: 198 (d) Fit 4, error: 139

Figure 17. Fitting EPSP: 411 from the apical region, generated at 1186 μm from soma. The graphs and the error estimates show that, fit 3 fails fitting this EPSP



(a) Fit 1, error: 122 (b) Fit 2, error: 122 (c) Fit 3, error: 81 (d) Fit 4, error: 102

Figure 18. Fitting EPSP: 516 from the apical region, generated 951 μm from soma. The graphs and error estimates show that fit 1 and 2 fails fitting this EPSP.

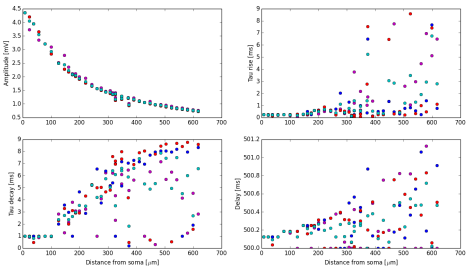


(a) Fit 1, error: 286 (b) Fit 2, error: 266 (c) Fit 3, error: 299 (d) Fit 4, error: 279

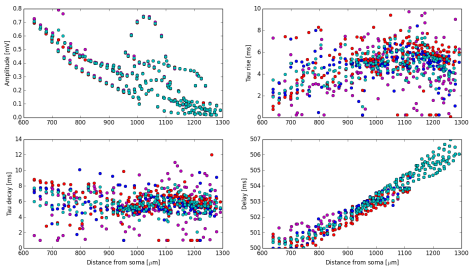
Figure 19. Fitting EPSP: 305 from the stem, generated 200 μm from soma. This is an example on fit 4s ability to smooth out errors, even when the other fits does not seem to fail.

groups; apical, stem, basal and branches. This resulted in sets of synapse parameters (A, t_0, τ_1 and τ_2) for each part of the neuron, for each of the four fitting methods (Fit 1, Fit 2, Fit 3 and Fit 4). This made

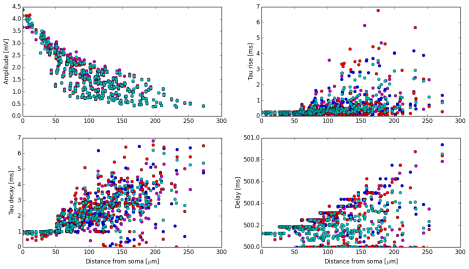
a total of 64 sets of fitted parameters. Figure 20 shows the fitted synapse parameters, with corresponding error estimates in table 2. Fit 1 is coloured magenta, fit 2 is blue, Fit 3 is red and fit 4 is cyan.
To gain more insight into the fitting process I also computed the standard deviations of the error estimates, see table 2, and graphed the errors for each fit against EPSPs from the Hay-model for each fitting method, dependent of somatic distance for the EPSPs, see figure 21.



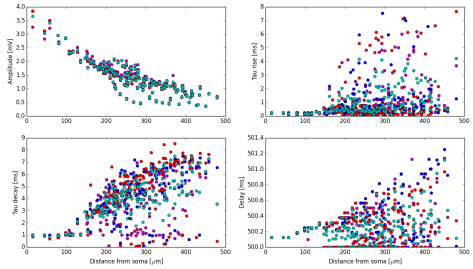
(a) The fitted parameters for the stem



(b) The fitted parameters for the apical

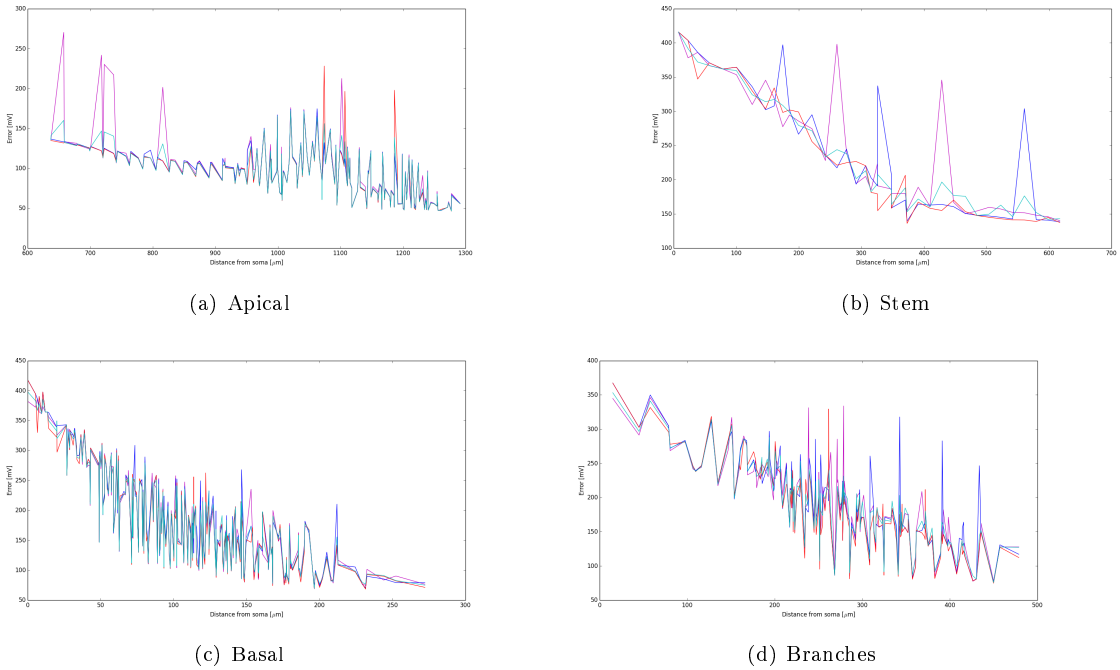


(c) The fitted parameters for the basal



(d) The fitted parameters for the branches

Figure 20. The fitted synapse parameters for the point model. The four synapse parameters are fitted in four different ways for the four parts of the neuron. Magenta: Fit 1, blue: Fit 2, red: Fit 3 and cyan: Fit 4. The blue dots are not showing for the amplitude, this is because fit 2 and 3 are have the same bounds on this parameter, it is fixed to the Hay-neuron's amplitude for the given index. The blue dots are all in the same places as the red.



(a) Apical

(b) Stem

(c) Basal

(d) Branches

Figure 21. The error estimates for the four fitting methods plotted for all synapse placements possible in the Hay-model, divided into the four parts of the morphology and dependent on somatic distance for the EPSPs from the Hay-model. The dots represent error estimates for the point model's attempt to recreate the Hay signal with its set of the four fitted synapse parameters. Magenta: Fit 1, blue: Fit 2, red, Fit 3 and cyan: Fit 4

	Fit 1	Fit 2	Fit 3	Fit 4
apical	97	93	93	93
stem	241	240	228	234
basal	191	194	190	191
branches	192	192	185	191

Table 2. The error estimates for the four fitting methods for the four parts of the neuron. There are 642 possible synapse placements for the Hay-model, this resulted in 642 sets of synapse parameters and the same number of error estimates. This magnitude of data is too big to analyse one by one, therefore errors are given in $\overline{error/fitting}$ [mV] for each fitting method for each part of the neuron. The error estimates describe how good the fitted synapse parameters in sets create synapses able to recreate the Hay-model's EPSPs.

	Fit 1	Fit 2	Fit 3	Fit 4
Apical	38	29	32	30
Stem	89	92	88	83
Basal	78	80	79	78
Branches	58	59	59	55

Table 3. The standard deviations for the error estimates in table 2. The standard deviations for the errors describe how often a fit fails for the given fitting method in a given part of the neuron. A failed fit will give synapse parameters that might deviate from the trend I hoped to capture.

In figure 20 on page 35 there are some trends in the fitted parameters, dependent on distance from soma.

The trend is showing especially for the amplitude which is decreasing by distance from soma for all the fitting methods and parts of the neuron. It can obviously be fitted as a sigmoid or an exponential function, with small errors.

The delay is increasing with distance from soma, but the clearest result is in the apical. This can be explained by the placement of the apical on the Hay-models morphology, which is located at the longest distance from soma. The delay will be larger and is easier detected by the fitting process.

Delay from the other parts of the neuron also have an increasing trend, but the fitted points differ from the trend and some are quite low. Some lie at 500 ms. This can be explained by the Hay-signal's shape, which the passive point neuron cannot recreate. It therefore tries to start the EPSP earlier, to get a better fit.

I tried to fit the EPSPs without any bound, but then the fitted delay went far below 500 ms. This indicated that the fitting process chose to start the fitted EPSP for the point model, early in some cases to avoid the large error that came from the different natures of the Hay-model's EPSPs and the point model's attempt to recreate them. I used bounds on the delays in the fitting processes. The delays could not go below 500 ms in value, because a delay under 500 ms would be unphysical (nothing can happen before the neuron receives a synaptic input at time t = 500 ms).

I expected tau rise and tau decay to increase with distance, because the EPSPs get broadened when they are generated at longer distances from soma and experience more spread and weakening [18]. A flattened signal use more time rising and decaying, and therefore have larger expected rise and decay time constants.

Most of the fitted parameters express an increasing trend in rise and decay time constants, but there is much spread. The magnitude of scattered parameter values for the time constants can be explained by the Hay-EPSPs shape arising from its active nature and the choice of synapse parameters we fit and their bounds. The fitting process can only change four parameters to achieve the best fit to the Hay-model's EPSPs. Some of the synapse parameters are even bounded. The most important parameters to decide the shape of the signal are the rise and the decay time constants. The rise and decay time constants probably get the responsibility of being both rise and decay and compensating for the active properties of the Hay-model. Because the amplitude and delay express only the basic properties for the EPSPs, the peak value and the time the signal arrived in soma.

The errors plotted for the parameter fitting in figure 21 show a clear decreasing trend in error as the distance from soma increases. Because the errors are estimated using the absolute difference between the Hay-model's EPSPs and the point model's recreation of the Hay-models EPSPs I can not amortize this with a bigger amplitude giving rise to bigger errors.

The reason for the decrease in error with distance might be because a signal generated at a long distance from soma encounters more flattening by propagating longer in the dendrites. A signal propagating longer distances in the dendrites gets more broadened [18], and the point point models analytical solution to synapse input seem to fit broadened EPSPs better than steep EPSPs.

The error estimates and standard deviation for the error estimates are summarized in table 2 and 3, respectively. The standard deviation for the error estimate expresses the mean aberration from the mean error estimate and can be used to indicate whether a fitting method often fails drasticly and give synapse parameters deviating from the trend I hoped to capture. The two tables show that fitting method 1 and 2 have (for the most parts of the neuron) the largest error estimates and the largest standard deviation for the error estimates. Fit 3 has the lowest error estimate, followed by Fit 4. Fit 4 has the lowest standard deviation for the error estimates followed by Fit 3.

I expect Fit 4 to capture the trend in the fitted synapse parameters best among the different fitting methods, because it has low error estimates and the lowest standard deviation for the error estimates, for most of the parts of the neuron.

7.4 Fitting synapse parameters for the whole neuron

I also fitted synapse parameters with the four fitting methods described in section 6.2, for all the synapse placements without splitting the Hay-neuron's morphology into four parts.

This would be a more easily applied and implemented result, because the number of result functions would be reduced from 16 to four.

Figure 22 shows the fitted parameters for each of the four fitting methods, with corresponding error estimates and standard deviations for the error estimates in table 4. The error estimates are graphed for each synapse placement in figure 23. Synapse parameters fitted with different fitting method are color coded as following: Fit 1 is magenta, Fit 2 is blue, Fit 3 is red and Fit 4 is cyan.

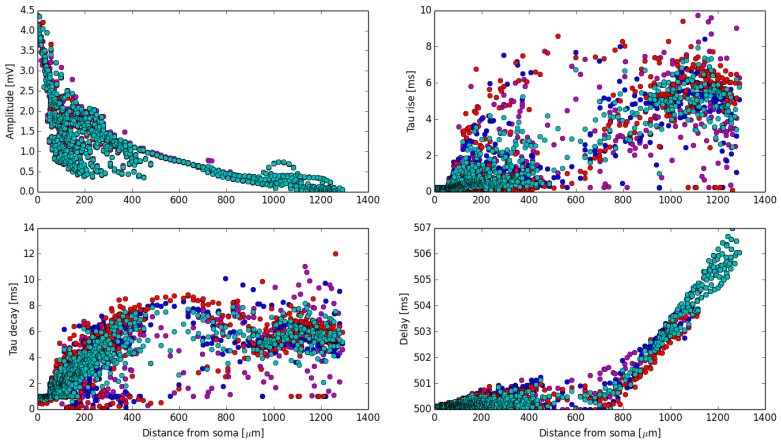


Figure 22. The fitted parameters for the whole neuron, containing 642 synapses. Fit 1 is magenta, fit 2 is blue, fit 3 is red and fit 4 is cyan. The blue dots are not showing for the amplitude, this is because fit 2 and 3 are have the same bounds on this parameter, it is fixed to the Hay-neuron's amplitude for the given EPSP. The blue dots are all in the same places as the red.

	Fit 1	Fit 2	Fit 3	Fit 4
Error/fitting	168	168	164	166
SD Error	79	81	79	78

Table 4. The error estimates for the four fitting methods for all the 642 synapse placements in the Hay-model.The errors are given as $\overline{error/fitting}$ [mV] for each fitting method. The error estimates describe how good the fitted synapse parameters in sets create synapses able to recreate the Hay-model's EPSPs. The standard deviations for the error estimates describe how often a fit fails for the given fitting method. A failed fit will give synapse parameters that might deviate from the trend I hoped to capture.

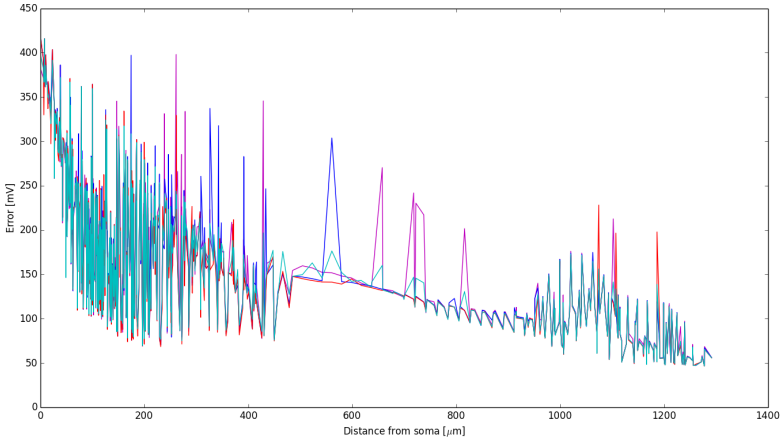


Figure 23. The error estimates for the four fitting methods, dependent on somatic distance from the Hay-model. The error estimates represent the point model's attempt to recreate the Hay signal with its sets of the four fitted synapse parameters. Magenta: Fit 1, blue: Fit 2, red: Fit 3 and cyan: Fit 4.

7.5 Curve fitting synapse parameters to functions to describe trends as distances from soma.

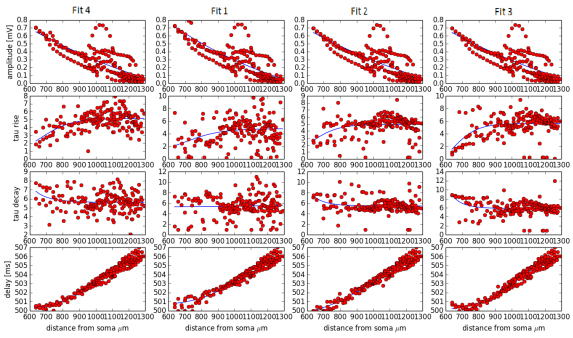
In section 6.3 I described that I captured the trend in the fitted synapse parameters by curve fitting them against the sigmoid function in equation 72.

$$s(d) = x_4 + (\frac{x_0}{x_1 + e^{-x_2 \cdot (d - x_3)}}) \tag{72}$$

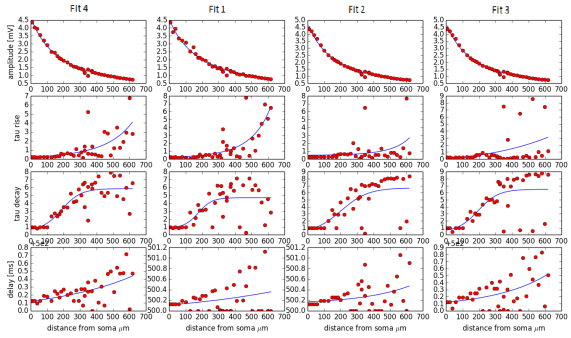
The result was 64 sigmoid result functions with five parameters each (x_0, x_1, x_2, x_3 and x_4). One result sigmoid function for each of the 64 synapse parameter sets described in section 7.3. For the parameter values for the fitted sigmoid functions, see tables 14, 15, 16 and 17 in Appendix A.

Table 5 show the error estimates for the curve fitting and figure 24 show the 64 fitted sigmoid functions and the respective synapse parameters.

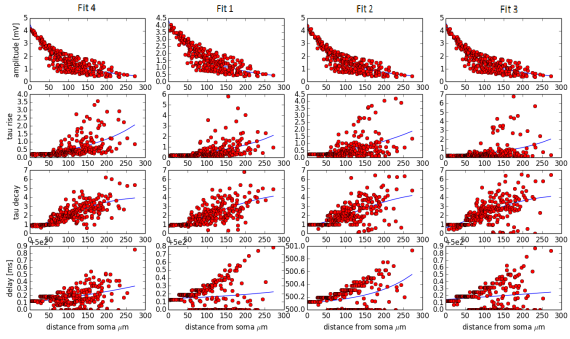
The Sigmoid result functions were dependent on the somatic distance the EPSP from the Hay-model was generated. The result functions give the synapse parameters to recreate a given EPSP from the Hay-model, when evaluated for the distance from soma the given EPSP was generated.



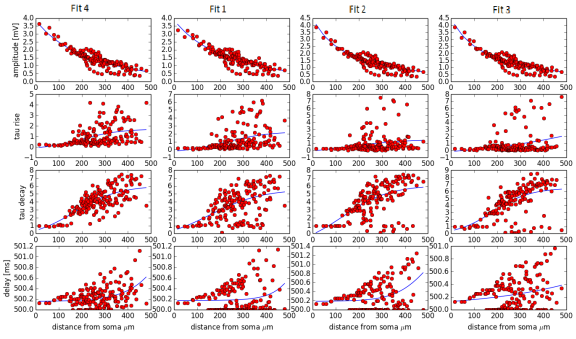
(a) Apical



(b) Stem



(c) Basal



(d) Branches

Figure 24. The fitted sigmoid function (blue lines) and the synapse parameters (red dots) from Fit 1, 2, 3 and 4. The synapse parameters are divided into four groups; apical, stem, basal and branches, dependent on which part the EPSP from the Hay-model they were fitted against were generated.

In section 7.3 I stated that I expected Fit 4 to capture the trend in the fitted synapse parameters best among the different fitting methods.

Part of neuron	Synapse parameter	Error fit 1	Error fit 2	Error fit 3	Error fit 4
Apical	amplitude	0.0899	0.0869	0.0869	0.0877
	tau rise	1.44	0.869	1.07	0.811
	tau decay	1.52	0.902	1.01	0.793
	delay	0.317	0.303	0.315	0.280
Stem	amplitude	0.0766	0.0549	0.0549	0.0518
	tau rise	0.669	0.726	1.26	0.644
	tau decay	1.19	1.40	1.51	0.836
	delay	0.206	0.180	0.147	0.0895
Basal	amplitude	0.336	0.311	0.311	0.318
	tau rise	0.351	0.373	0.456	0.305
	tau decay	0.561	0.661	0.706	0.485
	delay	0.125	0.141	0.131	0.0806
Branches	amplitude	0.201	0.182	0.182	0.185
	tau rise	0.947	0.808	0.947	0.642
	tau decay	1.26	1.18	1.18	0.811
	delay	0.210	0.240	0.188	0.134

Table 5. Error estimates for the curve fitting for the fitted synapse parameters against the sigmoid function. The errors describe the mean difference between the fitted sigmoid function and the synapse parameters for the point model in every point, for a given part of the neuron.

Table 5 shows that parameters fitted with method 4 gave smaller errors than the parameters fitted with other fitting methods, when fitted against sigmoid functions. There is an exception for the amplitude in apical, basal ans branches, but here the error is low for all the fitting methods.

It is important to have in mind that this fitting process captures a trend I believe, but do not know if exist in the synapse parameters. The errors only tell us that fit 4 have synapse parameters dependent on distance most suitable to be captured by a sigmoid function, and not that fit 4 have the most correct synapse parameters to recreate the Hay-model's EPSPs.

In section 7.4 I fitted synapse parameters for the point model with the four different fitting methods without dividing the Hay-model into four parts. I only fitted sigmoid functions to the synapse parameters obtained by fitting method 4. The results were four sigmoid functions, describing A, t_0, τ_1 and τ_2, dependent of somatic distance from the Hay-model. Figure 25 shows the four sigmoid functions fitted against synapse parameters obtained by fitting method 4 against all the EPSPs the Hay-model produced.

The errors per point for the curve fitting with the sigmoid function for the whole neuron as one are listed in table 6.

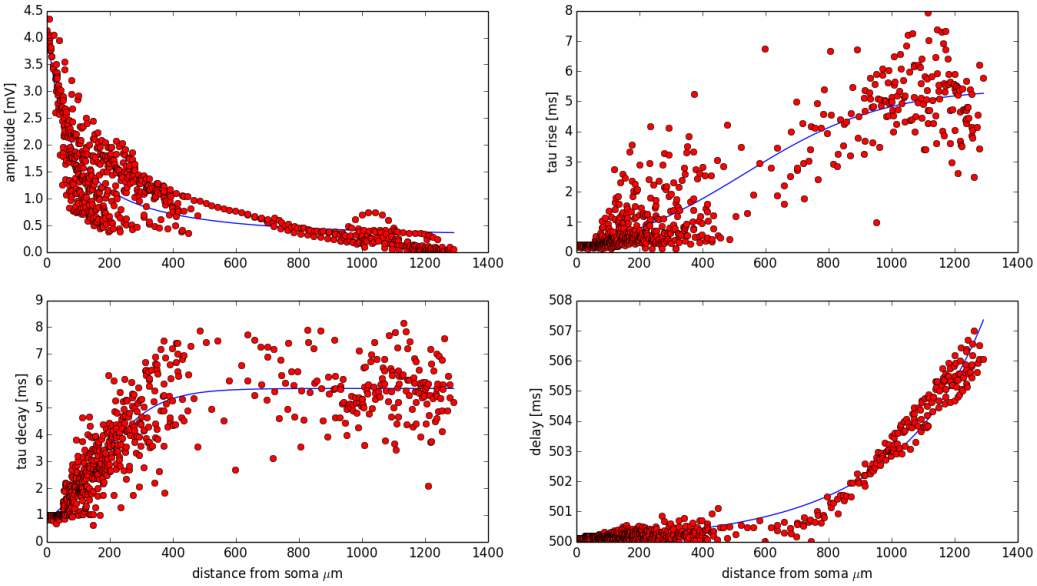


Figure 25. The fitted sigmoid functions (blue lines) and the synapse parameters (red dots) from Fit 4. The synapse parameters are obtained by fitting all the Hay-models EPSPs with fitting method 4.

Amplitude	Delay	Tau Rise	Tau Decay
0.356	0.203	0.571	0.690

Table 6. The errors per point for fitting the sigmoid curve to synapse parameters for Fit 4.

To investigate whether splitting the neuron into four parts was necessary to preserve the trends in the synapse parameter or not, I compared the values in the tables 6 and 18 on pages 42 and 60 respectively. Table 6 describe the mean difference between the fitted sigmoid curve and the synapse parameters for the whole neuron without splitting it up in the four parts. Table 18 describe the mean difference between the fitted sigmoid curve and the synapse parameters, with curve fitting done for each part of the neuron and then weighted by the fraction of synapses each part of the Hay-neuron contained. The values in the two tables can be compared, because they both describe an error estimate for the whole neuron per point. As I see the curve fitting done for the four parts of the neuron separately gives smaller error estimates per point. For amplitude and delay I see the largest differences in error estimates, 0.152 and 0.053, respectively, but for the rise and decay time constants the differences are only 0.02 and 0.016. This indicates that splitting the neuron into four parts gives a better fit for amplitude and a slightly better fit for delay, but for rise and decay the differences were almost insignificant.

7.6 Statistical analysis to choose the best result functions.

To end up with a synapse model with one set of sigmoid equations, describing its parameters, I had to decide which of the four fitting method that gave the best sigmoid functions. The best sigmoid result functions would be decided by which could recreate the EPSPs from the Hay-model with least deviation.

I tested the sigmoid result functions for each part of the neuron for each fitting method statistically using Python. For Python code, see section 11.4 in Appendix B. A statistical investigation was chosen, because my results are meant for point models in network simulations, were trends are more important than specific values.

The characteristics of an EPSP can be quantified by looking at values as the maximum soma-response [mV], the time for the maximum soma-response [ms], the half of the maximum soma-response [mV], the time for the half maximum soma-response [ms] and the time between the two half maximum soma-responses, as shown in figure 26.

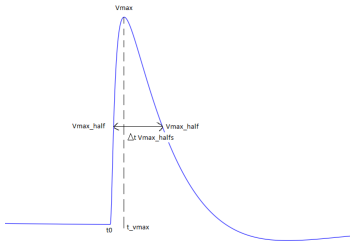


Figure 26. The values that captures the characteristics of an EPSP. Vmax is the maximum soma response [mV], t_Vmax is the time for the maximum soma response [ms], Vmax_half is the value for half the maximum soma response [mV] and time between the two half maximum soma responses is Δt Vmax_halfs [ms].

The statistics used for testing the sigmoid functions arising from different fitting methods are listed in table 7.

Statistical value	Symbol
mean amplitude	μ A
mean delay	μ t_0
mean time between half amplitudes	μ Δ_t
standard deviation for amplitude	sd A
standard deviation for delay	sd t_0
standard deviation for time between half amplitudes	sd Δ_t

Table 7. The statistical values I calculated and investigated to choose which of Fit 1, 2, 3 and 4 that gave the sigmoid functions that described synapse parameters for the point model that best recreated the EPSPs from the Hay-model.

Means (μ A, μ t_0, μ Δ_t) were calculated with equation 73 and standard deviations (sd A, sd t_0, sd Δ_t) were calculated with equation 74 [19].

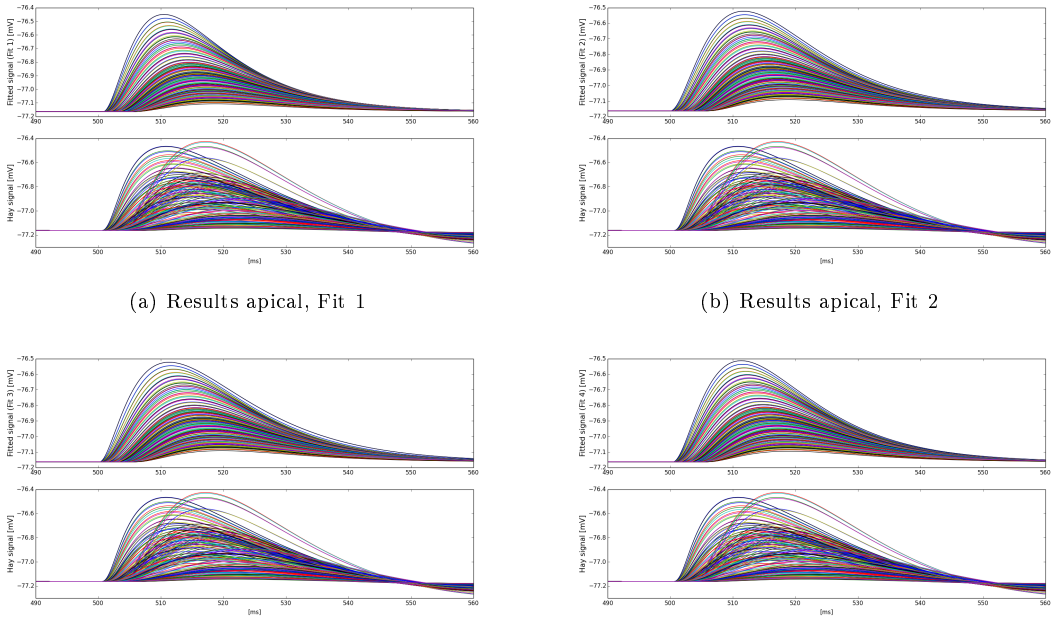
The three statistical values tells us much about the EPSPs shape. When the potential starts to rise is captured by delay, t_0, the amplitude, A, shows its maximum value and the time interval between the half value of the amplitude tells us if the point model's EPSPs have about right values for tau rise, τ_2,

and tau decay, τ_1.

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n}(x_i) \tag{73}$$

$$sd = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{74}$$

Figure 27 shows the results for the apical, with corresponding statistical analysis in table 8. Figure 28 shows the results for the stem, with corresponding statistical analysis in table 9. Figure 29 shows the results for the basal, with corresponding statistical analysis in table 10. Figure 30 shows the results for the branches, with corresponding statistical analysis in table 11.



(a) Results apical, Fit 1

(b) Results apical, Fit 2

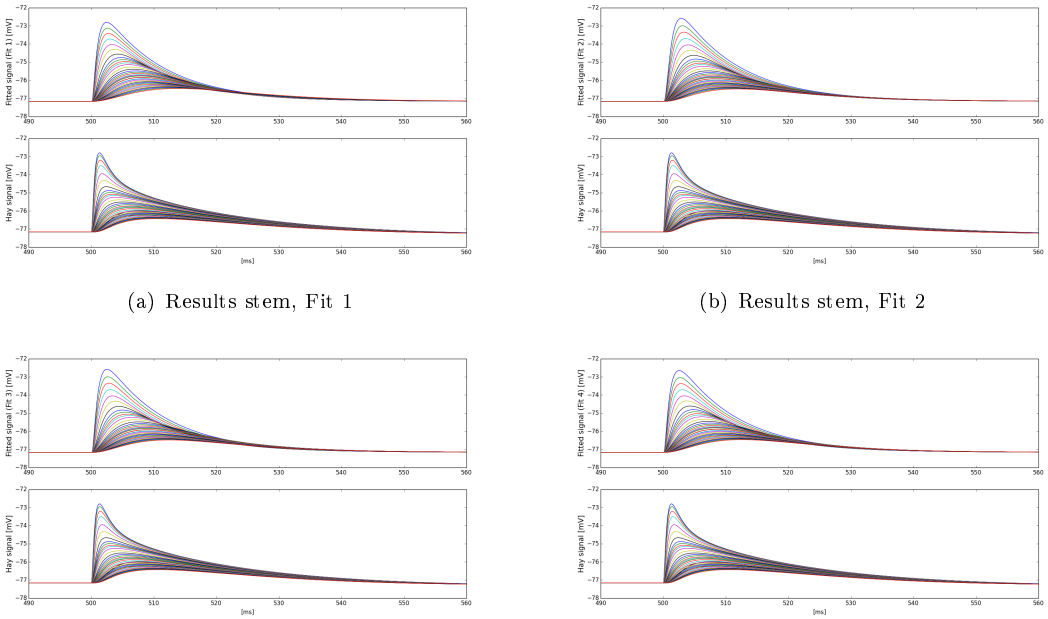
(c) Results apical, Fit 3

(d) Results apical, Fit 4

Figure 27. The figure shows the point model's recreation of the Hay model's EPSPs that were generated in the apical. The sigmoid result functions give synapse parameters to recreate an EPSP from the Hay-model when evaluated for the distance from soma the given EPSP was generated. Equation 58 on page 30 gives the point model's response when evaluated with synapse parameters obtained by the result sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the apical. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2.

	μ amp	μ del	$\mu\Delta$time vmax-halfs	sd amp	sd del	sdΔ time vmax-halfs
Fit 1	0.27256	503.78	21.678	0.16440	1.5923	0.83292
Fit 2	0.26639	503.68	22.302	0.14361	1.7543	0.18840
Fit 3	0.26639	503.58	23.439	0.14361	1.8250	0.38355
Fit 4	0.26830	503.66	22.472	0.14667	1.7254	0.48396
Hay-signal	0.26611	503.88	23.865	0.18365	1.4885	1.2849

Table 8. Statistical analysis for the EPSPs generated in the apical by the Hay-model and the point model's recreation using parameters given by the sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the apical. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2. The statistical values investigated are explained in the beginning of this section.



(a) Results stem, Fit 1

(b) Results stem, Fit 2

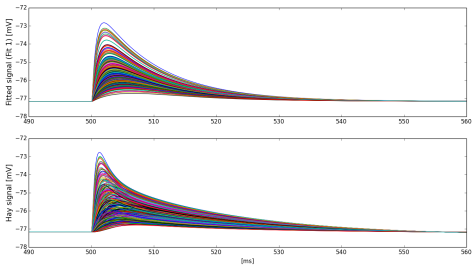
(c) Results stem, Fit 3

(d) Results stem, Fit 4

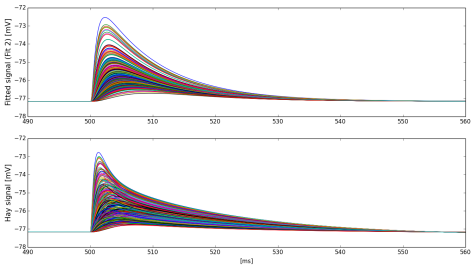
Figure 28. The figure shows the point model's recreation of the Hay model's EPSPs that were generated in the stem. The sigmoid result functions give synapse parameters to recreate an EPSP from the Hay-model when evaluated for the distance from soma the given EPSP was generated. Equation 58 on page 30 gives the point model's response when evaluated with synapse parameters obtained by the result sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the stem. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2.

	μ amp	μ del	$\mu\Delta$time vmax-halfs	sd amp	sd del	sdΔ time vmax-halfs
Fit 1	1.7501	500.22	16.020	0.95217	0.061531	3.6613
Fit 2	1.7030	500.26	16.577	0.99350	0.079972	3.5599
Fit 3	1,7030	500.26	16.923	0.99350	0.095574	3.8777
Fit 4	1.7208	500.24	16.461	0.97718	0.088558	3.6359
Hay-signal	1.7036	500.56	19.107	0.99563	0.32438	4.6672

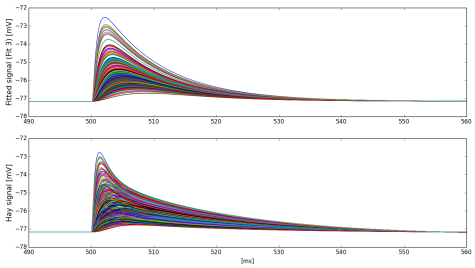
Table 9. Statistical analysis for the EPSPs generated in the stem by the Hay-model and the point model's recreation using parameters given by the sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the stem. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2. The statistical values investigated are explained in the beginning of this section.



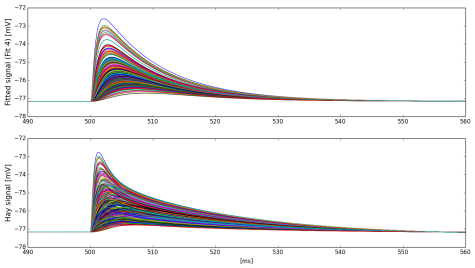
(a) Results basal, Fit 1



(b) Results basal, Fit 2



(c) Results basal, Fit 3

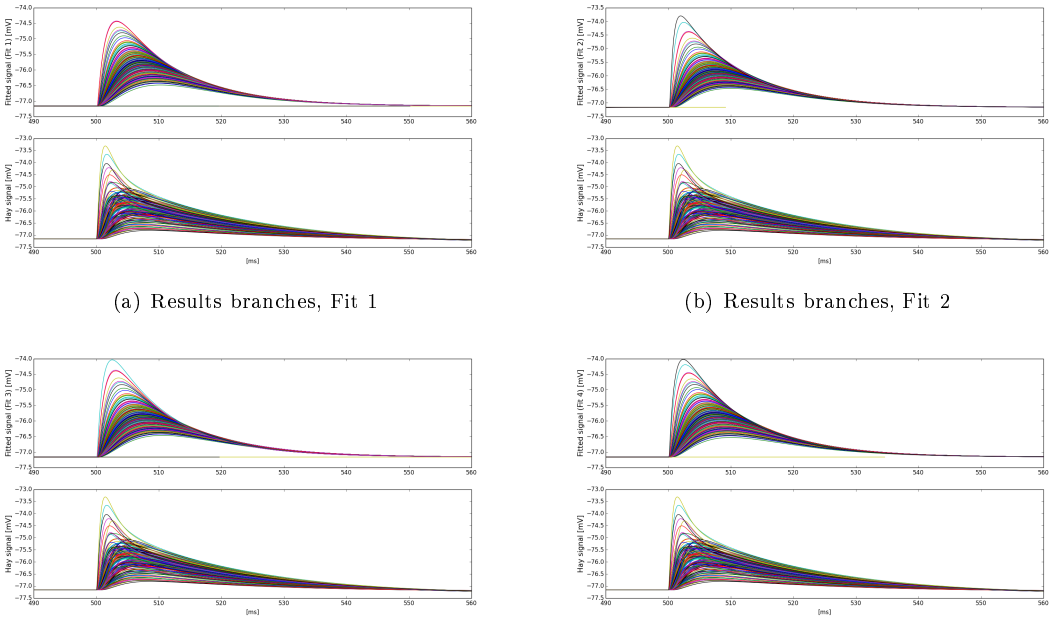


(d) Results basal, Fit 4

Figure 29. The figure shows the point model's recreation of the Hay model's EPSPs that were generated in the basal. The sigmoid result functions give synapse parameters to recreate an EPSP from the Hay-model when evaluated for the distance from soma the given EPSP was generated. Equation 58 on page 30 gives the point model's response when evaluated with synapse parameters obtained by the result sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the basal. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2.

	μ amp	μ del	$\mu\Delta$time vmax-halfs	sd amp	sd del	sdΔ time vmax-halfs
Fit 1	1.6924	500.16	12.028	0.78856	0.015648	1.6221
Fit 2	1.6433	500.19	12.346	0.80580	0.068552	1.7262
Fit 3	1.6433	500.19	12.578	0.80580	0.020883	1.7676
Fit 4	1.6556	500.18	12.385	0.80001	0.043068	1.7471
Hay-signal	1.6399	500.34	13.760	0.89542	0.16188	2.4560

Table 10. Statistical analysis for the EPSPs generated in the basal by the Hay-model and the point model's recreation using parameters given by the sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the basal. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2. The statistical values investigated are explained in the beginning of this section.



(a) Results branches, Fit 1

(b) Results branches, Fit 2

(c) Results branches, Fit 3

(d) Results branches, Fit 4

Figure 30. The figure shows the point model's recreation of the Hay model's EPSPs that were generated in the branches. The sigmoid result functions give synapse parameters to recreate an EPSP from the Hay-model when evaluated for the distance from soma the given EPSP was generated. Equation 58 on page 30 gives the point model's response when evaluated with synapse parameters obtained by the result sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the branches. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2.

	μ amp	μ del	$\mu\Delta$time vmax-halfs	sd amp	sd del	sdΔ time vmax-halfs
Fit 1	1.4350	500.20	16.565	0.48204	0.045664	8.1117
Fit 2	1.3830	500.28	16.250	0.50131	0.11497	2.1149
Fit 3	1.3830	500.24	17.082	0.50131	0.050599	5.4243
Fit 4	1.3974	500.23	16.249	0.48885	0.080164	2.5325
Hay-signal	1.3764	500.59	18.253	0.57487	0.24268	2.6610

Table 11. Statistical analysis for the EPSPs generated in the branches by the Hay-model and the point model's recreation using parameters given by the sigmoid functions. The different fits (Fit 1, 2 ,3 and 4) arise from the different fitting methods for synapse parameters in section 6.2. Sigmoid result functions were fitted to the synapse parameters obtained by the four fitting methods, giving four sets of result sigmoid function for the branches. Each set of sigmoid result functions contains functions describing the synapse parameters, A, t_0, τ_1 and τ_2. The statistical values investigated are explained in the beginning of this section.

To analyse how well each fitting method did their job, I calculated the percentage deviation for the point model's statistics from the Hay-models statistics from tables 8, 9, 10 and 11, using equation 75. The percentage deviations were calculated for all the statistical values I investigate, in each part of the neuron and for all four fitting methods. The results are summarized in table 12.

$$p_x = \frac{x_{point} - x_{Hay}}{x_{Hay}} \cdot 100\% \tag{75}$$

x is one of the statistical values I investigate, x_{point} is the statistical value for the point model, x_{Hay} is the statistical value for the Hay-model. p_x is the percentage deviation for the point model's statistics from the Hay-models statistics for the statistical value x, for a given part of the neuron and fitting method.

The percentage deviations in table 12 are difficult to analyse, because of the amount of values for each part of the neuron. I wanted statistical values for each fitting method, independent of the parts of the neuron.

To get rid of the dependence on which part of the neuron the synapse was placed for the statistical values in table 12, I weighted them using equation 76.

Table 13 shows the percentage deviation for the statistical values for each fitting method, independent of part of the neuron. The weighted error estimates makes comparing the different fitting methods easier and more fair. More fair, because statistics calculated with more data (more synapses) is more credible than statistics calculated with less data(less synapses).

The last column in table 13 is an error estimate for the statistics for each fitting method. It was estimated by summing the values for each fitting method in the same table. The error estimates are easy to compare and make the choice of final result functions from one of the fitting methods easier.

$$p_x^w = \frac{p_x^{Apical} \cdot 180 + p_x^{Stem} \cdot 38 + p_x^{Basal} \cdot 263 + p_x^{Branches} \cdot 161}{642} \tag{76}$$

p_x is the percentage deviation for a statistical value for a given fitting method, from one of the four parts of the neuron. p_x^{apical} is the percentage deviation for a statistical value, for a given fitting method in the apical, etc. p_x^w is the percentage deviation for a given statistical value for a given fitting method, for the whole neuron as one (independent of which part of the neuron the synapses were placed). The Apical has 180 synapses, the stem has 38, the basal has 263 and the branches has 161, which makes a total of 642 synapses.

		μ A	μ t_0	μ Δ_t	sd A	sd t_0	sd Δ_t
Apical	Fit 1	2.42	0.0198	9.16	10.5	6.97	35.2
	Fit 2	0.105	0.0397	6.55	21.8	17.9	85.33
	Fit 3	0.105	0.0595	1.79	21.8	22.6	70.1
	Fit 4	0.823	0.0437	5.84	20.1	15.9	62.3
Stem	Fit 1	2.73	0.0679	16.2	4.37	81.0	21.6
	Fit 2	0.0352	0.0599	13.2	0.214	75.3	23.7
	Fit 3	0.0352	0.0599	11.4	0.214	70.5	16.9
	Fit 4	1.01	0.0639	13.8	1.85	72.7	22.1
Basal	Fit 1	3.20	0.0360	12.6	11.9	90.3	34.0
	Fit 2	0.207	0.0300	10.3	10.0	57.7	29.7
	Fit 3	0.207	0.0300	8.59	10.0	87.1	28.0
	Fit 4	0.957	0.0320	10.0	10.7	73.4	28.9
Branches	Fit 1	4.26	0.0779	9.25	16.1	81.2	205
	Fit 2	0.480	0.0619	11.0	12.8	52.6	20.5
	Fit 3	0.480	0.0699	6.42	12.8	79.1	104
	Fit 4	1.53	0.0719	11.0	15.0	67.0	4.83

Table 12. The percentage deviations for the point model's statistics from the Hay-model's statistics. With values from tables 8, 9, 10 and 11 I calculated the percentage deviations using equation 75.

	μ A	μ t_0	μ Δ_t	sd A	sd t_0	sd Δ_t	Error esimate
Fit 1	3.22	0.0439	11.0	12.1	64.1	76.5	167
Fit 2	0.237	0.0425	9.60	13.4	46.3	42.6	112
Fit 3	0.237	0.0500	6.31	13.4	66.0	58.2	145
Fit 4	1.07	0.0472	9.31	13.9	55.6	31.8	111

Table 13. The weighted percentage deviations for each statistical value for the four fitting methods, independent on the different parts of the Hay-neuron. Each statistical value for a given fitting method were weighted by the fraction of synapses, the part it came from contained and the summed, using function 76. The column Error estimates is the sums of rows and gives error estimates for each fitting method.

7.6.1 The resulting synapse models

As can be seen in table 13, fitting method 4 has the lowest error estimate, when recreating the Hay-model's EPSPs. It was naturally to choose the sigmoid functions arising from fitting method 4 to be my main result.

The result equations for fitting method 4, for the four parts of the Hay-models morphology was my main synapse model. The 16 equations arise from inserting the fitted sigmoid function parameters in table 17 on page 60 into the sigmoid function, described in equation 72 on page 40. The 16 sigmoid functions describing the synapse parameters for this synapse model are summarized in figure 31.

In section 7.4 on page 38 I fitted synapse parameters for the point model without splitting the neuron into four parts. Because I have decided that fitting method 4 gives the best results, the sigmoid functions, describing synapse parameters for the whole neuron arise from synapse parameters fitted with fitting method 4.

The result was four sigmoid functions arising from inserting fitted function parameters for the sigmoid function in table 19 on page 61 into the sigmoid function, described in equation 72 on page 40.
The four sigmoid equations describing the synapse parameters for this synapse model are summarized in figure 32

The equations for the apical are valid for d ϵ [637-1291μm]

$$A^{apical}(d) = -0.48486 \cdot \frac{6.0111}{1.5793 + e^{0.0014006(d+299.62)}} \tag{77}$$

$$t_0^{apical}(d) = 500.18 \cdot \frac{0.93106}{0.14311 + e^{-0.0082072(d-787.04)}} \tag{78}$$

$$\tau_2^{apical}(d) = 0.30105 \cdot \frac{0.24439}{0.050617 + e^{-0.011020(d-398.92)}} \tag{79}$$

$$\tau_1^{apical}(d) = 0.013853 \cdot \frac{-212.83}{-38.124 + e^{-0.010239(d-825.02)}} \tag{80}$$

The equations for the stem are valid for d ϵ [8-617μm]

$$A^{stem}(d) = -0.0018240 \cdot \frac{0.48596}{-0.42209 + e^{0.0012222(d-527.25)}} \tag{81}$$

$$t_0^{stem}(d) = 499.98 \cdot \frac{1.1811}{-0.18931 + e^{-0.0018811(d-1159.7)}} \tag{82}$$

$$\tau_2^{stem}(d) = 0.026216 \cdot \frac{3.7897}{0.10165 + e^{-0.0054307(d-591.62)}} \tag{83}$$

$$\tau_1^{stem}(d) = 0.64334 \cdot \frac{26.072}{5.0104 + e^{-0.017599(d-285.02)}} \tag{84}$$

The equations for the basal are valid for d ϵ [0-272μm]

$$A^{basal}(d) = -0.11904 \cdot \frac{0.42072}{-0.31757 + e^{0.0034918(d-257.05)}} \tag{85}$$

$$t_0^{basal}(d) = 499.91 \cdot \frac{1.8872}{0.35925 + e^{-0.0029808(d-744.21)}} \tag{86}$$

$$\tau_2^{basal}(d) = -0.38962 \cdot \frac{6.2114}{-1.0814 + e^{-0.0046929(d-546.46)}} \tag{87}$$

$$\tau_1^{basal}(d) = -0.15656 \cdot \frac{21.808}{5.0073 + e^{-0.015379(d-200.11)}} \tag{88}$$

The equations for the branches are valid for d ϵ [15-478μm]

$$A^{branch}(d) = -0.55906 \cdot \frac{0.58890}{-0.35605 + e^{0.0011595(d-616.94)}} \tag{89}$$

$$t_0^{branch}(d) = 500.15 \cdot \frac{0.78493}{0.13001 + e^{-0.010582(d-521.31)}} \tag{90}$$

$$\tau_2^{branch}(d) = -0.22803 \cdot \frac{1.5846}{0.82517 + e^{-0.012222(d-185.19)}} \tag{91}$$

$$\tau_1^{branch}(d) = -0.58806 \cdot \frac{29.237}{4.4077 + e^{-0.010437(d-311.47)}} \tag{92}$$

Figure 31. Functions for my main synapse model. The synapse model consists of 16 sigmoid equations, dependent on a single variable d, representing the somatic distance from soma from the Hay-model. The 16 sigmoid functions are sets of four, describing the synapse parameters (A, t_0, τ_1 and τ_2), for each of the four parts of the neuron (apical, stem, basal and branches). The four functions for a given part of the neuron, evaluated for a given somatic distance from the Hay-model give four synapse parameters (A, t_0, τ_1 and τ_2) that when implemented in equation 58 on page 30 give the point model's recreation of the Hay-model's EPSP generated at the given somatic distance, in the given part of the neuron.

The equations are valid for d ϵ [0-1291μm]

$$A(d) = 0.3113 \cdot \frac{0.28919}{-0.56287 + e^{0.0017692(d-255.59)}} \tag{93}$$

$$t_0(d) = 499.93 \cdot \frac{1.8466}{-0.070623 + e^{-0.0027474(d-876.11)}} \tag{94}$$

$$\tau_2(d) = -0.23480 \cdot \frac{2.8125}{0.49507 + e^{-0.0045088(d-378.75)}} \tag{95}$$

$$\tau_1(d) = -0.11874 \cdot \frac{51.202}{8.7795 + e^{-0.010842(d-354.09)}} \tag{96}$$

Figure 32. Functions for the easily implemented and applied synapse model. The synapse model consists of four sigmoid equations, dependent on a single variable d, representing the somatic distance from the Hay-model. The four functions evaluated for a given somatic distance give four synapse parameters (A, t_0, τ_1 and τ_2) that when implemented in equation 58 on page 30 give the point model's recreation of the Hay-model's EPSP generated at the given somatic distance.

8 Conclusion

I presented two synapse models for a passive point neuron, able to model the sub-threshold dynamics from an active and spacious morphology model, achieving results previously limited to the realm of complex structured models. One synapse model was more accurate and the other easier implemented and applied.

My two synapse models contained fitted sigmoid functions, that when evaluated for a distance from soma, gave synapse parameters for a current based synapse for the point model and made it able to recreate the EPSP actually generated at the given distance from soma by the Hay-model.

The synapse parameters, obtained by using one of the two synapse models gave the point model's EPSP when implemented in equation 58 on page 30.

My first and main synapse model is presented in figure 31 on page 53. This synapse model consisted of 16 sigmoid equations, describing four synapse parameters (A, t_0, τ_1 and τ_2) for each of four parts of the neuron (apical, stem, basal and branches).

To recreate an EPSP from the Hay-model in the point model, one had to use the synapse index for the synapse placement on the Hay-model, to classify which part of the neuron it lied in. When the synapse placement was classified one used the four corresponding sigmoid equations and evaluated them with the distance from soma the given EPSP was generated.

My second synapse model is presented in figure 32 on page 54. This synapse model was easier implemented and applied and consists of four sigmoid equations, describing the four synapse parameters (A, t_0, τ_1 and τ_2) for the point model. The four equations were dependent on distance from soma, but independent of which part of the neuron the Hay-model's EPSPs were generated.

To recreate an EPSP from the Hay-model in the point model, one evaluated the four sigmoid functions with the distance from soma the given EPSP was generated.

My work also lead to an analytical solution for an RC-neuron's (point neuron's) potential response due to a synapse input, with a current based synapse with conductance modelled by the beta function. The derived equation is presented in equation 57 on page 29. The derived equation might be useful for others working with point model's synaptic responses.

9 Discussion

9.1 Limitations with the simplified model

One limitation with my synapse model is that I operated below threshold and only looked at single EPSPs and no action potential was fired by the Hay-model. The current based synapse model could be used on a point model with a spiking mechanism, ex. an integrate-and-fire model, but the drastic rise in potential in the Hay-model when firing an action potential would initiate its active properties fully and the passive point model would get into big problems trying to recreate this.

One possible error source is that the synapse parameters originally were fitted in sets of four, in section 7.3 on page 34. This sets of four synapse parameters were fitted together to create a synapse for the point model, able to recreate an EPSP for a given synapse placement in the Hay-model. When I fitted the sigmoid function for the synapse parameters, in section 7.5 on page 40, the four synapse parameters got split and fitted separately. A fit of one EPSP from the Hay-model that gave a too big value for one of the synapse parameters, might correspond to another synapse parameter given a too big or too small value within the same set of fitted synapse parameters. It could therefore have been of interest to investigate the correlation between the synapse parameters in each set of four, fitted together. This would give a clearer picture of how one outlier value for one synapse parameter in a given set of four synapse parameters, corresponded to outlier values for the other synapse parameters arising from the same fit.

When looking at the fitted parameters in figure 20 on page 35, I see that the delay for the stem, branches and basal had a lot of delays with the value 500. This is clearly not right. I expect a clear increasing trend with the distance from soma. The delays lying around 500 ms, give the sigmoid functions problems fitting the right trend, and the sigmoid functions for delay probably gives too low values for delay. The error for given fits, producing delays with a too low value, might be reflected in the other synapse parameters from the same fits, giving tau rise and tau delay to big values, because the signal starts too early and gains some extra ms.

Another error source is the morphology of the Hay-model. The branched structure of the dendrites is not uniform but unsystematic. How much an EPSP from the Hay-model gets flattened and weakened while propagating to the soma depends on how much spread and weakening it experiences. A signal travelling across more branched structure will encounter more spread and get more flattened. The point model recoups for this in its fitted parameter, but the trend will be deranged. A morphology with no branching would probably give a clearer trend in the fitted parameters, but this would make the project less realistic and the results less useful in a network simulation of realistic neurons.

9.2 Current based vs conductance based synapses

The Hay-model uses a conductance based synapse and the point model uses a current based synapses. The synapses in both models have time course modelled by the beta function(two-exponential function).

There are reasons why the choice of synapse type for the point model should not be a big problem. The amount of current used as synaptic input is low. The membrane potential V(t) does not change too much throughout a simulation and the extra factor for the conductance based synapse $E_{syn} - V(t)$ will not give drastic changes to the synaptic current .

I looked at single EPSPs and the model never had to account for more than one signal at a time. A realistic neuron will get synaptic input to different synapses constantly, but my synapse model is not suitable for integration of more than one signal at a time.

The conductance based and current based synapse work differently when it comes to signal integration. The additive properties in a current based synapse has no ceiling level for integrating synaptic inputs. If more than one synaptic input arrive in a current based synapse, the synaptic current will be the sum of the two synaptic currents. The conductance based synapse has a different behavior The factor $E_{syn} - V(t)$

makes the synaptic current voltage-dependent, as the membrane potential increase, the synaptic current decrease, because the factor $E_{syn} - V(t)$ decrease. If the membrane potential V(t) reaches the reversal potential of the synapse E_{syn}, the synaptic current is zero. The synaptic current can even be negative, if the membrane potential exceeds the reversal potential, causing ions to flow against a decrease in membrane potential. In my case the reversal potentials for the synapses are zero, and the peaks of the EPSPs are not near this value.

The changes in morphology from the spatial Hay-neuron to the point model and the transition from an active to a passive model, are drastic simplifications. To give the two models the same type of synapse, would surely make their synapses integrate signals in a more similar way, but because the two models have such different natures it would probably not have a large impact when it comes to making the point model recreate the somatic responses from the Hay-model.

The main reason why I chose a current based synapse for the point model was because it is impossible to derive an analytical solution for the point model's synaptic response with a conductance based synapse, because of its dependence of the membrane potential V(t). One would have to use numerical solutions and this was not my intention for this project.

9.3 Future prospects

It could be of interest to give the point model an active ion channel and see if it more easily could fit the Hay-models EPSPs. If the point model had an active ion channel, it could probably account for more of the Hay-model's active nature and give better fits against its EPSPs. Better fits against the Hay-models EPSPs would probably give clearer trends in the fitted parameters, better sigmoid functions for the parameters and better statistical results for the synapse models.

It would be interesting to expand this study to a case with more than one synapse receiving synaptic input at a time, this would be a more realistic project, but the signal integrating process would probably be far more complicated.

A better choice for the different fitting methods (Fit 1, 2, 3 and 4) would have been to choose the synapse parameters from the fitting method that gave the lowest error estimate for each individual EPSP fitted. This would give synapse parameters for the point model with lower errors than any of the four fitting methods I used did. It would also make the amount of fitted parameters and sigmoid functions smaller and the thesis more readable. This choice of fitted parameters was something I thought of at a late stage in the process of this thesis, when I had no time to redo my work.

References

[1] Willem AM Wybo, Klaus M Stiefel, and Benjamin Torben-Nielsen. The green's function formalism as a bridge between single-and multi-compartmental modeling. *Biological cybernetics*, 107(6):685–694, 2013.

[2] Etay Hay, Sean Hill, Felix Schürmann, Henry Markram, and Idan Segev. Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. *PLoS computational biology*, 7(7):e1002107, 2011.

[3] National Geographic. Brain. `http://science.nationalgeographic.com/science/health-and-human-body/human-body/brain-article/`, 2015.

[4] Tanya Lewis. Human brain: Facts, anatomy & mapping project. `http://www.livescience.com/29365-human-brain.html`, 2014.

[5] David Sterratt, Bruce Graham, Andrew Gillies, and David Willshaw. *Principles of computational modelling in neuroscience*. Cambridge University Press, 2011.

[6] Katie M Dabrowski, Diego J Castaño, and Jaime L Tartar. Basic neuron model electrical equivalent circuit: An undergraduate laboratory exercise. *Journal of Undergraduate Neuroscience Education*, 12(1):A49, 2013.

[7] Arnd Roth and Mark CW van Rossum. 6modeling synapses. 2009.

[8] Paul A Tipler and Gene Mosca. *Tipler*. WH Freeman & Co., 2008.

[9] Alain Destexhe, Michael Rudolph, and Denis Paré. The high-conductance state of neocortical neurons in vivo. *Nature reviews neuroscience*, 4(9):739–751, 2003.

[10] Matthew Larkum. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends in neurosciences*, 36(3):141–151, 2013.

[11] Nelson Spruston. Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience*, 9(3):206–221, 2008.

[12] Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

[13] Michael L Hines, Andrew P Davison, and Eilif Muller. Neuron and python. *Frontiers in neuroinformatics*, 3, 2009.

[14] Henrik Lindén, Espen Hagen, Szymon Łęski, Eivind S Norheim, Klas H Pettersen, and Gaute T Einevoll. Lfpy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Frontiers in neuroinformatics*, 7, 2013.

[15] Stephen G Nash. A survey of truncated-newton methods. *Journal of Computational and Applied Mathematics*, 124(1):45–59, 2000.

[16] The Scipy community. Scipy.org. `http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize`, 2008.

[17] Erwin Kreyszig. *Advanced engineering mathematics*. John Wiley & Sons, 2006.

[18] Vaneeda Allken, Joy-Loi Chepkoech, Gaute T Einevoll, and Geir Halnes. The subcellular distribution of t-type ca2+ channels in interneurons of the lateral geniculate nucleus. *PloS one*, 9(9):e107780, 2014.

[19] Gunnar G Løvås. *Statistikk for universiteter og høgskoler*. Universitetsforlaget, 2010.

10 Appendix A

10.1 Results for the four parts of the neuron for the four fitting methods

Part of neuron	Synapse parameter	x_0	x_1	x_2	x_3	x_4
apical	amplitude	$-4.0004 \cdot 10^{-1}$	5.9998	1.4000	$-1.6514 \cdot 10^{-3}$	$-1.9999 \cdot 10^{2}$
	tau rise	$5.3524 \cdot 10^{-1}$	$2.3361 \cdot 10^{-1}$	$5.4739 \cdot 10^{-2}$	$9.1208 \cdot 10^{-3}$	$3.7914 \cdot 10^{2}$
	tau decay	$9.8459 \cdot 10^{-5}$	$-2.0690 \cdot 10^{2}$	$-3.9584 \cdot 10^{1}$	$-2.0086 \cdot 10^{-3}$	$7.9076 \cdot 10^{2}$
	delay	$5.0054 \cdot 10^{2}$	1.0098	$1.6074 \cdot 10^{-1}$	$7.6944 \cdot 10^{-3}$	$7.9862 \cdot 10^{2}$
stem	amplitude	$-6.5074 \cdot 10^{-2}$	$5.4585 \cdot 10^{-1}$	$-3.7924 \cdot 10^{-1}$	$-1.2644 \cdot 10^{-3}$	$5.5158 \cdot 10^{2}$
	tau rise	$6.3451 \cdot 10^{-2}$	$2.3290 \cdot 10^{1}$	-1.4398	$5.9972 \cdot 10^{-3}$	$8.8485 \cdot 10^{2}$
	tau decay	$5.9254 \cdot 10^{-1}$	$2.7288 \cdot 10^{1}$	6.3606	$1.8472 \cdot 10^{-2}$	$2.7466 \cdot 10^{2}$
	delay	$4.9999 \cdot 10^{2}$	$6.8544 \cdot 10^{-1}$	$1.5461 \cdot 10^{-1}$	$1.6820 \cdot 10^{-3}$	$9.6202 \cdot 10^{2}$
basal	amplitude	$-7.2406 \cdot 10^{-2}$	1.3072	$-4.8237 \cdot 10^{-1}$	$-5.0032 \cdot 10^{-3}$	$4.9839 \cdot 10^{1}$
	tau rise	$-5.0391 \cdot 10^{-1}$	6.5422	-1.2984	$4.1649 \cdot 10^{-3}$	$5.9360 \cdot 10^{2}$
	tau decay	$-8.1854 \cdot 10^{-1}$	$3.0162 \cdot 10^{1}$	4.1417	$8.6399 \cdot 10^{-3}$	$3.2281 \cdot 10^{2}$
	delay	$5.0010 \cdot 10^{2}$	$5.2698 \cdot 10^{-1}$	$2.5031 \cdot 10^{-1}$	$4.3847 \cdot 10^{-3}$	$5.8518 \cdot 10^{2}$
branches	amplitude	$-5.2025 \cdot 10^{-1}$	$7.3073 \cdot 10^{-1}$	$-4.7918 \cdot 10^{-1}$	$1.0940 \cdot 10^{-3}$	$4.0016 \cdot 10^{2}$
	tau rise	$-4.0477 \cdot 10^{-1}$	1.3479	$5.0302 \cdot 10^{-1}$	$1.0913 \cdot 10^{-2}$	$1.6055 \cdot 10^{2}$
	tau decay	$-9.4981 \cdot 10^{-1}$	$2.7749 \cdot 10^{1}$	4.2260	$8.7457 \cdot 10^{-3}$	$3.1832 \cdot 10^{2}$
	delay	$5.0018 \cdot 10^{2}$	$8.4436 \cdot 10^{-1}$	$1.0069 \cdot 10^{-1}$	$2.6787 \cdot 10^{-2}$	$5.0354 \cdot 10^{2}$

Table 14. The sigmoid function's parameters found by curve fitting in section 6.3. The parameters for the sigmoid functions were fitted against the sets of synapse parameters obtained by fitting method 1.

Part of neuron	Synapse parameter	x_0	x_1	x_2	x_3	x_4
apical	amplitude	$-3.9929 \cdot 10^{-1}$	5.7733	1.9375	$-1.6046 \cdot 10^{-3}$	$-1.6319 \cdot 10^{2}$
	tau rise	$4.0603 \cdot 10^{-2}$	$2.3324 \cdot 10^{-1}$	$4.7543 \cdot 10^{-2}$	$1.2465 \cdot 10^{-2}$	$3.8298 \cdot 10^{2}$
	tau decay	$-1.6226 \cdot 10^{-1}$	$-2.1300 \cdot 10^{2}$	$-3.7658 \cdot 10^{1}$	$1.1615 \cdot 10^{-2}$	$8.3227 \cdot 10^{-2}$
	delay	$4.9906 \cdot 10^{2}$	$9.5729 \cdot 10^{-1}$	$1.0937 \cdot 10^{-1}$	$5.4785 \cdot 10^{-3}$	$6.1544 \cdot 10^{2}$
stem	amplitude	$1.6452 \cdot 10^{-2}$	$4.6979 \cdot 10^{-1}$	$-4.1634 \cdot 10^{-1}$	$-1.2643 \cdot 10^{-3}$	$5.2585 \cdot 10^{2}$
	tau rise	$5.3069 \cdot 10^{-1}$	2.6298	$-9.9640 \cdot 10^{-1}$	$7.2160 \cdot 10^{-3}$	$7.2218 \cdot 10^{2}$
	tau decay	$7.2079 \cdot 10^{-2}$	$2.5558 \cdot 10^{1}$	3.8879	$1.2211 \cdot 10^{-2}$	$3.0775 \cdot 10^{2}$
	delay	$4.9998 \cdot 10^{2}$	1.0459	$-8.2243 \cdot 10^{-2}$	$1.6065 \cdot 10^{-3}$	$1.1599 \cdot 10^{3}$
basal	amplitude	$-1.6440 \cdot 10^{-1}$	$3.0588 \cdot 10^{-1}$	$-3.0090 \cdot 10^{-1}$	$-2.8905 \cdot 10^{-3}$	$3.4892 \cdot 10^{2}$
	tau rise	$-2.2643 \cdot 10^{-1}$	4.0752	$4.1062 \cdot 10^{-1}$	$6.8677 \cdot 10^{-3}$	$3.3481 \cdot 10^{2}$
	tau decay	-1.1360	$2.7062 \cdot 10^{1}$	2.9018	$6.9745 \cdot 10^{3}$	$3.4966 \cdot 10^{2}$
	delay	$5.0007 \cdot 10^{2}$	3.7527	$-1.3019 \cdot 10^{-2}$	$8.9794 \cdot 10^{-3}$	$5.0007 \cdot 10^{2}$
branches	amplitude	$-7.7428 \cdot 10^{-2}$	$3.7204 \cdot 10^{-1}$	$-3.1389 \cdot 10^{-1}$	$-1.4603 \cdot 10^{-3}$	$6.3563 \cdot 10^{2}$
	tau rise	$-5.5810 \cdot 10^{-1}$	1.9079	$8.9404 \cdot 10^{-1}$	$8.1302 \cdot 10^{-3}$	$1.3272 \cdot 10^{2}$
	tau decay	-1.8451	$1.8240 \cdot 10^{1}$	2.2927	$9.7990 \cdot 10^{-3}$	$2.1450 \cdot 10^{2}$
	delay	$5.0018 \cdot 10^{2}$	1.5307	$9.1644 \cdot 10^{-1}$	$1.3585 \cdot 10^{-2}$	$5.0894 \cdot 10^{2}$

Table 15. The sigmoid function's parameters found by curve fitting in section 6.3. The parameters for the sigmoid functions were fitted against the sets of synapse parameters obtained by fitting method 2.

Part of neuron	Synapse parameter	x_0	x_1	x_2	x_3	x_4
apical	amplitude	$-3.9929 \cdot 10^{-1}$	5.7733	1.9375	$-1.6046 \cdot 10^{-3}$	$-1.6319 \cdot 10^{2}$
	tau rise	-2.4204	$1.9275 \cdot 10^{-1}$	$2.3977 \cdot 10^{-2}$	$1.3532 \cdot 10^{-2}$	$3.6284 \cdot 10^{2}$
	tau decay	$-2.9650 \cdot 10^{-1}$	$-2.2262 \cdot 10^{2}$	$-3.6402 \cdot 10^{1}$	$1.2006 \cdot 10^{-2}$	$8.4599 \cdot 10^{2}$
	delay	$4.9994 \cdot 10^{2}$	$9.7177 \cdot 10^{-1}$	$1.3852 \cdot 10^{-1}$	$8.0153 \cdot 10^{-3}$	$7.8945 \cdot 10^{2}$
stem	amplitude	$1.6452 \cdot 10^{-2}$	$4.6980 \cdot 10^{-1}$	$-4.1634 \cdot 10^{-1}$	$-1.2643 \cdot 10^{-3}$	$5.2585 \cdot 10^{2}$
	tau rise	$-1.3609 \cdot 10^{-1}$	5.0957	$-4.3513 \cdot 10^{-1}$	$3.3503 \cdot 10^{-3}$	$8.0161 \cdot 10^{2}$
	tau decay	$6.0034 \cdot 10^{-1}$	$2.7201 \cdot 10^{1}$	4.6837	$1.8186 \cdot 10^{-2}$	$2.8190 \cdot 10^{2}$
	delay	$4.9985 \cdot 10^{2}$	1.1082	$-1.7228 \cdot 10^{-1}$	$1.2653 \cdot 10^{-3}$	$1.1599 \cdot 10^{3}$
basal	amplitude	$-1.6440 \cdot 10^{-1}$	$3.0588 \cdot 10^{-1}$	$-3.0090 \cdot 10^{-1}$	$-2.8905 \cdot 10^{-3}$	$3.4892 \cdot 10^{2}$
	tau rise	$-2.9830 \cdot 10^{-1}$	5.3667	-1.2394	$4.9279 \cdot 10^{-3}$	$5.2863 \cdot 10^{2}$
	tau decay	$-5.9021 \cdot 10^{-1}$	$.2.4637 \cdot 10^{1}$	4.8804	$1.3667 \cdot 10^{-2}$	$1.9907 \cdot 10^{2}$
	delay	$5.0467 \cdot 10^{2}$	$-1.0172 \cdot 10^{1}$	2.0115	$-7.4462 \cdot 10^{-4}$	$1.9300 \cdot 10^{3}$
branches	amplitude	$-7.7428 \cdot 10^{-2}$	$3.7204 \cdot 10^{-1}$	$-3.1389 \cdot 10^{-1}$	$-1.4603 \cdot 10^{-3}$	$6.3563 \cdot 10^{2}$
	tau rise	-1.6930	1.7275	$-1.3778 \cdot 10^{-1}$	$1.5338 \cdot 10^{-3}$	$1.5017 \cdot 10^{-2}$
	tau decay	$-2.4204 \cdot 10^{-1}$	$2.6692 \cdot 10^{1}$	3.9767	$1.2490 \cdot 10^{-2}$	$2.9272 \cdot 10^{2}$
	delay	$5.0002 \cdot 10^{2}$	$8.8341 \cdot 10^{-1}$	$9.9672 \cdot 10^{-2}$	$2.6399 \cdot 10^{-3}$	$7.9996 \cdot 10^{2}$

Table 16. The sigmoid function's parameters found by curve fitting in section 6.3. The parameters for the sigmoid functions were fitted against the sets of synapse parameters obtained by fitting method 3.

Part of neuron	Synapse parameter	x_0	x_1	x_2	x_3	x_4
apical	amplitude	$-4.8486 \cdot 10^{-1}$	6.0111	1.5793	$-1.4006 \cdot 10^{-3}$	$-2.9962 \cdot 10^{2}$
	tau rise	$3.0105 \cdot 10^{-1}$	$2.4439 \cdot 10^{-1}$	$5.0617 \cdot 10^{-2}$	$1.1020 \cdot 10^{-2}$	$3.9892 \cdot 10^{2}$
	tau decay	$1.3853 \cdot 10^{-2}$	$-2.1283 \cdot 10^{2}$	$-3.8124 \cdot 10^{1}$	$1.0239 \cdot 10^{-2}$	$8.2563 \cdot 10^{2}$
	delay	$5.0018 \cdot 10^{2}$	$9.3106 \cdot 10^{-1}$	$1.4311 \cdot 10^{-1}$	$8.2072 \cdot 10^{-3}$	$7.8704 \cdot 10^{2}$
stem	amplitude	$-1.8240 \cdot 10^{-3}$	$4.8596 \cdot 10^{-1}$	$-4.2209 \cdot 10^{-1}$	$-1.2222 \cdot 10^{-3}$	$5.2725 \cdot 10^{2}$
	tau rise	$2.6216 \cdot 10^{-2}$	3.7897	$1.0165 \cdot 10^{-1}$	$5.4307 \cdot 10^{-3}$	$5.9162 \cdot 10^{2}$
	tau decay	$6.4334 \cdot 10^{-1}$	$2.6072 \cdot 10^{1}$	5.0104	$1.7599 \cdot 10^{-2}$	$2.8502 \cdot 10^{2}$
	delay	$4.9998 \cdot 10^{2}$	1.1811	$-1.8931 \cdot 10^{-1}$	$1.8811 \cdot 10^{-3}$	$1.1597 \cdot 10^{3}$
basal	amplitude	$-1.1902 \cdot 10^{-1}$	$4.2072 \cdot 10^{-1}$	$-3.1757 \cdot 10^{-1}$	$-3.4918 \cdot 10^{-3}$	$2.5705 \cdot 10^{2}$
	tau rise	$-3.8962 \cdot 10^{-1}$	6.2114	-1.0814	$4.6929 \cdot 10^{-3}$	$5.4646 \cdot 10^{2}$
	tau decay	$-1.5656 \cdot 10^{-1}$	$2.1808 \cdot 10^{1}$	5.0073	$1.5379 \cdot 10^{-2}$	$2.0011 \cdot 10^{2}$
	delay	$4.9991 \cdot 10^{2}$	1.8872	$3.5925 \cdot 10^{-1}$	$2.9808 \cdot 10^{-3}$	$7.4421 \cdot 10^{2}$
branches	amplitude	$-5.5906 \cdot 10^{-1}$	$5.8890 \cdot 10^{-1}$	$-3.5605 \cdot 10^{-1}$	$-1.1595 \cdot 10^{-3}$	$6.1694 \cdot 10^{2}$
	tau rise	$-2.2803 \cdot 10^{-1}$	1.5846	$8.2517 \cdot 10^{-1}$	$1.2222 \cdot 10^{-2}$	$1.8519 \cdot 10^{2}$
	tau decay	$-5.8806 \cdot 10^{-1}$	$2.9237 \cdot 10^{1}$	4.4077	$1.0437 \cdot 10^{-2}$	$3.1147 \cdot 10^{2}$
	delay	$5.0015 \cdot 10^{2}$	$7.8439 \cdot 10^{-1}$	$1.3001 \cdot 10^{-1}$	$1.0582 \cdot 10^{-2}$	$5.2131 \cdot 10^{2}$

Table 17. The sigmoid function's parameters found by curve fitting in section 6.3. The parameters for the sigmoid functions were fitted against the sets of synapse parameters obtained by fitting method 4.

Amplitude	Delay	Tau Rise	Tau Decay
0.204	0.150	0.551	0.674

Table 18. Weighted error estimates for the sigmoid curve fitting for fitting method 4. The error estimates are taken from table 5 on page 41 and weighted by the fraction of synapses each part of the neuron contains, using equation 76 on page 50. The weighted error estimates describe the mean difference between the fitted sigmoid function and the synapse parameters for the point model in every point, independent on which part of the neuron the synapse was placed in.

10.2 Results for the whole neuron with fitting method 4

Synapse parameter	x_0	x_1	x_2	x_3	x_4
amplitude	$3.1113 \cdot 10^{-1}$	$2.8919 \cdot 10^{-1}$	$-5.6287 \cdot 10^{-1}$	$-1.7692 \cdot 10^{-3}$	$2.5559 \cdot 10^{2}$
tau rise	$-2.3480 \cdot 10^{-1}$	2.8125	$4.9507 \cdot 10^{-1}$	$4.5088 \cdot 10^{-3}$	$3.7875 \cdot 10^{2}$
tau decay	$-1.1847 \cdot 10^{-1}$	$5.1202 \cdot 10^{1}$	8.7795	$1.0842 \cdot 10^{-2}$	$3.5409 \cdot 10^{2}$
delay	$4.9993 \cdot 10^{2}$	1.8466	$-7.0623 \cdot 10^{-2}$	$2.7474 \cdot 10^{-3}$	$8.7611 \cdot 10^{2}$

Table 19. The fitted parameters for the sigmoid function, describing synapse parameters independent on where in the Hay-model's morphology the EPSPs were generated.

11 Appendix B, Python codes

This Appendix contains the most important python scripts used in this thesis.

11.1 The Hay-model

```
'''Run model by Hay et al. (2011), runs the model a given number of synapse indexes.'''
#module imports
import numpy as np
import urllib as urllib2
import zipfile
import os
import platform
import LFPy
import neuron

#interactive plotting and closing old figures
#if not plt.rcParams['interactive']:
#    plt.ion()

#close open windows
#plt.close('all')

##############################################################################
# model pre-setup
##############################################################################
modelurl = "http://senselab.med.yale.edu/ModelDB/eavBinDown.asp?o=139653&a=23"
#Fetch Hay et al. 2011 model files, if they are missing
if not os.path.isfile('L5bPCmodelsEH/morphologies/cell1.asc'):
    if platform.platform().rfind("Windows") >= 0:
        if not os.path.isfile('L5bPCmodelsEH.zip'):
            print "download model files manually from: %s" % modelurl
    else:
        #get the model files:
        u = urllib2.urlopen(modelurl)
        localFile = open('L5bPCmodelsEH.zip', 'w')
        localFile.write(u.read())
        localFile.close()
        u.close()

    #unzip:
    myzip = zipfile.ZipFile('L5bPCmodelsEH.zip', 'r')
    myzip.extractall('.')
    myzip.close()


#assuming files are present, compile NMODL language files ones
if platform.platform().rfind('Windows') >= 0:
    dllpath = os.path.join("L5bPCmodelsEH", "mod", "nrnmech.dll")
```

```
    if not os.path.isfile(dllpath):
        dllcmd = "cd L5bPCmodelsEH\\mod\\ & \\nrn73w64\\mingw\\bin
        \\sh \\nrn73w64\\lib\\mknrndll.sh \\nrn73w64\\ "
        os.system(dllcmd)
else:
    if not os.path.isdir(os.path.join("L5bPCmodelsEH", "mod", platform.machine())):
        os.system('''
              cd L5bPCmodelsEH/mod/
              nrnivmodl
              ''')

#load compiled mod-files from their separate location:
if platform.platform().rfind("Windows") >= 0:
    if dllpath in neuron.nrn_dll_loaded:
        print "%s already loaded" % dllpath
    else:
        neuron.h.nrn_load_dll(dllpath)
        neuron.nrn_dll_loaded.append(dllpath)
else:
    neuron.load_mechanisms('L5bPCmodelsEH/mod/')


##############################################################################
# Simulation parameters
##############################################################################

#define cell parameters used as input to cell-class as a python dict
cellParameters = {
    'morphology'    : 'L5bPCmodelsEH/morphologies/cell1.asc',   #morphology file
    'templatefile'  : ['L5bPCmodelsEH/models/L5PCbiophys3.hoc', #model specs
                       'L5bPCmodelsEH/models/L5PCtemplate.hoc'],# ditto
    'templatename'  : 'L5PCtemplate',                           #LFPy need this
    'templateargs'  : 'L5bPCmodelsEH/morphologies/cell1.asc',   #morphology
    'passive' : False,      #the template-files assign passive membrane params
    'nsegs_method' : None,  #ditto for compartmentalization
    'timeres_NEURON' : 2**-4,   #(ms) time resolution for simulation in NEURON
    'timeres_python' : 2**-4,   #(ms) ditto, in python namespace
    'tstartms' : 0,             #(ms) start time of simulation
    'tstopms' : 700,            #(ms) end time of simulation
    'v_init' : -75,             #(mV) assigned membrane potentials at t=0
    'pt3d' : False,             #optional bool, just for fancier plotting below
    'extracellular' : False,    #turn off calculations of membrane currents
}

#define synapse parameters used as input to synapse-class as a python dict
synapseParameters = {
    'idx' : 0,              # insert synapse on index "0", the soma, you can
                            # override this when creating the cell commenting
                            # out here, and supply any integer number on
```

```
                            # [0, cell.totnsegs), cell.totnsegs being the
                            # total number of compartments
    'e' : 0.,               # reversal potential of synapse
    'syntype' : 'Exp2Syn',  # built-in, conductance based double-exponential synapse
    'tau1' : 0.25,          # Time constant, rise
    'tau2' : 1.0,           # Time constant, decay
    'weight' : 0.01,        # Synaptic weight, units of uS
    'record_current' : True, # Will enable synapse current recording
    'record_potential' : True, # enable synapse site voltage recording
    'color' : 'r',          # just for plotting,
    'marker' : '.',         # ditto
}

##############################################################################
# Main simulation procedure, setting up cell, synapse
##############################################################################
#

#run list decides which indexes for synapses the simulation is run for
run_list=[]

#making some lists for use in other scripts
counter = 0
soma_voltage = np.zeros(shape=(len(run_list), 11201))
synapse_distance=[]
voltage_max =[]
index_tops= []
t0_idx_list=[]
time_between_vmax_halfs=[]
#sets the synapse index to values in run_list
for i in run_list:
    synapseParameters['idx']= i

#delete old sections and load compiled mechs from the mod-folder,
#needed for successive runs
    neuron.h("forall delete_section()")


#Initialize cell instance, using the LFPy.TemplateCell class, align cell
    cell = LFPy.TemplateCell(**cellParameters)
    cell.set_rotation(x = 4.729, y =-3.166)

#set up a synapse object attached to the cell, and set the input spike time(s)
    synapse = LFPy.Synapse(cell=cell, **synapseParameters)
    synapse.set_spike_times(np.array([500.]))

#perform NEURON simulation, results saved as attributes in the cell instance
#synapse current and synapse site voltage response recordings is now switched on
    cell.simulate(rec_isyn=True, rec_vmemsyn=True, rec_vmem=False)
```

```
#############################################################################################
# finding several things used in fitting
#############################################################################################
    "finding the maximum soma response, it is used in fitting"
    voltage_start=cell.somav[8000:]
    vmax=np.amax(voltage_start)

    '''finding time for maximun soma-response:'''
    index_top=(abs(voltage_start)-vmax).argmin()
    vmax=cell.somav[index_top+8000]

    "finding the index where soma potential is 3% of maximum value
    on it's way up, used as t0"
    t0_value=abs(((vmax-cell.somav[8000])*3)/100)+cell.somav[8000]

    t0_list=np.where(voltage_start>=t0_value)
    t0_idx=t0_list[0][0]+8000

    "finding time for maximun soma-response:"
    for position, item in enumerate(voltage_start):
        if item==vmax:
            index_top=position

    "time between two vmax_halfs"
    vmax_half=((vmax-cell.somav[8000])/2)+cell.somav[8000]

    index_half_up =(abs(voltage_start[:index_top]-vmax_half)).argmin()
    index_half_down =(abs(voltage_start[index_top:]-vmax_half)).argmin()

    time_between_vmax_halfs.append(abs(cell.tvec[index_half_up+8000]-
    cell.tvec[index_half_down+8000+index_top]))

    "finding the distance from the synapse to soma"
    _, secname, frac = cell.get_idx_name(idx = 0) #soma idx is 0.
    neuron.h('access %s' % secname)      #make soma the currently accessed section
    neuron.h.distance(frac) #set the root point from were we're computing the distance
    _, secname, frac = cell.get_idx_name(idx = synapse.idx)
    neuron.h('access %s' % secname) #make branch the currently accessed section
    distance_to_syn = neuron.h.distance(frac) #get the distance to synapse site.

    "values for each simulation appended to list"
    synapse_distance.append(distance_to_syn)
    voltage_max.append(vmax)
    index_tops.append(index_top)
    t0_idx_list.append(t0_idx)
    soma_voltage[counter]=cell.somav
    counter += 1
```

11.2 Fit the synapse parameters

```
'''script for fitting synapse parameters for point model from Hay-model'''

import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import Hay

"The analytical solution for the point model"
def exp2(x, t):
    '''
    difference of two exponentials function implementation
    kwargs: x : list [delay, tau_rise, tau_decay, constant, amplitude]
            t : np.ndarray, time vector of signal
    returns:  : np.array, the difference of two exponentials
    '''
    Tm=9.67
    y=np.zeros(t.size)+Hay.cell.somav[8000]
    [i] = np.where((t>500))
    t=t-x[2]
    t[t<=0]=0
    A= (x[0]*Tm)/(x[0]-Tm)
    B= (x[1]*Tm)/(x[1]-Tm)
    num=abs((A-B)*np.exp(-t[i]/Tm)-A*np.exp(-t[i]/x[0])+B*np.exp(-t[i]/x[1]))
    if i.size>0:
        denom=abs((A-B)*np.exp(-t[i]/Tm)-A*np.exp(-t[i]/x[0])+
        B*np.exp(-t[i]/x[1])).max()
        y[i]=Hay.cell.somav[8000]+x[3]*(num/denom)
        return y

"costfunction for the fitting"

def costfun(x, t, data):
    '''cost function that should be minimized'''
    return ((exp2(x, t) - data)**2).sum()

"making empty arrays for the synapse parameters and error estimates"
results1= np.zeros(shape=(Hay.counter,4))
results2= np.zeros(shape=(Hay.counter,4))
results3= np.zeros(shape=(Hay.counter,4))
error_list1= []
error_list2= []
error_list_total= []
error_list3_cut_extended=[]

for i in range(Hay.counter):
    t = Hay.cell.tvec
    data = Hay.soma_voltage[i]
    amplitude = Hay.voltage_max[i]
```

```
        t0_index=Hay.t0_idx_list[i]
        index_top= Hay.index_tops[i]+8000
        stop=np.where(data[index_top:]<=data[8000])
        stop_spot=stop[0][0]
        stop_point= stop_spot+index_top
        data3=data[:stop_point]
        t3=t[:stop_point]

        "initial guess for fitting process, [tau_decay, tau_rise, delay, amplitude]"
        x = [1, 0.25 , t[t0_index],(amplitude-data[8000]) ]

        "Fitting method 1, 2 and 3"

        res1= minimize(costfun, x0=x, args=(t[500<t], data[500<t]), method='TNC',
        bounds=((None,None),(None,None),(500, t[t0_index]),(None, None)),
        options={'disp': True})

        res2= minimize(costfun, x0=x, args=(t[500<t], data[500<t]), method='TNC',
        bounds=((None,None),(None,None),(500, t[t0_index]),(amplitude-data[8000],
        amplitude-data[8000])), options={'disp': True})

        res3= minimize(costfun, x0=x, args=(t3[500<t3], data3[500<t3]),
        method='TNC',bounds=((None,None),(None,None),(500, t[t0_index]),
        (amplitude-data[8000],amplitude-data[8000])), options={'disp': True})

        results1[i]= res1.x
        results2[i]= res2.x
        results3[i]= res3.x

        "error estimate fitting method 1"
        error1 = (np.sqrt((exp2(results1[i],t)- (data))**2))
        error1 = (error1[t>=500]).sum()
        error_list1.append(error1)

        "error estimate fitting method 2"
        error2 = (np.sqrt((exp2(results2[i],t)- (data))**2))
        error2 = (error2[t>=500]).sum()
        error_list2.append(error2)

        "error estimate fitting method 3"
        error3_cut_extended = (np.sqrt((exp2(results3[i],t)-(data))**2))
        error3_cut_extended = (error3_cut_extended[t>=500]).sum()
        error_list3_cut_extended.append(error3_cut_extended)

tau_rise_total=[]
tau_decay_total=[]
delay_total=[]
amplitude_total=[]
```

```
tau_rise1=[]
tau_decay1=[]
delay1=[]
amplitude1=[]

tau_rise2=[]
tau_decay2=[]
delay2=[]
amplitude2=[]

tau_rise3=[]
tau_decay3=[]
delay3=[]
amplitude3=[]


for i in range(len(results1)):
    tau_decay1.append(results1[i,0])
    tau_rise1.append(results1[i,1])
    delay1.append(results1[i,2])
    amplitude1.append(results1[i,3])

for i in range(len(results2)):
    tau_decay2.append(results2[i,0])
    tau_rise2.append(results2[i,1])
    delay2.append(results2[i,2])
    amplitude2.append(results2[i,3]

for i in range(len(results3)):
    tau_decay3.append(results3[i,0])
    tau_rise3.append(results3[i,1])
    delay3.append(results3[i,2])
    amplitude3.append(results3[i,3])

"Fitting method 4"
for i in range(len(results3)):
    tau_decay_total.append((tau_decay1[i]+tau_decay2[i]+tau_decay3[i])/3)
    tau_rise_total.append((tau_rise1[i]+tau_rise2[i]+tau_rise3[i])/3)
    delay_total.append((delay1[i]+delay2[i]+delay3[i])/3)
    amplitude_total.append((amplitude1[i]+amplitude2[i]+amplitude3[i])/3)

for i in range(len(Hay.run_list)):
    parameters=[0,0,0,0]
    t = Hay.cell.tvec
    data = Hay.soma_voltage[i]
    t0_index=Hay.t0_idx_list[i]
    index_top= Hay.index_tops[i]+8000
    stop=np.where(data[index_top:]<=data[8000])
    stop_spot=stop[0][0]
```

```
    stop= stop_spot+index_top
    data3=data[:stop]
    t3=t[:stop]

    parameters[0]=tau_decay_total[i]
    parameters[1]=tau_rise_total[i]
    parameters[2]=delay_total[i]
    parameters[3]=amplitude_total[i]

"error for fitting method 4"
    error_total = (np.sqrt((exp2(parameters,t)- (data))**2))
    error_total = (error_total[t>=500]).sum()
    error_list_total.append(error_total)

"plot results for fitting method 4"
    plt.plot(t[7200:10400],exp2(parameters, t)[7200:10400])
    plt.plot(t[7200:10400],data[7200:10400])
    plt.suptitle('Fit 4' , fontsize=15)
    plt.xlabel('[ms]', fontsize=15)
    plt.ylabel('[mV]', fontsize=15)
    plt.show()

"sort the lists the same way as distances from soma"
Hay.synapse_distance, tau_rise_total, tau_rise1, tau_rise2,
tau_rise3, tau_decay_total, tau_decay1, tau_decay2, tau_decay3,
delay_total, delay1, delay2, delay3, amplitude_total, amplitude1,
amplitude2, amplitude3, error_list1, error_list2, error_list_total,
error_list3_cut_extended\
 = zip(*sorted(zip(Hay.synapse_distance, tau_rise_total, tau_rise1,
tau_rise2, tau_rise3, tau_decay_total, tau_decay1, tau_decay2,
tau_decay3, delay_total, delay1, delay2, delay3, amplitude_total,
amplitude1, amplitude2, amplitude3, error_list1, error_list2,
error_list_total, error_list3_cut_extended)))

"plot the fitted parameters with distance from soma"
plt.subplot(221)
plt.plot(Hay.synapse_distance, amplitude1, 'mo')
plt.plot(Hay.synapse_distance, amplitude2, 'bo')
plt.plot(Hay.synapse_distance, amplitude3, 'ro')
plt.plot(Hay.synapse_distance, amplitude_total, 'co')
plt.ylabel('Amplitude [mV]')

plt.subplot(222)
plt.plot(Hay.synapse_distance, tau_rise1, 'mo')
plt.plot(Hay.synapse_distance, tau_rise2, 'bo')
plt.plot(Hay.synapse_distance, tau_rise3, 'ro')
plt.plot(Hay.synapse_distance, tau_rise_total, 'co')
plt.ylabel('Tau rise [ms]')
```

```
plt.subplot(223)
plt.plot(Hay.synapse_distance, tau_decay1, 'mo')
plt.plot(Hay.synapse_distance, tau_decay2, 'bo')
plt.plot(Hay.synapse_distance, tau_decay3, 'ro')
plt.plot(Hay.synapse_distance, tau_decay_total, 'co')
plt.ylabel('Tau decay [ms]')
plt.xlabel('Distance from soma [$\mu$m]')

plt.subplot(224)
plt.plot(Hay.synapse_distance, delay1, 'mo')
plt.plot(Hay.synapse_distance, delay2, 'bo')
plt.plot(Hay.synapse_distance, delay3, 'ro')
plt.plot(Hay.synapse_distance, delay_total, 'co')
plt.ylabel('Delay [ms]')
plt.xlabel('Distance from soma [$\mu$m]')
plt.show()

"plot the error estimates with distance from soma"
plt.plot(Hay.synapse_distance, error_list1, color= 'm')
plt.plot(Hay.synapse_distance, error_list2, color = 'b')
plt.plot(Hay.synapse_distance, error_list3_cut_extended, 'r')
plt.plot(Hay.synapse_distance, error_list_total, color ='c')
plt.ylabel('Error [mV]')
plt.xlabel('Distance from soma [$\mu$m]')
plt.show()

"find mean error and standard deviations for the error estimates"
mean_error1=(sum(error_list1))/len(Hay.run_list)
mean_error2=(sum(error_list2))/len(Hay.run_list)
mean_error3=(sum(error_list3_cut_extended))/len(Hay.run_list)
mean_error4=(sum(error_list_total))/len(Hay.run_list)
sd_1=0
sd_2=0
sd_3=0
sd_4=0

for i in range( len(Hay.run_list)):
    sd_1+=(mean_error1-error_list1[i])**2
    sd_2+=(mean_error2-error_list2[i])**2
    sd_3+=(mean_error3-error_list3_cut_extended[i])**2
    sd_4+=(mean_error4-error_list_total[i])**2
sd_1=(np.sqrt((sd_1)/(len(Hay.run_list))))
sd_2=(np.sqrt((sd_2)/(len(Hay.run_list))))
sd_3=(np.sqrt((sd_3)/(len(Hay.run_list))))
sd_4=(np.sqrt((sd_4)/(len(Hay.run_list))))
```

11.3 Curve fitting with sigmoid function

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import fitting

def sigmoid(x, t):
    '''
    difference of two exponentials function implementation
    kwargs: x : list/tuple/array, [a,b,c,t0]
            t : np.ndarray, time vector of signal
        returns   : sigmoid function as a list
    '''
    sigmoid=[]
    for i in range(len(t)):
        y=x[4]+(x[0]/(x[1]+np.exp(-x[2]*(t[i]-x[3]))))
        sigmoid.append(y)
    return sigmoid

"cost function for the curve fitting"
def costfun_sigmoid(x, t, data):
    '''cost function that should be minimized'''
    return ((sigmoid(x, t) - data)**2).sum()

"initial guesses for the fitting process"
x_sigmoid_amp_total = [0.2,-0.3,-0.002,250,-0.18]
x_sigmoid_amp_1     = [0.2,-0.3,-0.002, 50,-0.18]
x_sigmoid_amp_2     = [0.2,-0.3,-0.002, 350,-0.18]
x_sigmoid_amp_3     = [0.2,-0.3,-0.002, 350,-0.18]

x_sigmoid_rise_total = [3.71,0.161,0.004306, 600,0]
x_sigmoid_rise_1     = [3.71,0.161,0.004306, 600,0]
x_sigmoid_rise_2     = [3.71,0.161,0.004306, 600,0]
x_sigmoid_rise_3     = [3.71,0.161,0.004306, 850,0]

x_sigmoid_decay_total = [26.5, 4.505, 0.014918,195,0]
x_sigmoid_decay_1     = [26.5, 4.505, 0.014918,195,0]
x_sigmoid_decay_2     = [26.5, 4.505, 0.014918,195,0]
x_sigmoid_decay_3     = [26.5, 4.505, 0.014918,200,0]

x_sigmoid_delay_total= [3.7,0.09,0.0035, 500, 499]
x_sigmoid_delay_1    = [3.7,0.09,0.0035, 500, 499]
x_sigmoid_delay_2    = [3.7,0.09,0.0035, 500, 499]
x_sigmoid_delay_3    = [3.7,0.09,0.0035, 500, 499]

"data used in the fitting"
'''data 1 is parameters fitted with bouds on delay '''
data1_amplitude=np.zeros(len(fitting.amplitude1))
data1_delay=np.zeros(len(fitting.delay1))
```

```
data1_rise=np.zeros(len(fitting.tau_rise1))
data1_decay=np.zeros(len(fitting.tau_decay1))

'''data 2 is parameters fitted with bounds on delay and amplitude'''
data2_amplitude=np.zeros(len(fitting.amplitude1))
data2_delay=np.zeros(len(fitting.delay2))
data2_rise=np.zeros(len(fitting.tau_rise2))
data2_decay=np.zeros(len(fitting.tau_decay2))

'''data 3 is parameters fitted with bounds on delay
and amplitude and time vector is cut when Hay-signal
reaches V_rest after peaking''
data3_amplitude=np.zeros(len(fitting.amplitude3))
data3_delay=np.zeros(len(fitting.delay3))
data3_rise=np.zeros(len(fitting.tau_rise3))
data3_decay=np.zeros(len(fitting.tau_decay3))

'''total data set, the mean of dataset 1, 2 and 3'''
data_total_amp=np.zeros(len(fitting.amplitude3))
data_total_del=np.zeros(len(fitting.delay3))
data_total_rise=np.zeros(len(fitting.tau_rise3))
data_total_decay=np.zeros(len(fitting.tau_decay3))

'''the distance from soma, used for all the fitting processes'''
t=np.zeros(len(fitting.Hay.synapse_distance))

for i in range(len(fitting.amplitude1)):

    '''dataset 1'''
    data1_amplitude[i]=fitting.amplitude1[i]
    data1_delay[i]=fitting.delay1[i]
    data1_rise[i]=fitting.tau_rise1[i]
    data1_decay[i]=fitting.tau_decay1[i]

    '''dataset 2'''
    data2_amplitude[i]=fitting.amplitude2[i]
    data2_delay[i]=fitting.delay2[i]
    data2_rise[i]=fitting.tau_rise2[i]
    data2_decay[i]=fitting.tau_decay2[i]

    '''dataset 3'''
    data3_amplitude[i]=fitting.amplitude3[i]
    data3_delay[i]=fitting.delay3[i]
    data3_rise[i]=fitting.tau_rise3[i]
    data3_decay[i]=fitting.tau_decay3[i]

    '''distance from soma used for all the fitting peocesses'''
    t[i]=fitting.Hay.synapse_distance[i]
```

```
''' Synapse parameters for fitting method 4'''
for i in range(len(data1_amplitude)):
    data_total_amp[i]=((data1_amplitude[i]+data2_amplitude[i]+data3_amplitude[i])/3)
    data_total_del[i]=((data1_delay[i]+data2_delay[i]+data3_delay[i])/3)
    data_total_rise[i]=((data1_rise[i]+data2_rise[i]+data3_rise[i])/3)
    data_total_decay[i]=((data1_decay[i]+data2_decay[i]+data3_decay[i])/3)

'''FITTING'''
'''dataset total'''
res_total_sigmoid_amp = minimize(costfun_sigmoid, x0=x_sigmoid_amp_total,
args=(t, data_total_amp),method='TNC', options={'disp': True})

res_total_sigmoid_del = minimize(costfun_sigmoid, x0=x_sigmoid_delay_total,
args=(t, data_total_del),method='TNC', options={'disp': True})

res_total_sigmoid_rise = minimize(costfun_sigmoid, x0=x_sigmoid_rise_total,
args=(t, data_total_rise),method='TNC', options={'disp': True})

res_total_sigmoid_decay = minimize(costfun_sigmoid, x0=x_sigmoid_decay_total,
args=(t, data_total_decay),method='TNC', options={'disp': True})

'''dataset 1'''
res1_sigmoid_amp = minimize(costfun_sigmoid, x0=x_sigmoid_amp_1,
args=(t, data1_amplitude),method='TNC', options={'disp': True})

res1_sigmoid_del = minimize(costfun_sigmoid, x0=x_sigmoid_delay_1,
args=(t, data1_delay),method='TNC', options={'disp': True})

res1_sigmoid_rise = minimize(costfun_sigmoid, x0=x_sigmoid_rise_1,
args=(t, data1_rise),method='TNC', options={'disp': True})

res1_sigmoid_decay = minimize(costfun_sigmoid, x0=x_sigmoid_decay_1,
args=(t, data1_decay),method='TNC', options={'disp': True})

'''dataset 2'''
res2_sigmoid_amp = minimize(costfun_sigmoid, x0=x_sigmoid_amp_2,
args=(t, data2_amplitude),method='TNC', options={'disp': True})

res2_sigmoid_del = minimize(costfun_sigmoid, x0=x_sigmoid_delay_2,
args=(t, data2_delay),method='TNC', options={'disp': True})

res2_sigmoid_rise = minimize(costfun_sigmoid, x0=x_sigmoid_rise_2,
args=(t, data2_rise),method='TNC', options={'disp': True})

res2_sigmoid_decay = minimize(costfun_sigmoid, x0=x_sigmoid_decay_2,
args=(t, data2_decay),method='TNC', options={'disp': True})

'''dataset 3'''
res3_sigmoid_amp = minimize(costfun_sigmoid, x0=x_sigmoid_amp_3,
```

```
args=(t, data3_amplitude),method='TNC', options={'disp': True})

res3_sigmoid_del = minimize(costfun_sigmoid, x0=x_sigmoid_delay_3,
args=(t, data3_delay),method='TNC', options={'disp': True})

res3_sigmoid_rise = minimize(costfun_sigmoid, x0=x_sigmoid_rise_3,
args=(t, data3_rise),method='TNC', options={'disp': True})

res3_sigmoid_decay = minimize(costfun_sigmoid, x0=x_sigmoid_decay_3,
args=(t, data3_decay),method='TNC', options={'disp': True})

"error estimation"
'''Fitting method 4'''
error_total_sig_amp = (np.sqrt((sigmoid(res_total_sigmoid_amp.x,t)-
(data_total_amp))**2))
error_total_sigmoid_amp = error_total_sig_amp.sum()

error_total_sig_del = (np.sqrt((sigmoid(res_total_sigmoid_del.x,t)-
(data_total_del))**2))
error_total_sigmoid_del = error_total_sig_del.sum()

error_total_sig_rise = (np.sqrt((sigmoid(res_total_sigmoid_rise.x,t)-
(data_total_rise))**2))
error_total_sigmoid_rise = error_total_sig_rise.sum()

error_total_sig_decay = (np.sqrt((sigmoid(res_total_sigmoid_decay.x,t)-
(data_total_decay))**2))
error_total_sigmoid_decay = error_total_sig_decay.sum()

'''Fitting method 1'''
error1_sig_amp = (np.sqrt((sigmoid(res1_sigmoid_amp.x,t)-
 (data1_amplitude))**2))
error1_sigmoid_amp = error1_sig_amp.sum()

error1_sig_del = (np.sqrt((sigmoid(res1_sigmoid_del.x,t)-
(data1_delay))**2))
error1_sigmoid_del = error1_sig_del.sum()

error1_sig_rise = (np.sqrt((sigmoid(res1_sigmoid_rise.x,t)-
(data1_rise))**2))
error1_sigmoid_rise = error1_sig_rise.sum()

error1_sig_decay = (np.sqrt((sigmoid(res1_sigmoid_decay.x,t)-
(data1_decay))**2))
error1_sigmoid_decay = error1_sig_decay.sum()

'''Fitting method 2'''
error2_sig_amp = (np.sqrt((sigmoid(res2_sigmoid_amp.x,t)-
(data2_amplitude))**2))
```

```
error2_sigmoid_amp = error2_sig_amp.sum()

error2_sig_del = (np.sqrt((sigmoid(res2_sigmoid_del.x,t)-
(data2_delay))**2))
error2_sigmoid_del = error2_sig_del.sum()

error2_sig_rise = (np.sqrt((sigmoid(res2_sigmoid_rise.x,t)-
(data2_rise))**2))
error2_sigmoid_rise = error2_sig_rise.sum()

error2_sig_decay = (np.sqrt((sigmoid(res2_sigmoid_decay.x,t)-
(data2_decay))**2))
error2_sigmoid_decay = error2_sig_decay.sum()

'''Fitting method 3'''
error3_sig_amp = (np.sqrt((sigmoid(res3_sigmoid_amp.x,t)-
(data3_amplitude))**2))
error3_sigmoid_amp = error3_sig_amp.sum()

error3_sig_del = (np.sqrt((sigmoid(res3_sigmoid_del.x,t)-
(data3_delay))**2))
error3_sigmoid_del = error3_sig_del.sum()

error3_sig_rise = (np.sqrt((sigmoid(res3_sigmoid_rise.x,t)-
(data3_rise))**2))
error3_sigmoid_rise = error3_sig_rise.sum()

error3_sig_decay = (np.sqrt((sigmoid(res3_sigmoid_decay.x,t)-
(data3_decay))**2))
error3_sigmoid_decay = error3_sig_decay.sum()
```

11.4 Statistical analyses for the result functions

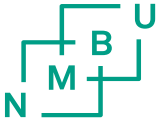
```
import numpy as np
import matplotlib.pyplot as plt
import Hay

"making list to classify in which part a synapse is placed"
Apical_indexes= []
for i  in range(380,560):
    Apical_indexes.append(i)

Basal_indexes = []
for i  in range(0,263):
    Basal_indexes.append(i)

Branches_indexes= []
for i  in range(274,305):
    Branches_indexes.append(i)
for i in range(308,333):
    Branches_indexes.append(i)
for i in range(336,341):
    Branches_indexes.append(i)
for i in range(344,347):
    Branches_indexes.append(i)
for i in range(349,359):
    Branches_indexes.append(i)
for i in range(362,367):
    Branches_indexes.append(i)
for i in range(560,642):
    Branches_indexes.append(i)

Stem_indexes= [263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 305,
306, 307, 333, 334, 335,341, 342, 343,347,348,359, 360, 361, 367, 368, 369,
370, 371, 372, 373, 374, 375, 376, 377, 378,379]

"Parameters Fit 4"
"Parameters for the sigmoid function,
listed as: Amplitude; tau_rise, tau_decay and delay"
Apical_parameters = np.ndarray(shape=(4,5))
Apical_parameters[0]=[ 6.0111, 1.5793, -0.0014006, -299.62, -0.48486]
Apical_parameters[1]=[0.24439, 0.050617, 0.011020, 398.92, 0.30105]
Apical_parameters[2]=[-212.83, -38.124, 0.010239, 825.63, 0.013853]
Apical_parameters[3]=[ 0.93106, 0.14311, 0.0082072, 787.04, 500.18]

Stem_parameters = np.ndarray(shape=(4,5))
Stem_parameters[0]=[ 0.48596, -0.42209, -0.0012222, 527.25, -0.0018240]
Stem_parameters[1]=[ 3.7897, 0.10165, 0.0054307, 591.62, 0.026216]
Stem_parameters[2]=[ 26.072, 5.0104, 0.017599, 285.02, 0.64334]
Stem_parameters[3]=[1.1811, -0.18931, 0.0018811, 1159.7, 499.98]
```

```
Basal_parameters= np.ndarray(shape=(4,5))
Basal_parameters[0]=[ 0.42072, -0.31757, -0.0034918, 257.05, -0.11902]
Basal_parameters[1]=[ 6.2114, -1.0814, 0.0046929, 546.46, -0.38962]
Basal_parameters[2]=[ 21.808, 5.0073, 0.015379, 200.11, -0.15656]
Basal_parameters[3]=[ 1.8872, 0.35925, 0.0029808, 744.21, 499.91]


Branches_parameters = np.ndarray(shape=(4,5))
Branches_parameters[0]=[0.58890, -0.35605, -0.0011595, 616.94, -0.55906]
Branches_parameters[1]=[1.5846, 0.82517, 0.012222, 185.19, -0.22803]
Branches_parameters[2]=[29.237, 4.4077, 0.010437, 311.47, -0.58806]
Branches_parameters[3]=[0.78439, 0.13001, 0.010582, 521.31, 500.15]

def sigmoid(x, distance):
    '''
    difference of two exponentials function implementation
    kwargs: x : list/tuple/array, [xo,x1,x2,t0,x3,x4]
            t : np.ndarray, time vector of signal
        returns          np.ndarray, sigmoid function
    '''
    y=x[4]+(x[0]/(x[1]+np.exp(-x[2]*(distance-x[3]))))

    return y

def exp2(x, t):
    '''
    difference of two exponentials function implementation
    kwargs: x : list [delay, tau_rise, tau_decay, constant, amplitude]
            t : np.ndarray, time vector of signal
    returns:    np.array, the difference of two exponentials
    '''
    Tm=9.67
    t=t-x[2]
    y=np.zeros(t.size)+Hay.cell.somav[8000]
    [i] = np.where((t>=0))
    A= (x[0]*Tm)/(x[0]-Tm)
    B= (x[1]*Tm)/(x[1]-Tm)
    num=abs((A-B)*np.exp(-t[i]/Tm)-A*np.exp(-t[i]/x[0])+B*np.exp(-t[i]/x[1]))
    if i.size>0:
        denom=abs((A-B)*np.exp(-t[i]/Tm)-A*np.exp(-t[i]/x[0])+
        B*np.exp(-t[i]/x[1])).max()
        y[i]=Hay.cell.somav[8000]+x[3]*(num/denom)
        return y

t=Hay.cell.tvec
amp= []
rise=[]
decay = []
delay = []
```

```
counter=0

for i in Hay.run_list:
    data = Hay.soma_voltage[counter]

    if i in Apical_indexes:
        parameters=Apical_parameters
        print 'Apical'
    if i in Basal_indexes:
        parameters=Basal_parameters
        print 'Basal'
    if i in Branches_indexes:
        parameters=Branches_parameters
        print 'Branches'
    if i in Stem_indexes:
        parameters=Stem_parameters
        print 'stem'

    amp.append(sigmoid(parameters[0], Hay.synapse_distance[counter]))
    rise.append(sigmoid(parameters[1], Hay.synapse_distance[counter]))
    decay.append(sigmoid(parameters[2], Hay.synapse_distance[counter]))
    delay.append(sigmoid(parameters[3], Hay.synapse_distance[counter]))
    counter+=1

"find the statistical values"
synapse_parameters= np.ndarray(shape=(counter, 4))
for i in range(counter):
    synapse_parameters[i][0]=decay[i]
    synapse_parameters[i][1]=rise[i]
    synapse_parameters[i][2]=delay[i]
    synapse_parameters[i][3]=amp[i]

soma_voltage = np.zeros(shape=(counter, 11201))
for i in range(counter):
    soma_voltage[i]=exp2(synapse_parameters[i],t)

time_between_vmax_halfs=[]

for i in range (counter):
    "finding time for maximun soma-response:"
    data=soma_voltage[i]

    "find the time between vmax-halfs fitted signal"
    voltage_start=data[8000:]
    index_top=(np.abs(voltage_start)-amp[i]).argmin()
    vmax_half=((amp[i])/2)+data[8000]
    index_half_up= (np.abs(voltage_start[:index_top]-vmax_half)).argmin()
    index_half_down= (np.abs(voltage_start[index_top:]-vmax_half)).argmin()
```

```
    time_between_vmax_halfs.append(abs(Hay.cell.tvec[index_half_up+8000]
    -Hay.cell.tvec[index_half_down+8000+index_top]))

"calculte mean values for the fitted signal:"
mean_amp=sum(amp)/counter
mean_delay=sum(delay)/counter

"calculate mean values for the Hay signal:"
voltage_max=Hay.voltage_max-Hay.cell.somav[8000]
mean_Hay_amp=sum(voltage_max)/counter
mean_Hay_delay=sum(Hay.t0_list)/counter
mean_time_between_vmax_halfs=sum(time_between_vmax_halfs)/counter
mean_Hay_time_between_vmax_halfs=sum(Hay.time_between_vmax_halfs)/counter

"calculate the standard deviations for the fitted signal:"
sd_amp=0
sd_delay=0
sd_time_between_vmax_halfs=0
for i in range(counter):
    sd_amp+=(amp[i]-mean_amp)**2
    sd_delay+=(delay[i]-mean_delay)**2
    sd_time_between_vmax_halfs+=(time_between_vmax_halfs[i]-
    mean_time_between_vmax_halfs)**2
sd_amp=np.sqrt(sd_amp/counter)
sd_delay=np.sqrt(sd_delay/counter)
sd_time_between_vmax_halfs=np.sqrt(sd_time_between_vmax_halfs/counter)

"calculate the standard deviation for the Hay signal:"
sd_Hay_amp=0
sd_Hay_delay=0
sd_Hay_time_between_vmax_halfs=0

for i in range(counter):
    sd_Hay_amp+=(voltage_max[i]-mean_Hay_amp)**2
    sd_Hay_delay+=(Hay.t0_list[i]-mean_Hay_delay)**2
    sd_Hay_time_between_vmax_halfs+=(Hay.time_between_vmax_halfs[i]-mean_Hay_time_betwee

sd_Hay_amp=np.sqrt(sd_Hay_amp/counter)
sd_Hay_delay=np.sqrt(sd_Hay_delay/counter)
sd_Hay_time_between_vmax_halfs=np.sqrt(sd_Hay_time_between_vmax_halfs/counter)

"plot the EPSPs for point model an Hay-model separately"
plt.subplot(211)
for i in range(counter):
    plt.plot(t,exp2(synapse_parameters[i],t))
plt.ylabel('Fitted signal [mV]')
plt.subplot(212)
for i in range(counter):
    data=Hay.soma_voltage[i]
```

```
        plt.plot(t,data)
plt.ylabel('Hay signal [mV]')
plt.xlabel('[ms]')
plt.show()
```



Norwegian University
of Life Sciences

Postboks 5003
NO-1432 Ås, Norway
+47 67 23 00 00
www.nmbu.no