Bielefeld University

Masters Thesis

Simulating sedation-induced unconsciousness in a Neural-Mass-Model to improve algorithms for state-of-consciousness detection in patients with unresponsive wakefulness syndrome

Author:
Felix Friese

Supervisor:
Dr. Malte Schilling

Examiner:
Prof Dr. Helge Ritter

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Neuroinformatics Groups
Faculty of Technology

March 10, 2022

BIELEFELD UNIVERSITY

Abstract

Faculty of Technology
Neuroinformatics Groups

Master of Science

Simulating sedation-induced unconsciousness in a Neural-Mass-Model to improve algorithms for state-of-consciousness detection in patients with unresponsive wakefulness syndrome

by Felix Friese

Patients with severe Traumatic Brain Injuries (TBI) often remain in a state of unresponsive wakefulness. Brain-Computer-Interface (BCI)-based systems promise to improve the state assessment and to open a communication channel for patients to express their intent while in conscious states. Developing such a BCI-System (e.g. with EEG), including the necessary algorithms to assess a patients current wakefulness or consciousness state from EEG data is a challenging task. Development, testing and evaluation of these algorithms requires labeled data (ground truth), which is almost impossible to obtain given the patients' lack of communication capabilities. Therefore, it would be desirable to generate a synthetic signal, which should ideally resemble real EEG data in all relevant features.

We previously developed a simple ICA-based model, which generates a multichannel EEG from base-signals with configurable spectral features. While this proved useful for testing numerous components of our signal-analysis framework, it lacks biological plausibility and explanatory power to model the changes in the signal's properties given an altered state of consciousness.

In this thesis, we propose an approach towards overcoming these issues while sticking with the original goal of generating realistic, practically useful surrogate data. A biologically motivated Neural Mass Model (NMM) on cortical-column level is implemented, which is able to approximate the effects of sedation-induced unconsciousness on the generated signal. The model is then shown to be able to reproduce the characteristic effects that sedation has on the EEG-Signals of real subjects. This is a first step to a model of consciousness-altering processes in the brain, which could ultimately be extended to realistically simulate other processes like sleep and trauma-induced DoC, facilitating better detection algorithms and furthering the goal to develop working BCI-Systems in the given context.

Contents

List of Abbreviations

NMM	**N**eural **M**ass **M**odel
DoC	**D**isorder **o**f **C**onsciousness
TBI	**T**raumatic **B**rain **I**njury
BCI	**B**rain **C**omputer **I**nterface
ICA	**I**ndependent **C**omponent **A**nalysis
PSP	**P**ost-**S**ynaptic-**P**otential
EPSP	**E**xcitatory **P**ost-**S**ynaptic-**P**otential
IPSP	**I**nhibitory **P**ost-**S**ynaptic-**P**otential
GABA	**G**amma-**A**mino**b**utyric **A**cid
EEG	**E**lectro**e**ncephalo**g**raphy
PC	**P**yramidal **C**ell
EIN	**E**xcitatory **I**nter**n**euron
IIN	**I**nhibitory **I**nter**n**euron

Chapter 1

Introduction

TODO: *go more into detail about the motivation and the challenges of developing stable signal processing algorithms for use cases with little (labeled) data*

1.1 Related Work

TODO: *Menon [9], Liang et al. [11], COALIA [14], ...*

Chapter 2

Technical Concepts

2.1 States of Consciousness

2.1.1 Definition

TODO: *which states of consciousness are commonly defined*

2.1.2 State Transitions

2.1.2.1 Natural Transitions

TODO: *how states transition into each other naturally*

2.1.2.2 Sedation

TODO: *how sedation differs from natural loss of consciousness*

2.1.3 Disorders of Consciousness

TODO: *the symptoms of DOCs*

2.2 Neurobiology

TODO: *straightforward recap of the basics, more focus on topic-relevant details*

2.2.1 Membrane Potential

Nerve cells (neurons) have a resting membrane potential of roughly -70mV. This electric potential is the result of a multitude of factors that ultimately decide the different concentrations of ions inside and outside of the cell, which in turn cause the potential difference. The most important ions involved in this process are sodium (Na^+), potassium (K^+) and chloride (Cl^-). These ions cannot simply cross the lipid-bilayer of the cell membrane by themselves. To do that, they depend on special enzymes embedded into the membrane: ion-channels, ion-pumps and ion-transporters. Ion-channels can either be permea

The most important ion-pumps are sodium-potassium pumps, which move potassium into the cell, while expelling sodium.

Sitting in the membrane are multiple kinds of ion-channels that can change the permeability for specific ions depending on certain factors. For example, voltage-gated channels open and close depending on the membrane potential, while ligand-gated channels are controlled by certain chemicals (neurotransmitters) binding to them.

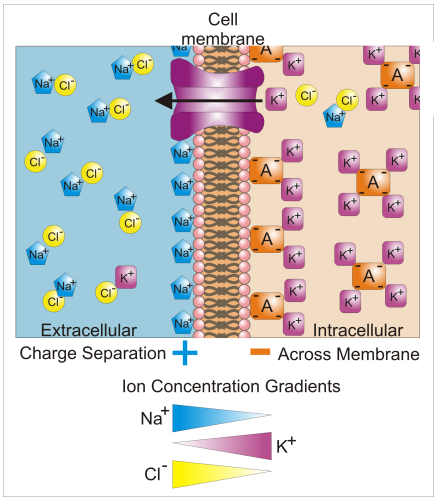

Figure 2.1: Membrane potential: Basic schema.

By Synaptidude - Own work, CC BY-SA 3.0 `https://commons.wikimedia.org/wiki/File:Basis_of_Membrane_Potential2.png`

A cell's resting potential primarily depends on the equilibrium state of the potassium ions, where the influx of potassium ions due to the concentration gradient is equal to the efflux due the voltage gradient.

The membrane potential can leave it's resting state, when processes disturb this balance. Most commonly, when ligand-gated ion-channels open and let ions enter the cell. Depending on the type of ion, this can lead to depolarisation or hyperpolarization of the membrane potential. Because depolarization increases the chances of a neuron firing, this process is also described as *excitation*, while polarization, which in turn decreases that chance, is called *inhibition*.

2.2.1.1 Action Potential

When the membrane potential of a Neuron reaches a certain Threshold at the axon hillock, it triggers

2.3 EEG

2.3.1 Measurement

TODO: *how the EEG is measured technically and which neuronal processes it actually observes (signal amplification, pyramidal cells, ...)*

2.3.2 Advantages/Disadvantages

TODO: *spatial/temporal resolution, invasiveness, ...*

2.3.3 States of Consciousness in the EEG Signal

TODO: *how do the states differ in the signal*

2.3.3.1 State Detection in healthy Subjects

TODO: *rough explanation of prevalent algorithms*

2.3.3.2 State Detection in Subjects with DOC

TODO: *issues with data collection, structurally different source signal, ...*

2.3.4 Simulation

2.3.4.1 Motivation

TODO: *what can we hope to achieve by simulating an EEG signal*

2.3.4.2 Approaches

TODO: *which tools are at our disposal (naive frequency mixing, population models, simulating individual neurons, ...)*

2.3.4.3 Model Choice

TODO: *argue about models => why did we land on NMMs/population models?*

2.4 Neural Mass Models

TODO: *slightly deeper introduction (we already have this in the EEG section) to NMMs in general*

2.4.1 The Jansen-Rit Model

The widely used Jansen-Rit Model [5], [6], is based on earlier models by Wilson & Cowan [1], Lopes da Silva et al. [2], [3] and Zetterberg et al. [4]. It represents a cortical column in the brain, which is made up of three main components, each modeling a population of neurons with distinct characteristics.

The basic schema of the model is visualized in Fig. 2.2, showing the connections between the main components. There is a population of Pyramidal Cells which receives input from two populations of inter-neurons, one of which is excitatory while the other is inhibitory. Each of the inter-neuron-populations receives the output of the PC population. Additionally, there is external excitatory input to the PC population from other regions of the brain.

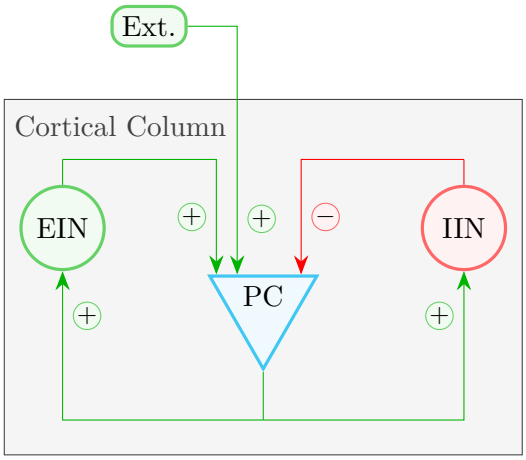

Figure 2.2: Basic Schema of the Jansen-Rit Model: Three populations of neurons

The Block Diagram (Fig. 2.3) shows the individual modules of the model. A population consists of two types of blocks: The *PSP-Block* models the behavior of the synapses and neuronal somata. It can be either excitatory or inhibitory and converts the incoming average pre-synaptic pulse density to an average post-synaptic membrane potential by convolving it with an impulse response function ($h_e(t)$ and $h_i(t)$, for excitation and inhibition respectively). The second block (sometimes called *Potential-To-Rate-Block* after it's functionality) calculates the populations response to this stimulation, transforming the incoming average membrane potential back into an average pulse density of action potentials. It may be roughly viewed as a functional counterpart to the axon hillock by establishing a firing threshold and is usually implemented by a Sigmoid Function ($Sigm$). External input from other regions of the brain is represented by $p(t)$. The Connectivity Constants C_1, C_2, C_3 and C_4 are a proportional representation of the average number of synapses between the populations. The signal most closely related to the EEG and therefore the variable of interest, is the summed average membrane potential of the PC population ($y_1(t) - y_2(t)$ in Fig. 2.3).

TODO: *explain neurophysiology why EEG* $\approx y_1 - y_2$*, or reference back to explanation in EEG section*

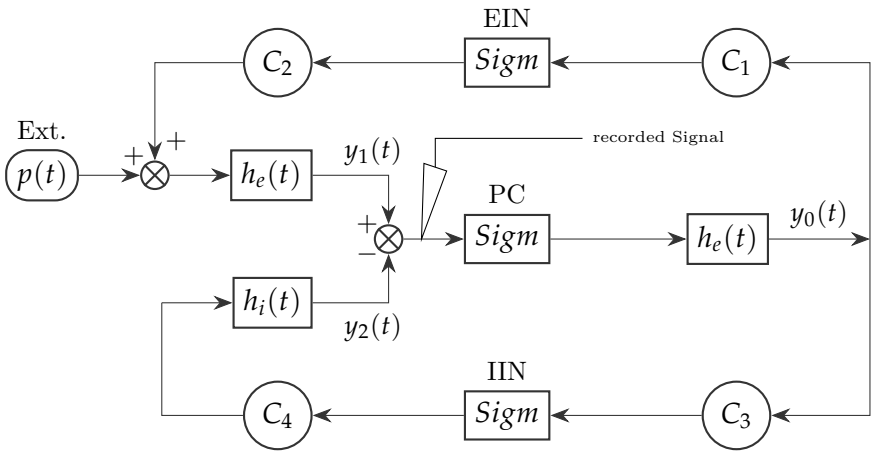

Figure 2.3: Simple Block Diagram after [6]: It's structure might be somewhat confusing when trying to visualize the biological analogy, where each population would have individual afferent PSP-Blocks. However, the fact that the two interneuron populations share a single excitatory PSP Block, because it produces identical results for both of them (disregarding their individual connectivity factor, which is simply applied afterwards), is a computational performance gain, thus likely explaining the authors' choice.

2.4.1.1 Potential-To-Rate Block

For a neuron to fire an action potential, its membrane potential needs to surpass a certain threshold. Since we are modeling not a single neuron but a whole population, we need an operator that can transform the mean membrane potential that the neurons of the population receive as input into an average firing rate for the whole population. While neurons within the population may have individual firing-thresholdit can be assumed due to their sheer number, that these thresholds are normally distributed around some mean value v_0. An additional assumption that this approach rests on, is that the number of afferent (i.e. incoming) connections to the individual neurons is sufficiently large to justify the assertion that all neurons receive roughly the same stimulation. This must be modeled by a monotonically increasing function. The Potential-To-Rate Block represents this process with a Sigmoid. After multiple iterations by Zetterberg [4], Lopes da Silva [3] and others, Jansen and Rit [5], [6] landed on the following equation:

$$Sigm(v) = \frac{2e_0}{1 + e^{r(v_0 - v)}} \tag{2.1}$$

The parameter values (Table 2.1) are empirically determined [5]. The maximum firing rate the population can achieve is set at $5Hz$. A mean membrane potential of $6mV$ (equal to the populations average firing threshold) elicits half of the maximum firing rate, while $\frac{0.56}{mV}$ defines the steepness. The plot in Fig. 2.4 visualizes these properties.

Parameter		Default Value	Unit
half of maximum firing rate	e_0	2.5	Hz
average firing threshold	v_0	6.0	mV
sigmoidal steepness	r	0.56	mV^{-1}

Table 2.1: Parameters of the Sigmoid Function

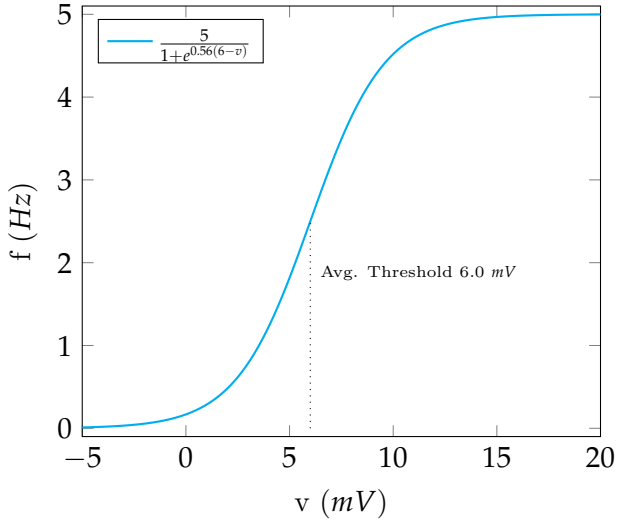

Figure 2.4: Sigmoid (Eq. 2.1) [5]

2.4.1.2 PSP-Blocks

In Physics, Linear Time-Invariant Systems (LTI systems) are oftentimes used to describe the response of electrical circuits to arbitrary input signals. They consist of a kernel function (or impulse-response function), that models the system's response to a single unit-impulse. The PSP-Blocks are an LTI system, fully represented by an impulse response function. It describes a PSP relative to the onset of a pulse. Since the PSP differs depending on the type of cell (excitatory or inhibitory) there are two different impulse-response functions. The parameters for the EPSP (Eq. 2.2) and IPSP (Eq. 2.3) are given in Table 2.2. The respective plots are visualized in Fig. 2.5.

Parameter		Default Value	Unit
Excmaxamplitude / e	A	3.25	mV
Lumped reprof sum of excdelays	a	100	Hz
Inh. max. amplitude / e	B	22	mV
Lumped reprof sum of inhdelays	b	50	Hz

Table 2.2: Parameters of the PSP Blocks

Excitatory impulse response:

$$h_e(t) = \begin{cases} Aate^{-at} & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2.2}$$

Inhibitory impulse response:

$$h_i(t) = \begin{cases} Bbte^{-bt} & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2.3}$$

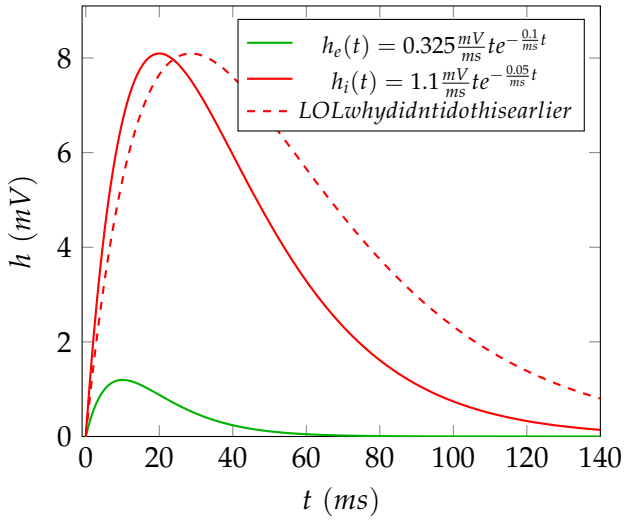

Figure 2.5: Impulse Response Functions: Note the small EPSP (Eq. 2.2) and the large IPSP (Eq. 2.3) [5]

Jansen and Rit [5] justify the difference in amplitude by referencing Lopes da Silva et al. [3] and stating that inhibitory neurons synapse closer to the somata of pyramidal cells (often on the cell body) than excitatory cells, increasing the effect of an inhibitory neuron about 10-fold.

TODO: *maybe go more into detail about the reasons for stronger inhibition*

The output of the Linear System defined by the PSP-Blocks is calculated by a convolution (denoted by $*$) of the incoming impulse density $x(t)$ with the impulse response function $h(t)$ (Eq. 2.4).

Remark (Convolution)**.** *The convolution of two functions $f(t)$ and $g(t)$ is defined as the integral of their product after one function has been reversed and shifted* [1]*:*

$$f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$

If $f(t)$ is a unit-impulse $\delta(t)$ (in our case that would mean each cell of the previous population firing a single action potential at the same time) the result is just $g(t)$ (in our case representing a single full-amplitude impulse response as the mean membrane potential):

$$\delta(t) * g(t) = \int_{-\infty}^{+\infty} \delta(\tau)g(t-\tau)d\tau = g(t)$$

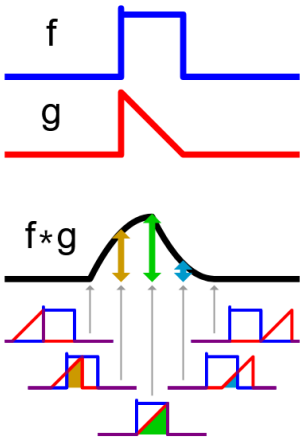

Figure 2.6: Convolution: The area enclosed by $f(\tau)$ and $g(t-\tau)$ is the value of $(f * g)(t)$.

By Cmglee - Own work, CC BY-SA 3.0 https://commons.wikimedia.org/w/index.php?curid=20206883

[1] There is a very intuitive explanation of convolutions by Kalid Azad on his website https://betterexplained.com/articles/intuitive-convolution/

In the general case, this process can be used to mathematically model the integration of incoming action potential densities in the soma.

Importantly, the Convolution Theorem states that the convolution of $f(t)$ and $g(t)$ becomes a simple multiplication when applying the Laplace Transform:

$$\mathcal{L}\{f(t) * g(t)\} = \mathcal{L}\{f(t)\}\mathcal{L}\{g(t)\} = F(s)G(s)$$

That means you can calculate a convolution with the inverse Laplace-Transform of the multiplication of the functions' individual Laplace-Transforms:

$$f(t) * g(t) = \mathcal{L}^{-1}\{F(s)G(s)\}$$

Since the convolution in the time-domain is a computationally heavy operation, it is oftentimes faster to transform the equation into the Laplace-Domain (see Eq. 2.5), apply the Convolution Theorem and perform the multiplication there, and transform the results back to the time-domain. This results in a second order differential equation (Eq. 2.6) that can be efficiently solved by numerical integration. To obtain this form, we need the Laplace transform $H_e(s)$ (in this context also called *Transfer Function*) of our response function $h_e(t)$:

$$H_e(s) = \mathcal{L}\{h_e(t)\} = \mathcal{L}\{Aate^{-at}\} = \frac{Aa}{(s+a)^2} = \frac{Aa}{s^2 + 2as + a^2}$$

With that, we can start to transform our initial equation into the desired Second Order System:

$$\overbrace{y(t)}^{\text{PSP}} = \overbrace{h_e(t)}^{\text{impulse response}} * \overbrace{x(t)}^{\text{impulse density}} \tag{2.4}$$

applying the Laplace-Transform eliminates the convolution:

$$\begin{aligned}
&\overset{\mathcal{L}}{\Longleftrightarrow} && Y(s) &&= \overbrace{H_e(s)}^{\text{transfer function}} \cdot X(s) && (2.5)\\
&\Longleftrightarrow && Y(s) &&= \frac{AaX(s)}{s^2 + 2as + a^2} &&\\
&\Longleftrightarrow && (s^2 + 2as + a^2)Y(s) &&= AaX(s) &&\\
&\Longleftrightarrow && s^2Y(s) + 2asY(s) + a^2Y(s) &&= AaX(s) &&
\end{aligned}$$

reversing the Laplace-Transform yields a differential equation in the time domain:

$$\begin{aligned}
&\overset{\mathcal{L}^{-1}}{\Longleftrightarrow} && \ddot{y}(t) + 2a\dot{y}(t) + a^2y(t) &&= Aax(t) &&\\
&\Longleftrightarrow && \ddot{y}(t) &&= Aax(t) - 2a\dot{y}(t) - a^2y(t) && (2.6)
\end{aligned}$$

which can be expressed as a system of two coupled first order equations:

$$\dot{y}(t) = z(t) \tag{2.7}$$

$$\dot{z}(t) = Aax(t) - 2az(t) - a^2y(t) \tag{2.8}$$

where $y(t)$ is the resulting PSP and $x(t)$ the incoming pulse density. This works analogously for the inhibitory case with $h_i(t)$.

TODO: *maybe explain why* $\mathcal{L}^{-1}\{sY(s)\} = \dot{y}(t)$ *and* $\mathcal{L}^{-1}\{s^2Y(s)\} = \ddot{y}(t)$

2.4.1.3 Full Linear System

Taking the two first order equations for $\dot{y}(t)$ (Eq. 2.7) and $\dot{z}(t)$ (Eq. 2.8), and the Block diagram (Fig. 2.7) as a base, we can now state the equations for the full Jansen-Rit Model with it's three populations. Each PSP-Block $h(t)$ needs it's own system of coupled differential equations. The value of $x(t)$ can be easily taken from the Block Diagram. $y_0(t)$ is the EPSP received by both the EIN and IIN population, while $y_1(t)$ is the EPSP and $y_2(t)$ the IPSP received by the PC population:

$$\begin{aligned}
\dot{y}_0(t) &= z_0(t) \\
\dot{z}_0(t) &= Aa\,Sigm[y_1(t) - y_2(t)] - 2az_0(t) - a^2y_0(t) \\
\dot{y}_1(t) &= z_1(t) \\
\dot{z}_1(t) &= Aa(p(t) + C_2\,Sigm[C_1y_0(t)]) - 2az_1(t) - a^2y_1(t) \\
\dot{y}_2(t) &= z_2(t) \\
\dot{z}_2(t) &= Bb(C_4\,Sigm[C_3y_0(t)]) - 2bz_2(t) - b^2y_2(t)
\end{aligned} \tag{2.9}$$

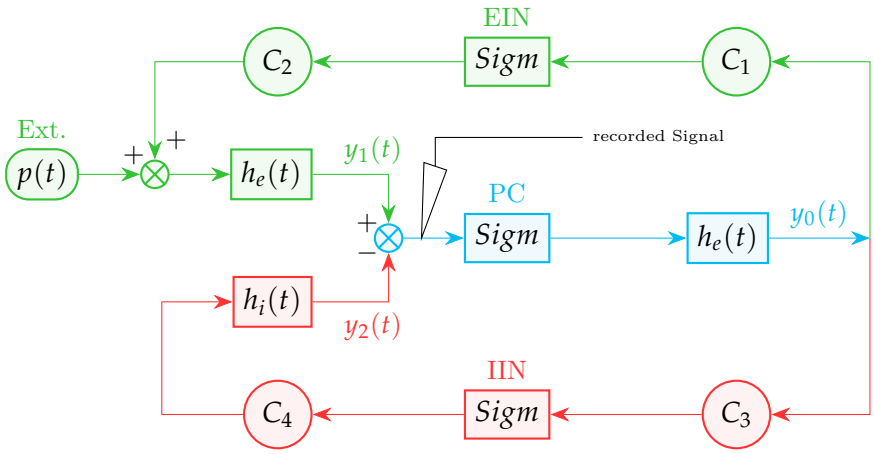

Figure 2.7: Colored Block diagram, visualizing the components of (Eq. 2.9)

2.4.1.4 Connectivity Constants

A sensible choice for the Connectivity Constants C_1 to C_4 was determined by Jansen and Rit empirically by defining a histologically motivated relationship between them ($C_1 = \frac{C_2}{0.8} = \frac{C_3}{0.25} = \frac{C_4}{0.25}$) and varying C_1 until the system produced the desired natural alpha-like activity at $C_1 = 135 \Rightarrow C_2 = 108; C_3 = C_4 = 33.75$. Varying C_1 can account for common synaptic phenomena like neurotransmitter depletion [6].

TODO: *go more into detail about the biological motivation and the effects of these constants on the generated signal*

2.4.1.5 Model Input

The model input $p(t)$ represents the average activity of populations outside of the modeled column that synapse on the columns PC population. Since this activity's source is so diverse, it is modeled by white noise (120-320 Hz).

TODO: *go more into detail why the input is modeled like this*

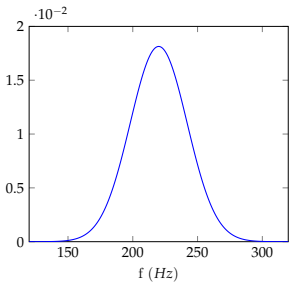

Figure 2.8: Input distribution. The input frequency representing $p(t)$ is sampled from a normal distribution with $\mu = 220$ and $\sigma = 22$

2.4.1.6 Model Output

The simulated data from $y_1 - y_2$ while varying C looks like this:

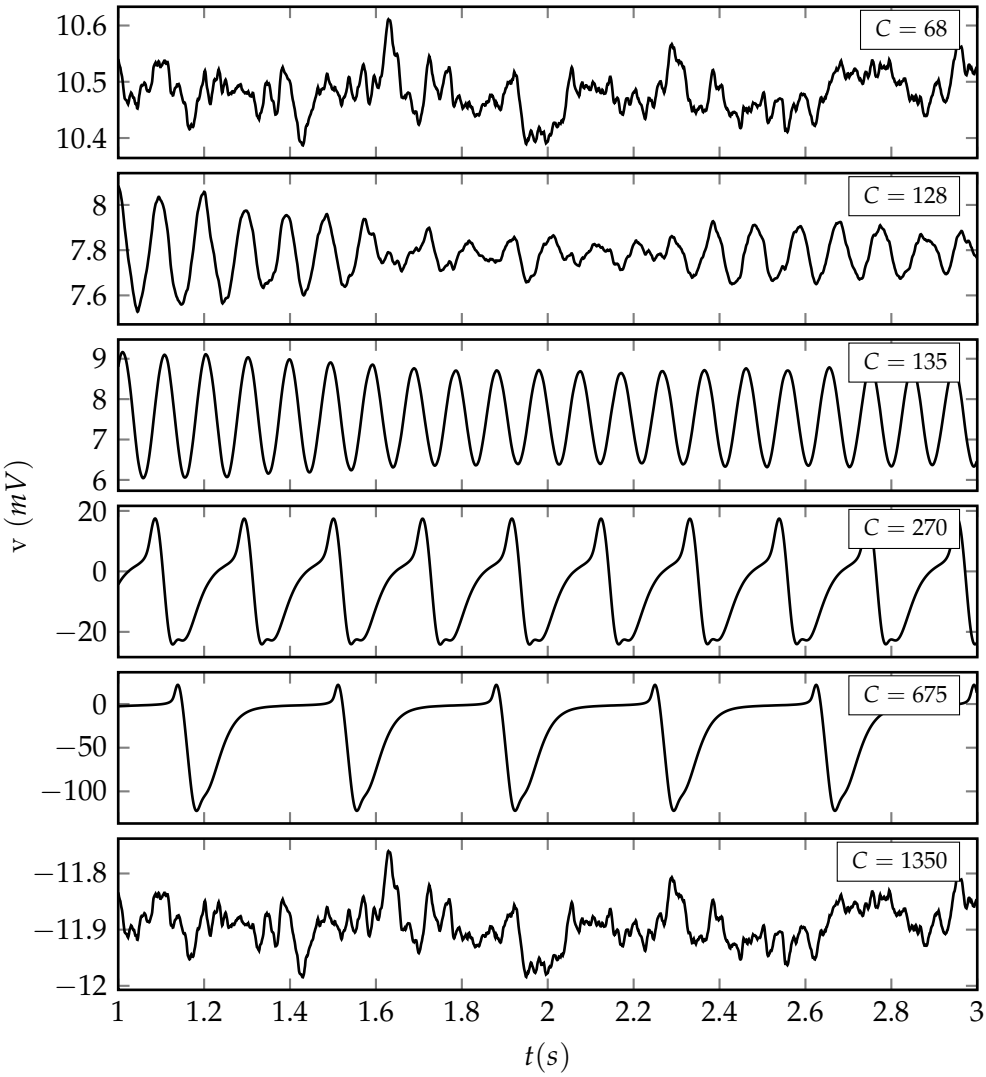

Figure 2.9: Model Output for varying C. Well defined alpha-activity is visible at $C = 135$.

TODO: *Explain Graph, generally provide more information to the systems output*

2.4.2 The David and Friston Model

Section Incomplete the whole David-Friston-Section is still very much preliminary

While the Jansen-Rit model succeeds in generating realistic alpha activity, real EEG Signals contain much richer spectra [7]. David and Fristion [8] proposed a modification to the Jansen-Rit model, that could produce a more realistic frequency spectrum by introducing sub-populations to the model. They can be tuned individually to produce oscillations in different frequencies.

2.4.2.1 Introducing sub-populations

David and Friston slightly redefine $h(t)$ by introducing the parameters H and τ (see Table 2.3), which is just a minor alteration of A and a.

$$h(t) = Aate^{-at} => h(t) = \frac{H}{\tau} te^{-\frac{1}{\tau}}$$

Furthermore, as they are tweaking these parameters to produce slower or faster subpopulations, they define the products $H_e\tau_e = 0.0325mVs$ and $H_i\tau_i = 0.44mVs$ as constants. This is done to preserve the oscillatory behavior of each population [8]. When varying τ, H is therefore adjusted accordingly ($H_e = \frac{0.0325mVs}{\tau_e}$, $H_i = \frac{0.44mVs}{\tau_i}$).

Parameter		Value	Unit	Relation to [6]
Excitatory delays	τ_e	0.01	s	$\tau_e = \frac{1}{a}$
Inhibitory delays	τ_i	0.02	s	$\tau_i = \frac{1}{b}$
Excitatory synaptic gain	H_e	3.25	mV	$H_e = A$
Inhibitory synaptic gain	H_i	22	mV	$H_i = B$

Table 2.3: Parameters of the PSP Blocks after [8]

Attention: From now on, the indices $[0, ..., N]$ for y, h, τ and H refer only to the subpopulations within a single population. The indices used above in the formulation for the Simple Jansen-Rit Model (and the Block Diagram) should not be confused with these. However, e and i as indices still denote excitatory and inhibitory populations respectively.

By introducing subpopulations, we split up the general impulse response function $h(t)$ in N individual subfunctions:

$$h_n(t) = \frac{H_n}{\tau_n} te^{-\frac{1}{\tau_n}}$$

The previously defined general PSP-Block Equation:

$$y(t) = h(t) * x(t)$$

then becomes:

$$y(t) = \sum_{n=0}^{N} (w_n \cdot h_n(t) * x(t)) \qquad \text{with} \qquad \sum_{n=0}^{N} w_n = 1 \qquad \text{and} \qquad 0 \leq w_n \leq 1$$

with N individually weighted (w_n) and parameterized ($h_n(t)$) subpopulations.
We can then declare:

$$y_n(t) = h_n(t) * x(t) \quad \text{and} \quad y(t) = \sum_{n=0}^{N} (w_n y_n)$$

which produces the following differential equations for a single PSP Block:

$$\begin{aligned}
\dot{y}_0(t) &= z_0(t) \\
\dot{z}_0(t) &= \frac{H_0}{\tau_0} x(t) - \frac{2}{\tau_0} z_0(t) - \left(\frac{1}{\tau_0}\right)^2 y_0(t) \\
&\ldots \\
\dot{y}_N(t) &= z_N(t) \\
\dot{z}_N(t) &= \frac{H_N}{\tau_N} x(t) - \frac{2}{\tau_N} z_N(t) - \left(\frac{1}{\tau_N}\right)^2 y_N(t) \\
y(t) &= w_1 y_1 + \ldots + w_N y_N
\end{aligned} \tag{2.10}$$

David and Friston further propose an example with two subpopulations for each population with the following parameters: $\tau_{e_1} = 10.8ms$, $\tau_{i_1} = 22ms$, $\tau_{e_2} = 4.6ms$, $\tau_{i_2} = 2.9ms$. While the kinetics for the first subpopulation were still close to those of the original populations ($\tau_e = 10ms, \tau_i = 20ms$, which produce alpha-activity), the second population's parameters were chosen to produce gamma activity.

TODO: *put the values in a table*

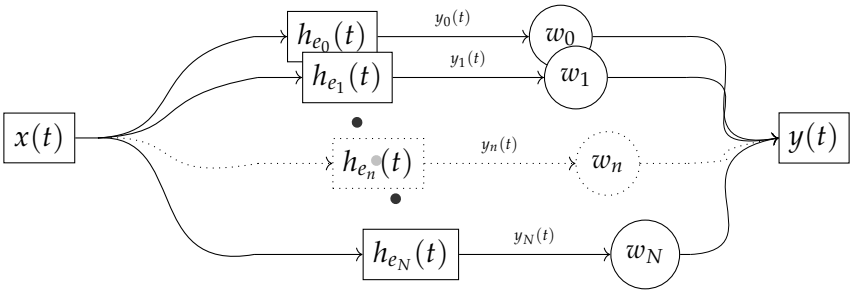

Figure 2.10: Example of subpopulations ($h_{e_0}(t), \ldots, h_{e_N}(t)$) forming an excitatory population $h_e(t)$

2.4.3 New PSP Functions

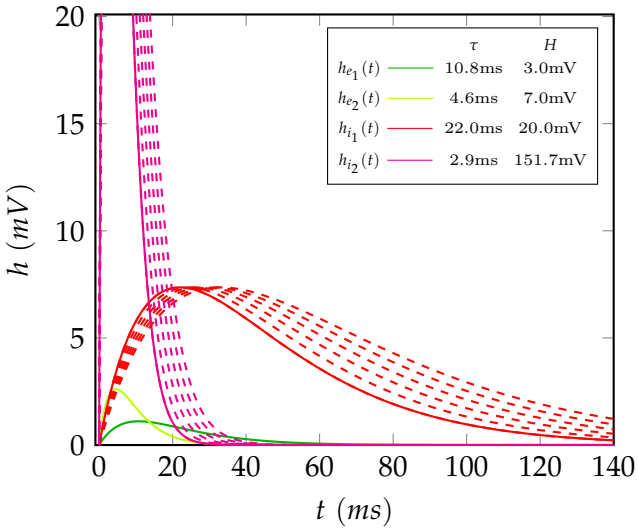

Figure 2.11: PSP functions for:

Chapter 3

Methodology

3.1 PyRates Framework

Section Incomplete The whole PyRates section is still very much preliminary

The PyRates Framework is a Python software framework, written by Richard Gast and Daniel Rose at the Max-Plank-Institute in Leipzig. It can simulate a wide range of graph-representable neural models, while setting a focus on rate-based population models [15]. It wraps computational backends like Numpy and Tensorflow and offers predefined nodes and edges (components that model units like cells or cell populations and the connections between them with mathematical equations) to be used, replaced or extended with custom equations. Furthermore it provides two simple ways to define these components and the derived network configurations: either by YAML-File or within Python code. These configurations are then compiled into optimized executable code with respect to the chosen backend before being executed. It comes with pre-configured model-definitions for some of the most frequently used models, e.g. the basic Jansen-Rit Circuit [6] and the Montbrio-Model [12], as well as some variations thereof. It's ease of use, the fact that it could easily reproduce the characteristics of the basic Jansen-Rit model out of the box, and the open-source character made it a sensible choice for this thesis.

3.1.1 Network Representations

3.1.1.1 YAML Representation

```yaml
JansenRitSynapse: # name of the template
    description: ... # optional descriptive text
    base: OperatorTemplate # parent template or Python class to use
    equations: # unordered list of equations
    - 'd/dt * V = V_t'
    - 'd/dt * V_t = h/tau * r_in - (1./tau)^2 * V - 2.*1./tau*V_t'
    - 'd/dt * V_t = h/tau * r_in - (1./tau)^2 * V - 2.*1./tau*V_t'
    variables: # additional information to define variables in equations
        r_in:
            default: input # defines variable type
        V:
            default: output
        V_t:
            description: integration variable # optional
            default: variable
        tau:
            description: Synaptic time constant
            default: constant
        h:
            default: constant
```

Figure 3.1: Example YAML Synapse

```yaml
JansenRitCircuit:
    base: CircuitTemplate
    nodes: # list nodes and label them
        EIN: ExcitatoryInterneurons
        IIN: InhibitoryInterneurons
        PC: PyramidalCellPopulation
    edges: # assign edges between nodes
    # [<source>, <target>, <template_or_operators>, <values>]
    - [PC/PRO/r_out, IIN/RPO_e/r_in, null, {weight: 33.75}]
    - [PC/PRO/r_out, EIN/RPO_e/r_in, null, {weight: 135.}]
    - [EIN/PRO/r_out, PC/RPO_e/r_in, null, {weight: 108.}]
    - [IIN/PRO/r_out, PC/RPO_i/r_in, null, {weight: 33.75}]
```

Figure 3.2: Example YAML Circuit

3.1.1.2 Python Representation

```
from pyrates.frontend import OperatorTemplate
from copy import deepcopy

pro = OperatorTemplate(
    name='PRO', path=None,
    equations=[
        # R_out = 2e_0 / (1+e^(r(v_0-v)))
        "rate_out = 2.*e_0 / (1 + exp(r*(v_0 - v)))"],
    variables={
        'rate_out': {'default': 'output'}, # output pulse density m_out
        'v': {'default': 'input'},  # incoming avg. membrane potential v
        'v_0': {'default': 6e-3},  # avg. firing thresh. v_0 = 6mV
        'e_0': {'default': 2.5},  # half of max. firing rate e_0 = 2.5Hz
        'r': {'default': 560.0}},  # sigmoidal steepness r = 560V^-1
    description="sigmoidal potential-to-rate operator")

rpo_e = OperatorTemplate(
    name='RPO_e', path=None,
    equations=[
        # ẏ(t) = z(t)
        'd/dt * y = z',
        # ż(t) = H/τ x(t) - 2/τ z(t) - 1/τ^2 y(t)
        'd/dt * z = H/tau * x - 2 * z/tau - y/tau^2'],
    variables={
        'y': {'default': 'output'},  # output membrane potential y(t)
        'z': {'default': 'variable'},  # helper variable z(t) = ẏ(t)
        'x': {'default': 'input'},  # incoming pulse density x(t)
        'tau': {'default': 0.01},  # exc. delays τ_e = 0.01s
        'H': {'default': 0.00325}},  # exc. synaptical gain H_e = 3.25mV
    description="excitatory rate-to-potential operator")

rpo_i = deepcopy(rpo_e).update_template(
    name='RPO_i', path='',
    variables={
        'tau': {'default': 0.02},  # inh. delays τ_i = 0.02s
        'H': {'default':  0.022}},  # inh. synaptical gain H_i = 22mV
    description="inhibitory rate-to-potential operator")

```

Figure 3.3: Python Example for the relevant Operators

3.1.2 Implementation of the Jansen-Rit Model

PyRates works with population models by compositing multiple operators, like the PSP- (or Rate-To-Potential-) and Sigmoid- (or Potential-To-Rate) Block into nodes. These nodes represent populations that can then be connected via edges (synapses). For example one might combine two PSP-Blocks (for excitatory and inhibitory input respectively) with a Sigmoid Block to create a PC-Node. This node can then receive

rate-input to each of it's PSP-Blocks and produces rate-output from it's Sigmoid-Block. The EIN- and IIN- nodes are functionally identical and just combine an excitatory PSP-Block with a Sigmoid Block. By connecting these Blocks (see Fig. 3.4) and adding random input to the excitatory PSP-Block of the PC-Node, the simple Jansen-Rit Circuit is already complete.

$$
\begin{aligned}
\frac{d}{dt}PSP_{EIN} &= PSP_{t_{EIN}} \\
\frac{d}{dt}PSP_{t_{EIN}} &= \frac{H_e}{\tau_e} \cdot C_1 Sigm[PSP_{PC}] - \frac{2}{\tau_e} \cdot PSP_{t_{EIN}} - \left(\frac{1}{\tau_e}\right)^2 \cdot PSP_{EIN} \\
\frac{d}{dt}PSP_{IIN} &= PSP_{t_{IIN}} \\
\frac{d}{dt}PSP_{t_{IIN}} &= \frac{H_e}{\tau_e} \cdot C_3 Sigm[PSP_{PC}] - \frac{2}{\tau_e} \cdot PSP_{t_{IIN}} - \left(\frac{1}{\tau_e}\right)^2 \cdot PSP_{IIN} \\
\frac{d}{dt}PSP_{PC_E} &= PSP_{t_{PC_E}} \\
\frac{d}{dt}PSP_{t_{PC_E}} &= \frac{H_e}{\tau_e} \cdot (p(t) + C_2 Sigm[PSP_{EIN}]) - \frac{2}{\tau_e} \cdot PSP_{t_{PC_E}} - \left(\frac{1}{\tau_e}\right)^2 \cdot PSP_{PC_E} \\
\frac{d}{dt}PSP_{PC_I} &= PSP_{t_{PC_I}} \\
\frac{d}{dt}PSP_{t_{PC_I}} &= \frac{H_i}{\tau_i} \cdot C_4 Sigm[PSP_{IIN}]) - \frac{2}{\tau_i} \cdot PSP_{t_{PC_I}} - \left(\frac{1}{\tau_i}\right)^2 \cdot PSP_{PC_I} \\
PSP_{PC} &= PSP_{PC_E} - PSP_{PC_I}
\end{aligned}
\tag{3.1}
$$

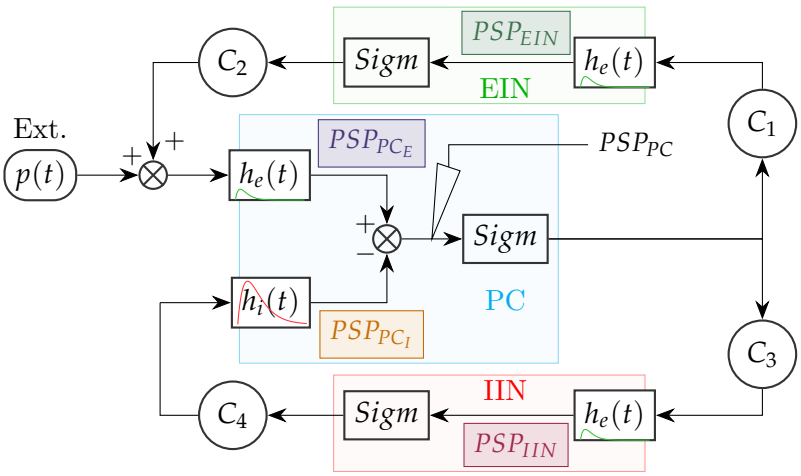

Figure 3.4: Jansen-Rit Block Diagram as implemented in PyRates: Each population can be clearly identified by one or more afferent PSP-Blocks and a single Sigmoid that calculates the populations output. This approach is more modular and simplifies conceptual understanding while staying mathematically equivalent. However, due to the explicit fourth PSP-Block it gives up the performance boost.

TODO: *possibly the backend-graph-optimization takes care of this? maybe check this later on...*

3.1.3 Implementation of the David & Friston extensions

```
rpo_e = OperatorTemplate(
    name='RPO_e', path=None,
    equations=[
        #--------------------------
        # Subpopulation 0: h_0(t)
        # ẏ_0 = z_0
        'd/dt * y_0 = z_0',
        # ż_0 = H_0/τ_0 x - 2/τ_0 z_0 - 1/τ_0^2 y_0
        'd/dt * z_0 = H_0/tau_0 * x - 2./tau_0 * z_0 - (1./tau_0)^2. * y_0',
        #--------------------------
        # Subpopulation 1: h_1(t)
        # ẏ_1 = z_1
        'd/dt * y_1 = z_1',
        # ż_1 = H_1/τ_1 x - 2/τ_1 z_1 - 1/τ_1^2 y_1
        'd/dt * z_1 = H_1/tau_1 * x - 2./tau_1 * z_1 - (1./tau_1)^2. * y_1',
        #--------------------------
        # Population output:
        # y = Σ_{n=0}^{N} (w_n y_n)
        'PSP = w_0*y_0 + w_1*y_1'
        #--------------------------
        ],
    variables={
        'PSP': {'default': 'output'},
        **{var: {'default': 'variable'} for var in ['y_0', 'y_1', 'z_0', 'z_1']},
        'x': {'default': 'input'},
        'w_0': {'default': 1.0},
        'w_1': {'default': 0.0},
        'tau_0': {'default': tau_0},
        'tau_1': {'default': tau_1},
        'H_0': {'default': h_0},
        'H_1': {'default': h_1}},
    description="rate-to-potential operator")
```

Figure 3.5: PSP Block with two Sub-populations in PyRates

3.1.4 Simulating GABA-A Sedatives

TODO: *why does the reduction of C represent inhibition of the whole system - what is the difference between thalamic regulation (natural sleep, etc) and GABA-A-receptor binding substances (sedation)?*

TODO: *do we have other ways of simulating sedatives in the system?*

Chapter 4

Results

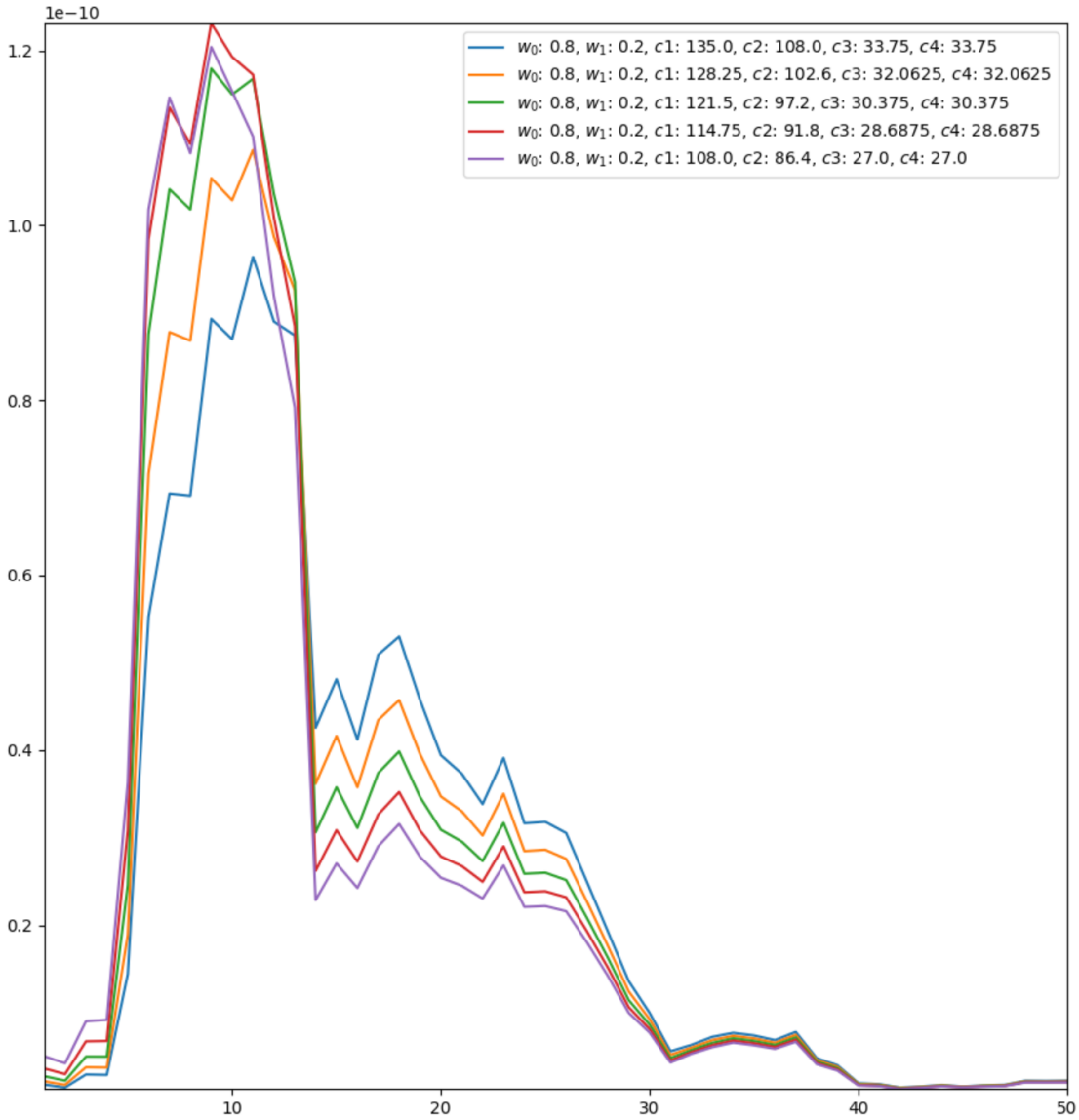

Figure 4.1: Power Spectral Density of Simulation Results: Reduction of the connectivity parameter C from 135 to 108 shows a tendency of reducing the strength of the frequency-bands above 12-14Hz, while increasing it below that value - especially around 8-12Hz.

TODO: *objective description of simulation results*

Chapter 5

Discussion

5.1 Comparison with real EEG Data during Propofol-Sedation

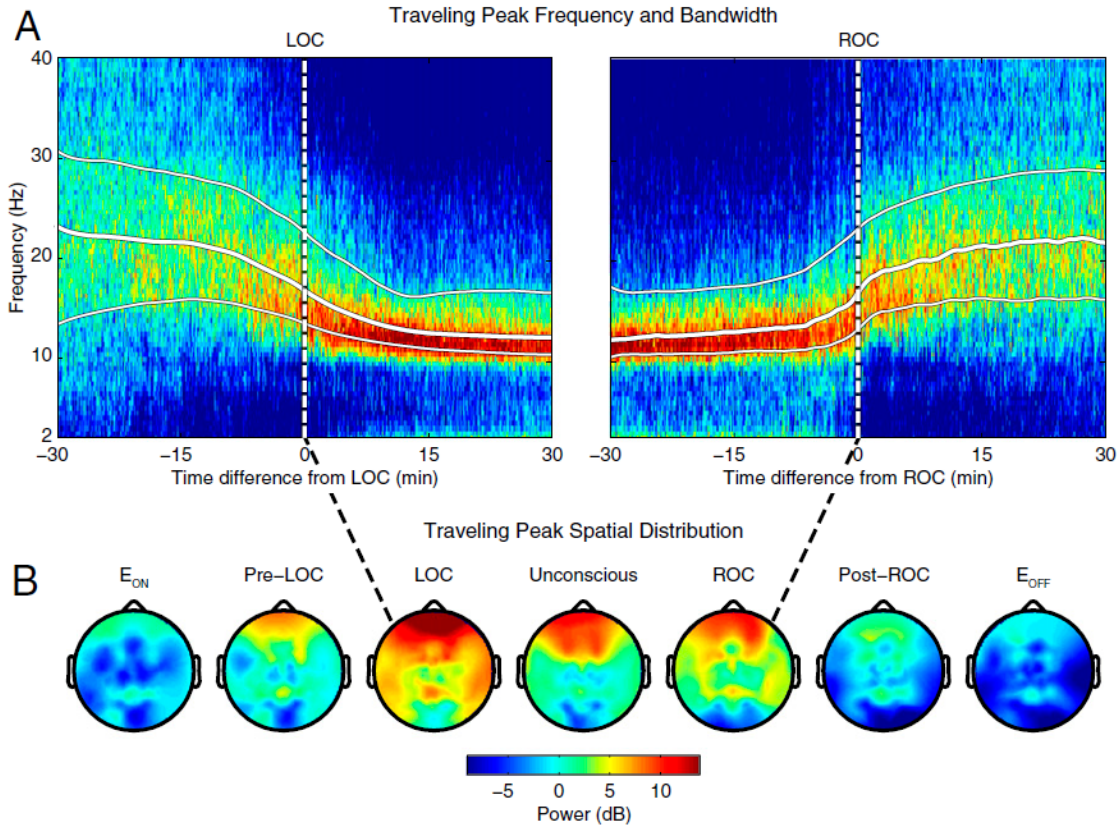

Figure 5.1: Real EEG Data [10]: Traveling Peak and general shift of Spectral Power from above 20 to the 10Hz Range during sedation. Also note the power increase in the very slow frequencies.

TODO: *compare to Purdon [10], Lee [13]*

5.2 Outlook

TODO: *how this could be extended towards our goal (include thalamus like in COALIA, find ways to simulate TBI-induced DOC more specifically, create combination of conditions and states of awareness in the same model, ...)*

Appendix A

PyRates Implementation Code

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal

from pyrates.frontend import OperatorTemplate, NodeTemplate, CircuitTemplate
from pyrates.utility.data_analysis import fft
from pyrates.utility.grid_search import grid_search


def create_pro(name):
    return OperatorTemplate(
        name=name, path='',
        equations=["rate_out = m_max / (1 + exp(r*(v_0 - PSP)))"],
        variables={'rate_out': {'default': 'output'},
                   'PSP': {'default': 'input'},
                   'v_0': {'default': 6e-3},
                   'm_max': {'default': 5.},
                   'r': {'default': 560.0}},
        description="sigmoidal potential-to-rate operator")


def create_rpo(name, tau_0, tau_1, h_0, h_1):
    return OperatorTemplate(
        name=name, path='',
        equations=['d/dt * y_0 = z_0',
                   'd/dt * z_0 = H_0/tau_0 * rate_in - (1./tau_0)^2. * y_0 - 2. * 1./tau_0 * z_0',
                   'd/dt * y_1 = z_1',
                   'd/dt * z_1 = H_1/tau_1 * rate_in - (1./tau_1)^2. * y_1 - 2. * 1./tau_1 * z_1',
                   'PSP = w_0*y_0 + w_1*y_1'],
        variables={'PSP': {'default': 'output'},
                   'y_0': {'default': 'variable'},
                   'y_1': {'default': 'variable'},
                   'z_0': {'default': 'variable'},
                   'z_1': {'default': 'variable'},
                   'rate_in': {'default': 'input'},
                   'w_0': {'default': 1.0},
                   'w_1': {'default': 0.0},
                   'tau_0': {'default': tau_0},
                   'tau_1': {'default': tau_1},
                   'H_0': {'default': h_0},
                   'H_1': {'default': h_1}},
        description="rate-to-potential operator")


def subpopulation_model():
    """
        Creates a Jansen-Rit Circuit with two subpopulations in each rate-to-potential operator
         as proposed by David and Friston
    """
    pro = create_pro(name='PRO')
    pro_pc = create_pro(name='PRO_pc')

    # subpopulation parameters taken from the paper
    tau_e = [10.8e-3, 4.6e-3]
    tau_i = [22e-3, 2.9e-3]

    h_e = [3.25e-3 * 10e-3 / tau for tau in tau_e]
    h_i = [-22e-3 * 20e-3 / tau for tau in tau_i]

```

```
    rpo_e = create_rpo('RPO_e', tau_e[0], tau_e[1], h_e[0], h_e[1])
    rpo_e_pc = rpo_e.update_template('RPO_e_pc', '', equations={'replace': {'rate_in': '(rate_in+p)'}},
                                     variables={'p': {'default': 220.0}})
    rpo_i = create_rpo('RPO_i', tau_i[0], tau_i[1], h_i[0], h_i[1])

    ein = NodeTemplate(name="EIN", path='', operators=[pro, rpo_e])
    iin = NodeTemplate(name="IIN", path='', operators=[pro, rpo_e])
    pc = NodeTemplate(name="PC", path='', operators=[pro_pc, rpo_e_pc, rpo_i])

    for rp in [rpo_i, rpo_e, rpo_e_pc]:
        print(rp.equations)
        print(rp.variables)

    jrc = CircuitTemplate(
        name="JRC", nodes={'PC': pc, 'EIN': ein, 'IIN': iin},
        edges=[
               ("PC/PRO_pc/rate_out", "EIN/RPO_e/rate_in", None, {'weight': 135.}),
               ("EIN/PRO/rate_out", "PC/RPO_e_pc/rate_in", None, {'weight': 108.}),
               ("PC/PRO_pc/rate_out", "IIN/RPO_e/rate_in", None, {'weight': 33.75}),
               ("IIN/PRO/rate_out", "PC/RPO_i/rate_in", None, {'weight': 33.75})],
        path='')
    return jrc


def simulate(template, step_size, sampling_step_size, seconds, cutoff):
    """
        Performs a grid search for w (mix the subpopulations)
    """
    ws = np.arange(0, 1.1, 0.2)
    param_grid = {
        'w_0': [round(w, 1) for w in ws],
        'w_1': [round(1-w, 1) for w in ws],
    }
    param_map = {
        'w_0': {
            'vars': ['RPO_e/w_0',
                     'RPO_e_pc/w_0',
                     'RPO_i/w_0'], 'nodes': ['PC', 'EIN', 'IIN']},
        'w_1': {
            'vars': ['RPO_e/w_1',
                     'RPO_e_pc/w_1',
                     'RPO_i/w_1'], 'nodes': ['PC', 'EIN', 'IIN']},
    }
    size = (int(np.round(seconds / step_size, decimals=0)), 1)

    results, results_map = grid_search(circuit_template=template,
                                       param_grid=param_grid,
                                       param_map=param_map,
                                       simulation_time=seconds,
                                       step_size=step_size,
                                       sampling_step_size=sampling_step_size,
                                       inputs={'PC/RPO_e_pc/p': np.random.normal(loc=220.0, scale=22.0, size=size)},
                                       outputs={'I0_pce': 'PC/RPO_e_pc/y_0',
                                                'I1_pce': 'PC/RPO_e_pc/y_1',
                                                'I0_pci': 'PC/RPO_i/y_0',
                                                'I1_pci': 'PC/RPO_i/y_1'},
                                       init_kwargs={'backend': 'numpy', 'solver': 'scipy'},
                                       verbose=False,
                                       permute_grid=False)
    results = results.loc[cutoff:]

    results = (results['I0_pce'] + results['I0_pci'])*results_map.loc[:, 'w_0'].values + \
              (results['I1_pce'] + results['I1_pci'])*results_map.loc[:, 'w_1'].values

    return results, results_map, sampling_step_size


def plot(results, results_map, sampling_step_size):
    """
        Plot the resulting signals and their spectrum
    """
    fig = plt.figure(figsize=(8, 12))
    fig.suptitle('David and Friston (2003) Fig.5')
    gs = fig.add_gridspec(len(results_map), 2)

    def periodogram(data, fs, freq_range):
        _f, power = scipy.signal.periodogram(data, fs)
        selection = np.where((freq_range[0] <= _f) & (_f <= freq_range[1]))
        f = _f[selection]
        power = power[selection]
        return f, power
```

```

    def welch(data, fs, freq_range):
        w_len = min(4096, int(len(data) / 2))
        win = scipy.signal.windows.hamming(w_len)
        _f, power = scipy.signal.welch(data, fs, nperseg=w_len,
                                       noverlap=int(w_len / 2), window=win)
        power /= np.sum(power)
        selection = np.where((freq_range[0] <= _f) & (_f <= freq_range[1]))

        return _f[selection], power[selection]

    def simple_fft(data, fs, freq_range):
        n = len(data)

        # Get closest power of 2 that includes n for zero padding
        n_two = 1 if n == 0 else 2 ** (n - 1).bit_length()

        data_tmp = data
        data_tmp = data_tmp - np.mean(data_tmp)

        freqs = np.linspace(0, fs, n_two)
        spec = np.fft.fft(data_tmp, n=n_two, axis=0)

        # Cut of PSD and frequency arrays since its mirrored at N/2
        power = np.abs(spec[:int(len(spec) / 2)])
        10 * np.log10(np.power(power, 2))
        _f = freqs[:int(len(freqs) / 2)]
        selection = np.where((freq_range[0] <= _f) & (_f <= freq_range[1]))
        f = _f[selection]
        power = power[selection]
        return f, power

    for row, key in enumerate(results_map.index):

        # get infos from results map
        ws = [results_map.at[key, c] for c in results_map.columns]
        v_lb = ", ".join([f'${key} = {w}$' for key, w in zip(results_map.columns, ws)])

        # calculate combined LFP
        #psp_0 = results.loc[:, ('I0_pce', key)] + results.loc[:, ('I0_pci', key)]
        #psp_1 = results.loc[:, ('I1_pce', key)] + results.loc[:, ('I1_pci', key)]
        #v = psp_0*ws[0] + psp_1*ws[1]
        #v = psp_0 + psp_1
        #v = results.loc[:, (key,)]
        v = results.iloc[:, row]
        ax = fig.add_subplot(gs[row, 0])
        ax.plot(results.index, v, label=v_lb, color=f'C{row}')
        ax.legend(loc='upper right')
        ax.margins(0.0)

        ax2 = fig.add_subplot(gs[row, 1])
        ax2.margins(0.0)
        ax2.plot(*simple_fft(data=np.squeeze(v.values), fs=1/sampling_step_size, freq_range=(0.1, 100)),
                 label=v_lb, color=f'C{row}')
        ax2.legend(loc='upper right')
        row += 1
    plt.tight_layout()
    plt.show()


if __name__ == '__main__':

    res = []
    for i in range(1):
        results, results_map, sampling_step_size = simulate(template=subpopulation_model(),
                    step_size=1e-4,  # choosing 1e-3 here does not really make a difference
                    sampling_step_size=1e-3,
                    seconds=2.0, cutoff=1.0)
        res.append(results)
    results = sum(res) / len(res)

    plot(results, results_map, sampling_step_size)
    # plot(*simulate(template=subpopulation_model(),
    #                step_size=1e-4,  # choosing 1e-3 here does not really make a difference
    #                sampling_step_size=1e-3,
    #                seconds=2.0, cutoff=1.0))
```

Bibliography

[1] H. R. Wilson and J. D. Cowan, "Excitatory and Inhibitory Interactions in Localized Populations of Model Neurons," en, *Biophysical Journal*, vol. 12, no. 1, pp. 1–24, Jan. 1972, ISSN: 00063495. DOI: 10.1016/S0006-3495(72)86068-5. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0006349572860685 (visited on 10/23/2021).

[2] F. H. Lopes da Silva, A. Hoeks, H. Smits, and L. H. Zetterberg, "Model of brain rhythmic activity: The alpha-rhythm of the thalamus," en, *Kybernetik*, vol. 15, no. 1, pp. 27–37, 1974, ISSN: 0340-1200, 1432-0770. DOI: 10.1007/BF00270757. [Online]. Available: http://link.springer.com/10.1007/BF00270757 (visited on 08/20/2021).

[3] F. Lopes da Silva, A. van Rotterdam, P. Barts, E. van Heusden, and W. Burr, "Models of Neuronal Populations: The Basic Mechanisms of Rhythmicity," en, in *Progress in Brain Research*, vol. 45, Elsevier, 1976, pp. 281–308, ISBN: 978-0-444-41457-1. DOI: 10.1016/S0079-6123(08)60995-4. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0079612308609954 (visited on 10/10/2021).

[4] L. H. Zetterberg, L. Kristiansson, and K. Mossberg, "Performance of a model for a local neuron population," en, *Biological Cybernetics*, vol. 31, no. 1, pp. 15–26, 1978, ISSN: 0340-1200, 1432-0770. DOI: 10.1007/BF00337367. [Online]. Available: http://link.springer.com/10.1007/BF00337367 (visited on 10/10/2021).

[5] B. H. Jansen, G. Zouridakis, and M. E. Brandt, "A neurophysiologically-based mathematical model of flash visual evoked potentials," en, *Biological Cybernetics*, vol. 68, no. 3, pp. 275–283, Jan. 1993, ISSN: 0340-1200, 1432-0770. DOI: 10.1007/BF00224863. [Online]. Available: http://link.springer.com/10.1007/BF00224863 (visited on 11/27/2020).

[6] B. H. Jansen and V. G. Rit, "Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns," en, p. 10, 1995.

[7] M. Steriade, "Impact of Network Activities on Neuronal Properties in Corticothalamic Systems," en, *Journal of Neurophysiology*, vol. 86, no. 1, pp. 1–39, Jul. 2001, ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.2001.86.1.1. [Online]. Available: https://www.physiology.org/doi/10.1152/jn.2001.86.1.1 (visited on 01/09/2022).

[8] O. David and K. J. Friston, "A neural mass model for MEG/EEG:: Coupling and neuronal dynamics," en, *NeuroImage*, vol. 20, no. 3, pp. 1743–1755, Nov. 2003, ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2003.07.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811903004579 (visited on 12/04/2020).

[9] J. P. Menon, V. Gupta, and R. Aravamudhan, "A computational model of mild traumatic brain injury," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, ISSN: 1558-4615, Aug. 2012, pp. 1354–1357. DOI: 10.1109/EMBC.2012.6346189.

[10] P. L. Purdon, E. T. Pierce, E. A. Mukamel, *et al.*, "Electroencephalogram signatures of loss and recovery of consciousness from propofol," en, *MEDICAL SCIENCES*, p. 10, 2013.

[11] Z. Liang, X. Duan, C. Su, L. Voss, J. Sleigh, and X. Li, "A Pharmacokinetics-Neural Mass Model (PK-NMM) for the Simulation of EEG Activity during Propofol Anesthesia," en, *PLOS ONE*, vol. 10, no. 12, D. Marinazzo, Ed., e0145959, Dec. 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0145959. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0145959 (visited on 10/23/2021).

[12] E. Montbrió, D. Pazó, and A. Roxin, "Macroscopic Description for Networks of Spiking Neurons," en, *Physical Review X*, vol. 5, no. 2, p. 021 028, Jun. 2015, ISSN: 2160-3308. DOI: 10.1103/PhysRevX.5.021028. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.5.021028 (visited on 01/09/2022).

[13] B.-R. Lee, D.-O. Won, K.-S. Seo, J. K. Hyun, and S.-W. Lee, "Classification of wakefulness and anesthetic sedation using combination feature of EEG and ECG," in *2017 5th International Winter Conference on Brain-Computer Interface (BCI)*, Jan. 2017, pp. 88–90. DOI: 10.1109/IWW-BCI.2017.7858168.

[14] S. Bensaid, J. Modolo, I. Merlet, F. Wendling, and P. Benquet, "COALIA: A Computational Model of Human EEG for Consciousness Research," English, *Frontiers in Systems Neuroscience*, vol. 13, 2019, Publisher: Frontiers, ISSN: 1662-5137. DOI: 10.3389/fnsys.2019.00059. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnsys.2019.00059/full (visited on 06/14/2020).

[15] R. Gast, D. Rose, C. Salomon, H. E. Möller, N. Weiskopf, and T. R. Knösche, "PyRates—A Python framework for rate-based neural simulations," en, *PLOS ONE*, vol. 14, no. 12, e0225900, Dec. 2019, Publisher: Public Library of Science, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0225900. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0225900 (visited on 11/27/2020).