Aria Alinejad

Signal compression techniques for multi-channel electrophysiological signals recorded in neuroscience research experiments

Specialization Project in Electrical Engineering
Supervisor: Lars Lundheim
Co-supervisor: Are Hellandsvik
November 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU

ABSTRACT

Brain disorders like Dementia and Parkinson's affect millions and are one of the biggest health problems that the world faces today. However, we are limited by a lack of mechanical understanding of the brain [1]. To address this, research is being conducted to develop wireless brain monitoring devices for use on rats. This report will explore compression techniques that can be used to enable the development of these devices by reducing power consumption. Various methods including Differential Pulse Code Modulation (DPCM), Adaptive Rice Coding, Finite Impulse Response (FIR) filtering, Discrete Cosine Transform (DCT), Principal Component Analysis (PCA), and MS Quadro will be evaluated, and their effectiveness in compressing Local Field Potentials (LFPs), Extracellular Action Potential (EAPs), Electrocorticography (EcoG) signals, and Electromyography (EMG) signals will be demonstrated. This report also provides an analysis of the signal characteristics of these four signals and their signal applications, and provides discussions of prior research, results, and future work in light of this. The MS Quadro method, combined with FIR filtering, DPCM, and Adaptive Rice coding, achieved a compression rate of 40-50% without compromising information integrity. This method's low hardware and computational demands suggest the potential for additional power savings, which has the potential to facilitate the creation of a wireless system for monitoring rat brain activity.

SAMMENDRAG

Hjernesykdommer som Demens og Parkinsons rammer millioner av mennesker og er em av de største helseproblemene verden står overfor i dag. Likevel er vi begrenset av manglende mekanisk forståelse av hjernen [1]. Heldigvis er det pågående forskning for å utvikle trådløse hjerneovervåkingsystemer til bruk på rotter. Dette prosjektet tar derfor for seg komprimeringsteknikker som kan muliggjøre utviklingen av denne teknologien gjennom reduksjon i strømforbruk. Ulike metoder, deriblant Differential Pulse Code Modulation (DPCM), Adaptive Rice Coding, Finite Impulse Response (FIR)-filtrering, Discrete Cosine Transform (DCT), Principal Component Analysis (PCA) og MS Quadro, vil bli evaluert, og det vil bli demonstrert hvor effektive de er til å komprimere signaler fra lokale feltpotensialer (LFP), ekstracellulære aksjonspotensialer (EAP), elektrokortikografi (EcoG) signaler og elektromyografi (EMG) signaler. Rapporten inneholder også en analyse av karakteristikkene og applikasjonene til signalene, samt diskusjon av publisert arbeid, resultater, og framtidig arbeid med bruk av den informasjonen. MS Quadrometoden, kombinert med FIR-filtrering, DPCM og Adaptive Rice koding, oppnår en komprimering på 40-50% uten at det går ut over informasjonsintegriteten. Metodens lave system- og beregningskrav peker mot et potensial for gode strømbesparelser, noe som kan muliggjøre utviklingen av et trådløst system for overvåking av hjerneaktiviteten til rotter.

PREFACE

This project was carried out as part of the 15-credit course TFE4580 at the Norwegian University of Science and Technology (NTNU). The topic sprung out as an extension of the work I started at SINTEF in the summer of 2023. The project therefore has an indirect connection to the project dubbed BrainChip, which is a collaboration project between the University of Oslo (UiO) and SINTEF [2, 3]. This report assumes fundamental knowledge of signal processing and electronics. I would like to thank my supervisor Lars Lundheim for all of the support, guidance, and insightful questions along the way. I would also like to thank my co-supervisor Are Hellandsvik for giving me the opportunity to work on this project, and for supporting and mentoring me from the very start. I am also grateful to Charlotte Boccara and her research team for inviting Espen Holsen and me to their laboratory in Oslo. This trip provided me with fresh perspectives and valuable understanding.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ADC** Analog to Digital Converter
- **AP** Action Potential
- **ASIC** Application Specific Integrated Circuit
- **BLE** Bluetooth Low Energy
- **CR** Compression Ratio
- **CS** Compressed Sensing
- **DCT** Discrete Cosign Transform
- **DFT** Discrete Fourier Transform
- **DPCM** Differential Pulse Code Modulation
- **DWT** Discrete Wavelet Transform
- **EAP** Extracellular Action Potential
- **EEG** Electroencephalogram
- **EcoG** Electrocorticography
- **EMG** Electromyography
- **FIR** Finite Impulse Response
- **FPGA** Field Programmable Gate Array
- **KLT** Karhunen-Loeve Transform
- **LFP** Local Field Potential
- **LNN** Linear Neural Network
- **LUT** Look Up Table
- **MSE** Mean Square Error

- **NTNU** Norwegian University of Science and Technology
- **PCA** Principal Component Analysis
- **RCR** Relative Compression Ratio
- **SVD** Singular Value Decomposition

CHAPTER

ONE

INTRODUCTION

This chapter outlines the project's motivation and goals, linking them to global concerns like neurological disorders and sustainable development. It then previews the report's structure and subsequent chapters.

1.1 Motivation

Figure 1.1.1: Sustainable development goals 3 and 10 from the United Nations [4].

The World Health Organization estimates that over a billion people suffer from neurological disorders and their effects across the world [5, 1]. Furthermore, these disorders unproportionally affect low-income countries. "A study conducted in Europe estimated that the annual economic cost of neurological diseases (dementia, epilepsy, migraine, and other headaches, multiple sclerosis, Parkinson's disease, and stroke) amounted to 139 billion euros in 2004" [5]. It is clear that the global burden is already high, and it is projected to increase even further. "A clear message emerges: unless immediate action is taken globally, the neurological burden is expected to become an even more serious and unmanageable threat to public health" [5]. Consequently, tackling these problems directly addresses two of the goals for sustainable development set by the United Nations, namely goals 3 and 10 displayed in figure 1.1.1 [4].

Currently, a great deal of brain research is being conducted through monitoring the brain activity of rats and mice. These experiments have resulted in several breakthroughs, including the discoveries that were awarded the 2014 Nobel Prize in Physiology and Medicine [6]. One common way to measure the brain activity of rodents is currently through the use of wired sensor systems surgically mounted to the heads of rodents called headstages. Figure 1.1.2 shows a rat with a wireless headstage on its head.

Figure 1.1.2: Rat with custom-built measurement device on its head (wireless headstage). Taken from [7] and used under creative commons CC BY 4.0 license.

Given the same test setting as depicted in the picture, a wire would significantly limit the movent of the rat and jeopardize the experiment. But as of now, no commercial wireless headstage with 16 or more channels exists. Whereas the wired headstages that are available significantly limit the range of experiments that are possible, since they don't allow for social and naturalistic experiments where the rodents can freely move and interact. A typical problem is that the rats chew on the wire given the chance, making a test testing with two or more rats using wired headstages problematic. This is why initiatives are being made to develop wireless headstages. One can only imagine the new types of experiments that can be done when wireless headstages become more developed and how these, in turn, might allow for new types of discoveries about the brain.

Making a wireless measurement device has several inherent difficulties. Ideally, the system should be lightweight, non-bulky, measure big amounts of data, have a long battery life, and maximal resolution on the data sent. Making such an embedded system is immensely difficult, and one of the biggest challenges is the high power consumption that results from the large amounts of data that need to be sent wirelessly. Therefore this report will explore compression of the neural data as a possible solution to this problem. By compressing the data we can reduce power consumption and allow for longer experiments, smaller battery packs, or more channels of data.

1.2 Background

This report has been in collaboration with the ongoing research project called BrainChip[2, 3]. This joint effort between The University of Oslo and SINTEF aims to produce a wireless headstage for researching the brain activity of rats.

1.3 Objective

The objective of the project is to evaluate and demonstrate various compression techniques based on an understanding of experimental measurement methods and the research goals of scientists addressing the problem at hand. The primary research question is: How well do different compression methods perform on neuroscientific data, and how do they compare?

1.4 Outline

Chapter 2 gives background information about the origin and applications of the signals that will be worked with. It also provides a brief theoretical background for some of the compression methods that will be tested. Chapter 3 uses the knowledge presented in chapter 2 to give an overview and understanding of the work that has been published on compression of the signals at hand. Chapter 5 describes the implementation of some of the compression methods that will be tested and compared. Chapter 4 presents the datasets that have been tested on and illustrates some of the characteristics of the signals in the dataset. Chapter 6 provides tables that summarize the results from testing different compression methods, and comments on the results. Chapter 7 contextualizes the results in the frame of the report as a whole and discusses flaws and possibilities for future work. Lastly, chapter 8 briefly ties together the information presented in this report and underlines important results and findings.

CHAPTER

TWO

BACKGROUND

This chapter starts with a short description of the brain and the signals it produces. Following this, a description of a whole signal pipeline from the sensors to digitally available data for a wired measurement system will be described. Subsequently, a possible design of a wireless signal acquisition system will be described. This is followed by explanations of some possible applications for the previously described brain signals. Lastly, a brief overview of relevant theory about signal compression will be given.

2.1 The Brain

The rat brain is similar in structure to the human brain and has been a useful model for studying the human brain for many years. Neurons are the fundamental unit of the brain and the nervous system for rats, humans, and all other animals [8]. Figure 2.1.1 shows a simple illustration of two neurons.

Each neuron takes input from numerous other neurons at its dendrites, if the summed potential at the dendrites is sufficiently large the neuron will send an action potential (AP) signal down the axon and to other neurons through the axon terminals. The connection point between the axon terminal of one neuron and the dendrites of another is called the synapse. If the receiving neuron has sufficient summed potential at its dendrites it then also sends out an AP signal. In this way the AP signal plays a central role in cell-to-cell communication, and enables our brain to communicate both internally and to our limbs [10].

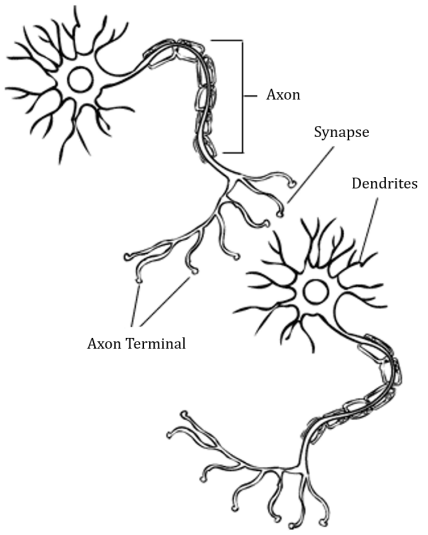


Figure 2.1.1: Simple drawing of two neurons, from [9].

The AP signal can be recorded by placing an electrode inside a neuron cell, in intracellular space, and measuring the membrane potential. Figure 2.1.2 gives a simplified depiction of the membrane potential of a neuron that transmits an AP signal. This quick rise and subsequent fall of the membrane potential is what is transmitted across the axon and referred to as the AP signal.

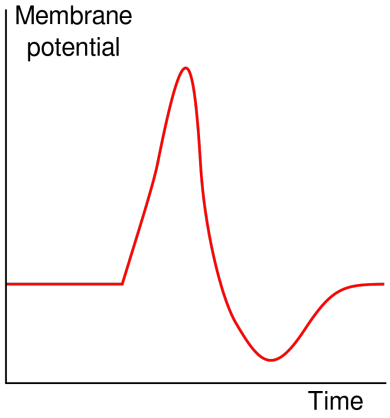


Figure 2.1.2: Membrane potential of neuron sending AP signal. Taken from [10] and used under creative commons CC BY-SA 4.0 license.

Part A of figure 2.1.3 shows how Electroencephalography (EEG), electrocorticography (EcoG), and Local Field Potential (LFP) signals all measure the brain but are distinguished by the placement of the electrodes.

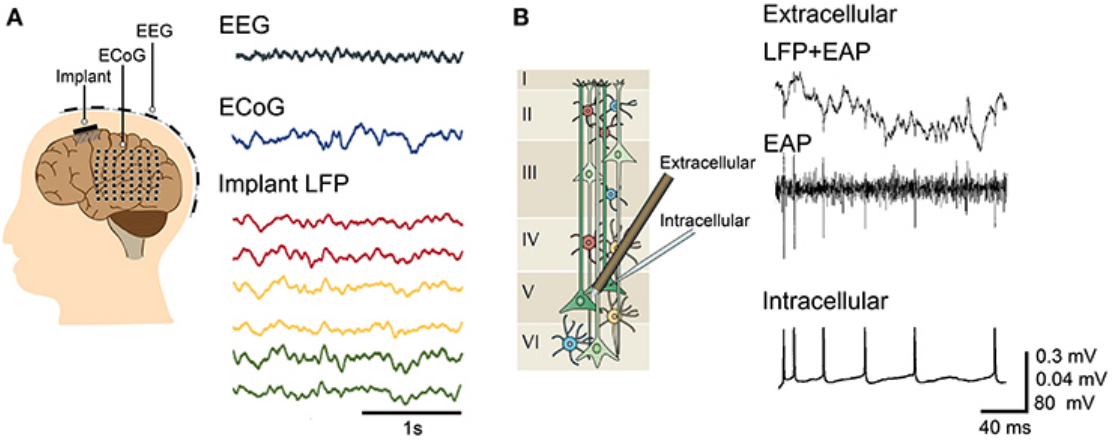


Figure 2.1.3: Part A illustrates where EEG, EcoG, and LFP are recorded. Part B Illustrates Intracellular and Extracellular electron recordings. Taken from [11] and used under creative commons CC BY 4.0 license.

EEG is recorded on the surface of the scalp, while EcoG records the surface of the brain below the skull [12].[1] While LFP is recorded in-depth, in extracellular space, inside the brain structure. All of these signals result from the summed

[1]More precisely the EcoG is recorded at a sub-dural level. Elementary information about brain anatomy is provided here [13].

and synchronous electrical activity of several neurons [14, 12].[2] The difference in EEG, EcoG, and LFP signals can be attributed to the different media that the different signals propagate through, which subjects them to different filtering. Despite these differences, EEG, EcoG, and LFP still have some similarities, for example, they display the same types of oscillations during wake and sleep states.

2.2 Design of wired headstages

Headstages are sensing systems that are surgically implanted into the brains of rats to measure their brain and muscle activity. Figure 2.2.1 shows a rat with a wired headsatge in a test environment. Each headstage measures several signals at several points, the totality of all the sensors that are used in the headstage will be referred to as the Drive. In the figure there is also a camera that monitors the mouse during the experiment. The data from the headstage in addition to the footage is sent to a computer simultaneously, thus mouse movement can be synced to the measured brain activity. However, the data from the headstage is first processed before being sent to the computer.

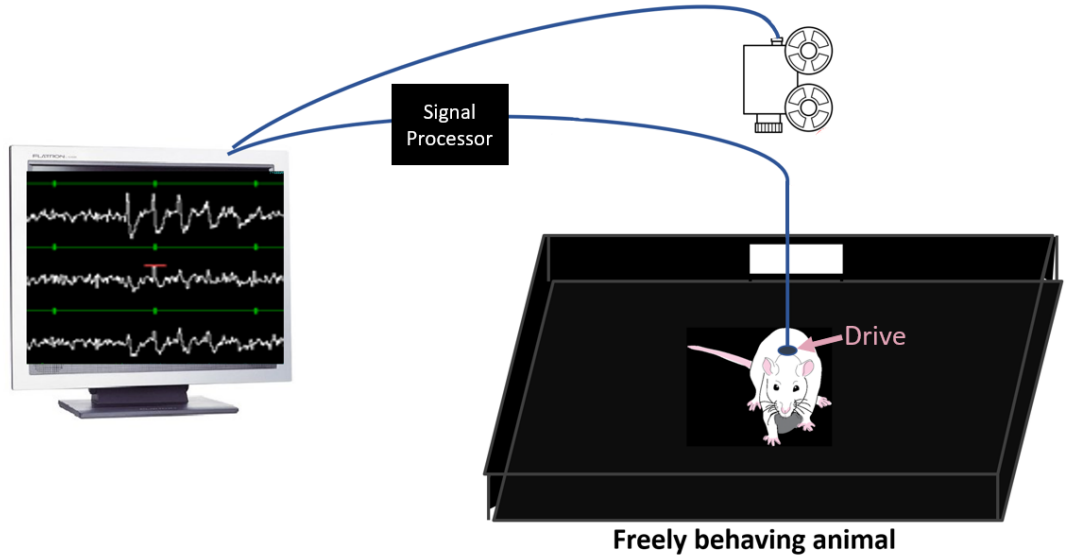


Figure 2.2.1: Test set up wired headstage on rat and camera.

The datasets that are used in this report are recorded from a wired headstage with a Drive that measures 16 channels of data. Figure 2.2.2 illustrates how the signal that is measured by the Drive is processed before it reaches the computer.

[2]The current view is that LFPs are generated by the synchronized synaptic currents arising on cortical neurons, which are the neurons in the outer layer of the brain, the cortex.

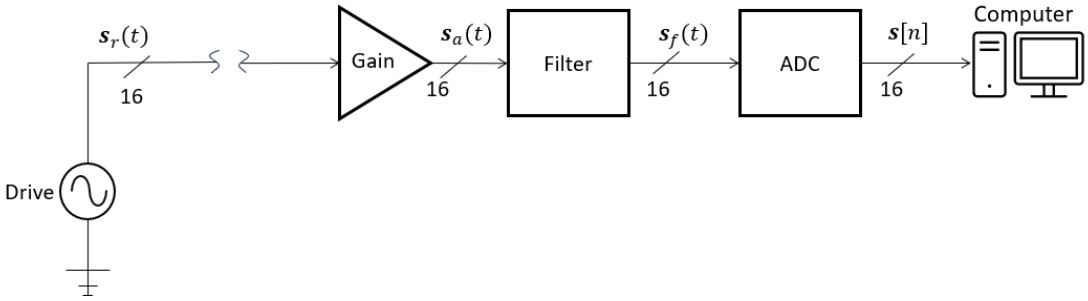


Figure 2.2.2: Block diagram of how the signal from a wired headstage is processed.

First the Drive records the raw 16 channel signal $\mathbf{s}_{r,i=[1,...16]}(t)$.[3] To avoid messy notation not indicating the channels $\mathbf{s}_r(t)$ will be used as a shorthand to denote all 16 channels of the signal $\mathbf{s}_{r,i=[1,...16]}(t)$. $\mathbf{s}_r(t)$ is first amplified. Note that the amplification can be set differently for each individual channel.[4] The amplified signal $\mathbf{s}_a(t)$ is then band pass filtered using analog filters. The filtered signal $\mathbf{s}_f(t)$ is then digitized with an analog-to-digital converter (ADC) using 16 bits per sample. The sampling frequency is usually set to either 24kHz or 48kHz. The output of the ADC is the digitized signal $\mathbf{s}[n]$, $n = 0, \ldots (N-1)$, where N is the number of samples in each channel $\mathbf{s}_i$. Lastly, $\mathbf{s}[n]$ is sent to the computer, ready to be analyzed.

[3]Bold indicates that it is a vector.

[4]The importance of this will become clear in section 4.

2.3 The sensors

Drives can be handmade, meaning that the sensors used can vary from experiment to experiment. The experiments where the datasets used in this report were recorded used 16 channel Drives as the one in figure 2.3.1.[5]

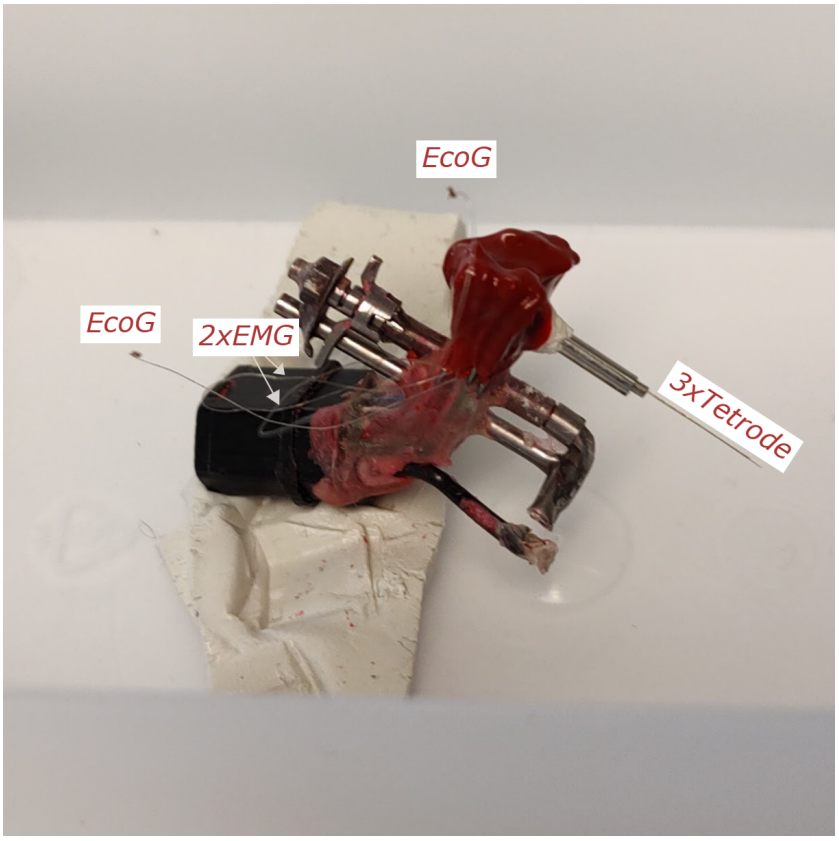


Figure 2.3.1: Drive with 3 tetrodes, 2 EcoG channels, and 2 EMG channels. Courtesy of Lina Okinina, Boccara laboratories [15].

Here there are two sensors for EcoG signals, two sensors for electromyography (EMG) signals, and three tetrodes. Each tetrode is made by twisting four electrodes together, meaning that they pick up four signals each, see figure 2.3.2.

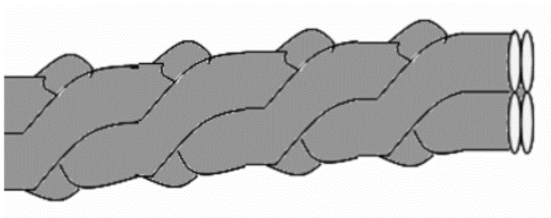


Figure 2.3.2: Four electrodes twisted together, making a tetrode. Courtesy of Charlotte Boccara, Boccara laboratories [15].

[5]Here red nail polish is used as an insulating seal.

Each electrode picks up the sum of LFP and AP signals in extracellular space. This is because when an electrode is placed in extracellular space it can be close enough to some neurons that it picks up extracellular Action Potentials (EAPs) in addition to LFP, see part B of figure 2.1.3. This figure also shows how an electrode in intracellular space picks up the AP signals directly. In contrast, an extracellular electrode picks up the AP signals from a distance, which subjects the signal to noise in addition to the AP spikes.

The total signal that each electrode picks up can be separated by filtering the signal. The lower band, spanning approximately 0.1 Hz to 300 Hz, contains the LFP signals. While the higher band, ranging from about 300 Hz to 5000 Hz, contains the EAP signals.[6] It is also worth noting that the EcoG signal has a similar frequency range to that of LFP. Figure 2.3.3 and 2.3.4 show a low-pass and high-pass filtered tetrode signal respectively.

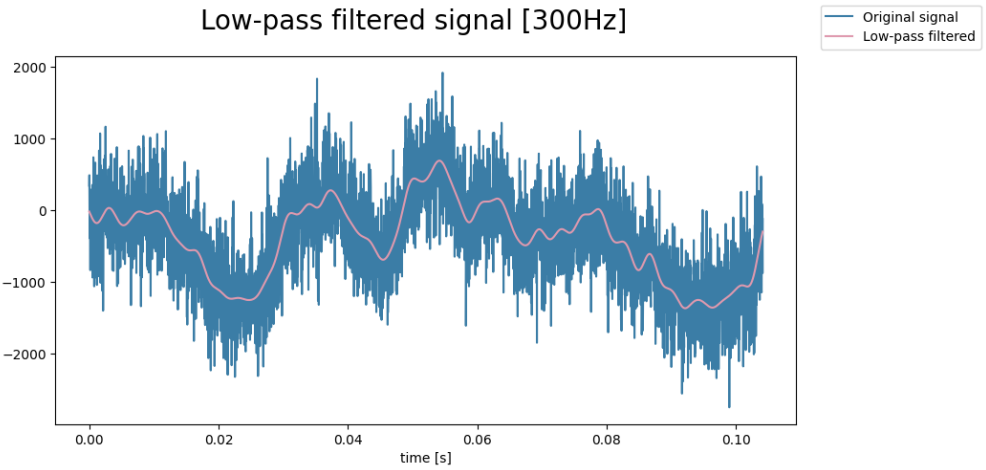


Figure 2.3.3: Low-pass filtered tetrode signal, 0.1 - 300Hz.

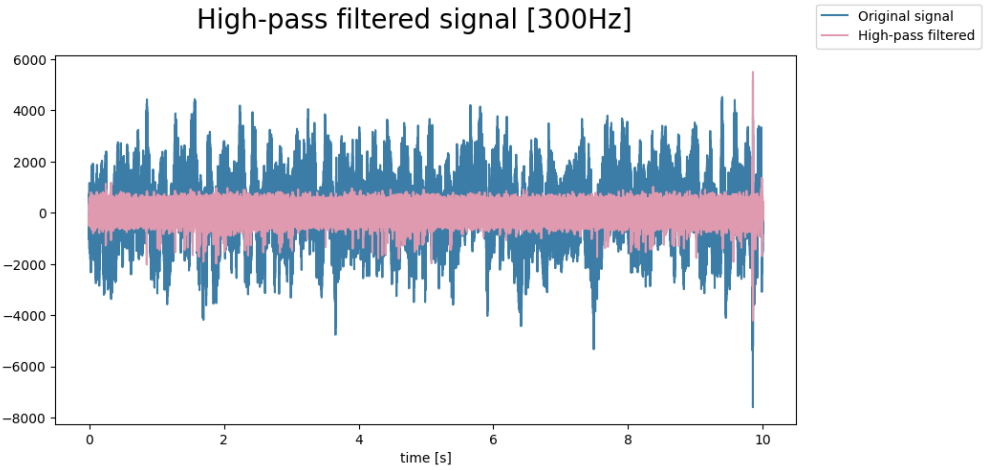


Figure 2.3.4: High-pass filtered tetrode signal, 300 - 5000Hz.

[6]There doesn't seem to be any widely accepted or definitive answer regarding the exact frequency bands in this context.

Each electrode in a tetrode is made of tungsten or platinum/iridium and coated in polyamide. The twisted quartet is then cut at the end. Thus each tetrode has four measurement points in close proximity at their bottom, resulting from the fact that the metal core conducts electricity while the polyamide insulates. Figure 2.3.5 shows a tetrode that measures four signals from two neurons.

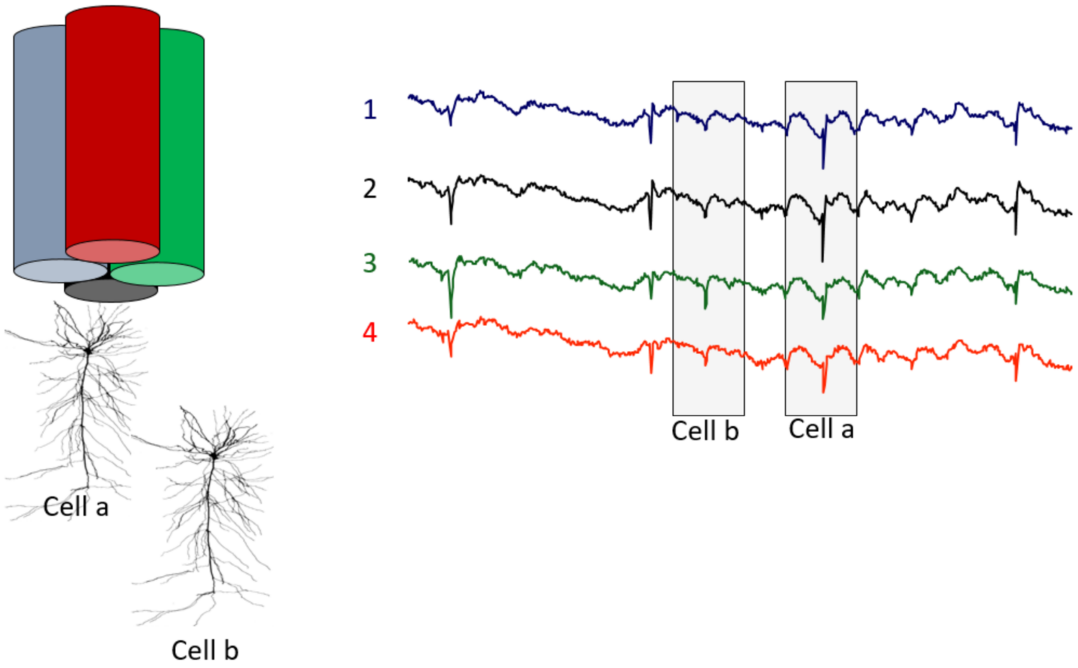


Figure 2.3.5: Tetrode picking up signals from two neurons. Courtesy of Charlotte Boccara, Boccara laboratories [15]

As shown by the figure, each tetrode is surrounded by several neurons that each give out different AP signals. The four points on the same tetrode typically pick up the same AP signals, but one can determine which direction the AP signal comes from based on the different magnitudes measured at different points. Furthermore, the different tetrodes are typically separated by enough distance so that they don't pick up the same AP signals.

Figure 2.3.6 shows how a Drive with a tetrode can be placed into the brain of a rodent.

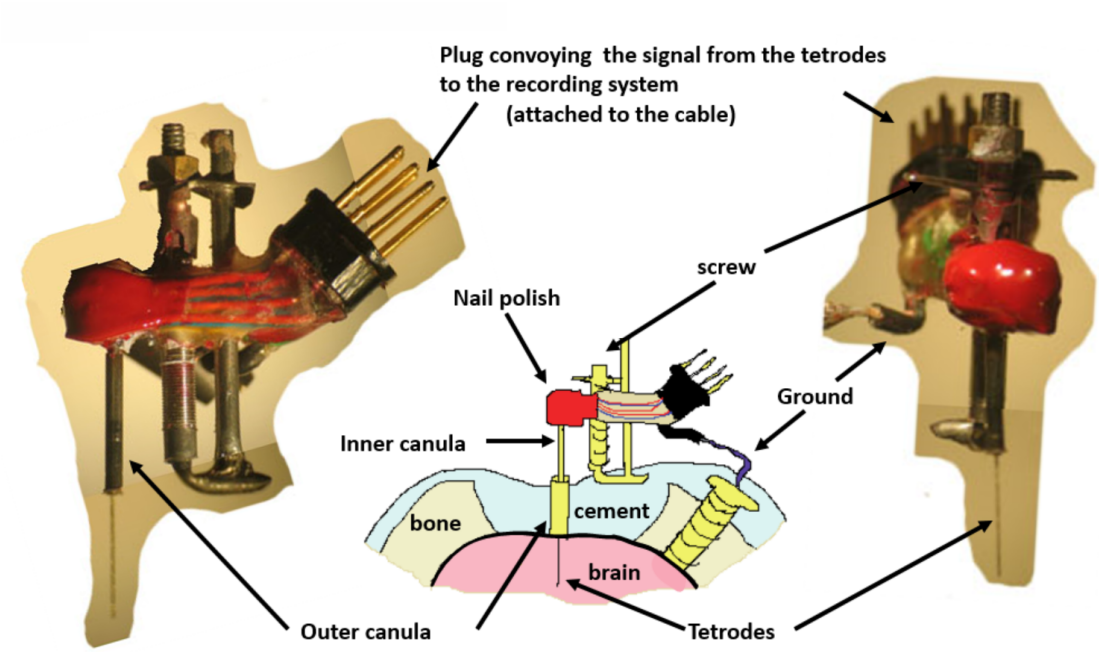


Figure 2.3.6: Depiction of how the Drive is placed. Courtesy of Charlotte Boccara, Boccara laboratories [15]

The bone is removed in a region of the head, then the tetrode is placed into the brain, and cement is used to hold it in place. Since the tetrodes are placed in the brain of the rodents at an early age the brain grows and changes during the use of the tetrodes. To compensate for this the depth of the tetrode is re-adjusting using screws, see figure 2.3.6. This makes sure that the tetrode measures at the desired level of the brain throughout several experiments. Furthermore, the measured signals are measured against a ground screw which is placed into the skull (above the brain), as shown in figure 2.3.6.[7]

The EMG sensors differ from the previously mentioned sensors in that they are placed on the neck of the rodent and measure the electric potential generated by the muscles, meaning that they can measure muscle tension and muscle activation.

[7]More precisely it is placed in the dura-level of the brain which is below the skull, but above the brain. Elementary information about brain anatomy is provided here [13].

2.4 Design of a wireless headstages

Wireless headstages can be designed using the same Drive as is used for the wired headstages. Furthermore, the raw signal should be processed in the same way to start with, see figure 2.4.1. First $\mathbf{s}_r(t)$ is amplified, then $\mathbf{s}_a(t)$ is band pass filtered, and $\mathbf{s}_f(t)$ is consequently digitized giving $\mathbf{s}[n]$. This is then compressed by the encoder resulting in the signal $\mathbf{x}$, which is lastly transmitted as $\mathbf{t}$.

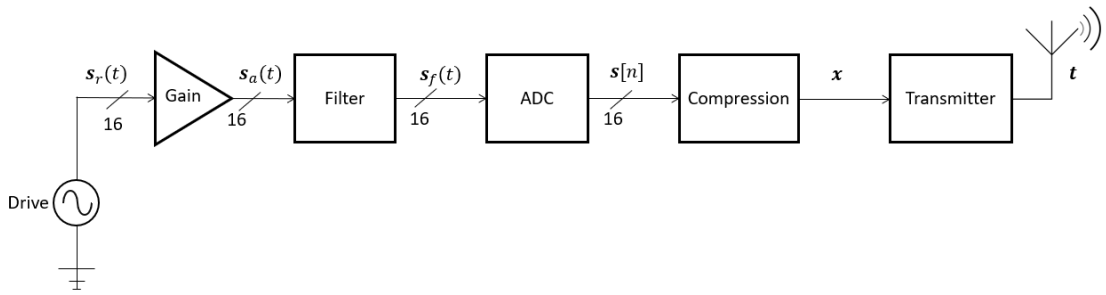


Figure 2.4.1: Block diagram of transmitter system.

At the receiver side $\mathbf{x}$ is obtained as shown in figure 2.4.2.[8] The decoder then tries to reconstruct the original signal resulting in $\hat{\mathbf{s}}[n]$.

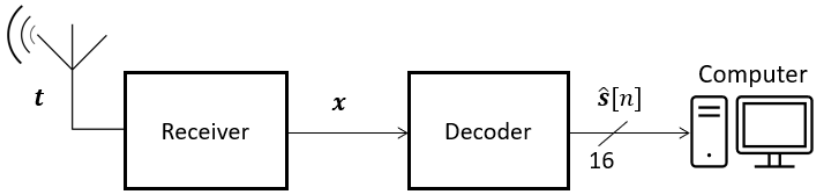


Figure 2.4.2: Block diagram of receiver system.

2.5 Electronics of the wireless headstage

Figure 2.5.1 shows one possible layout of the electronics for a wireless headstage. We will focus on this implementation from now on in order to concertize the problem.

[8] Assuming perfect transmission.

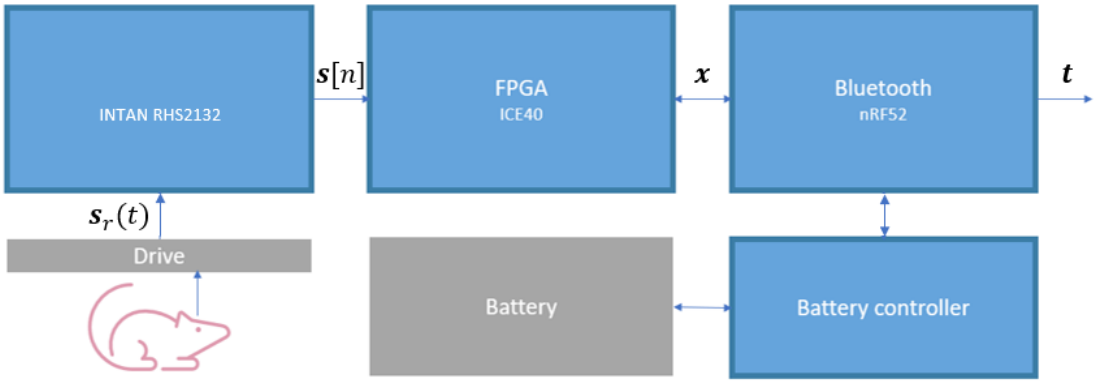


Figure 2.5.1: Block diagram of electronics for wireless headstage

The INTAN RHD2132 is connected to the Drive, and processes the analog signals [16]. It band-pass filters, amplifies, and digitizes the signal, before sending it to the ICE40 Field Programmable Gate Array (FPGA) [17]. The FPGA is where the digital signal processing, and thus the compression algorithm, will be implemented. After digital signal processing the signal is sent with Bluetooth using the Nordic nRF52 Bluetooth Low Energy (BLE) chip [18]. Additionally, there is also a battery controller unit and a battery.

INTAN RHD2132 contains a 3rd-order analog Butterworth filter and a 1st-order Butterworth low-pass filter that will be useful for band-pass filtering the signal right after acquisition. Additionally, the amplifiers that are provided in the chip are adjustable, but not individually for each channel. Meaning that all channels get the same gain. A sampling frequency of 20kHz will be used, with 16 bits allocated to each sample.

2.6 Intended Signal Use

After the signal is measured, processed, and transmitted by the headstage it is received by a computer that does post-processing. Here the tetrode signal can be filtered into high- and low-frequency parts, respectively LFP and EAP.

2.6.1 Action Potentials processing

The EAP signal consists of background noise, in addition to spikes that indicate the actual signaling between neurons. These spikes are what are of interest, and since these typically are of higher magnitude than the noise, the signal is thresholded to detect where the spikes are. Afterward, only the parts of the signal with spikes are kept. Following this, the spikes from the different channels in a tetrode are clustered using a clustering algorithm, for example, Principal Component Analysis (PCA) can be used for this.[9] Figure 2.6.1 illustrates how clustering of spikes from different neurons can look.

[9]PCA will be explained in the next section.

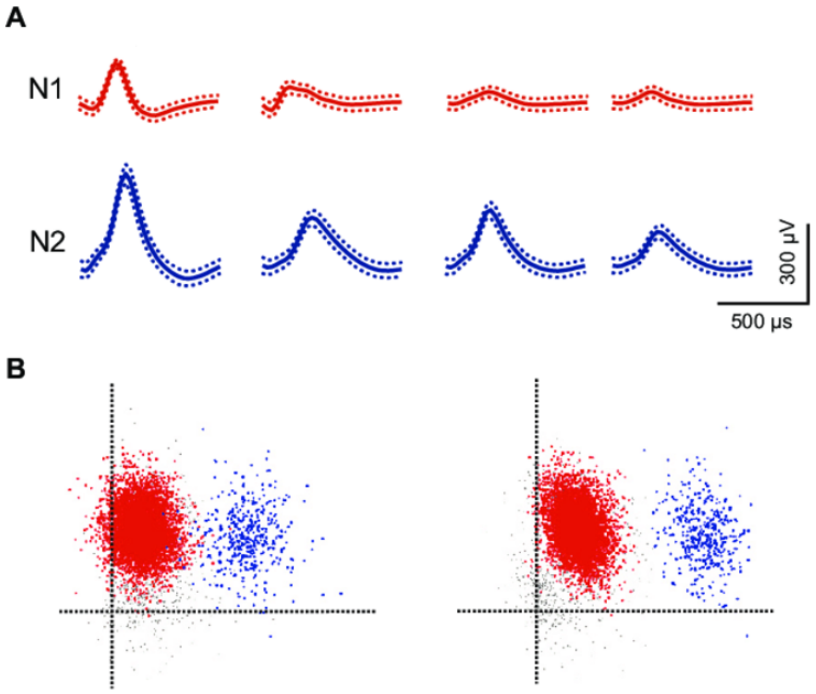


Figure 2.6.1: Illustration of AP clustering of spikes from two neurons $N1$ and $N2$. Taken from [19] and used under creative commons CC BY 4.0 license.

Since different neurons produce spikes of different shapes, the resulting clusters indicates which spikes that come from the same neuron. Additionally, the difference in amplitude can also be used to distinguish spikes from different neurons. Notice the difference in amplitude and shape for the signals from neuron one (N1) and neuron two (N2) in figure 2.3.5. To distinguish between signals coming from different directions in a plane theoretically only three channels are needed. However, the reason that we want four measurement points close to each other instead of three is firstly that four points increases a PCA algorithm's performance (whereas five does not help much), but also because one measurement point can become faulty and stop measuring. Measurement points can also become partially faulty, and thus have more noise, but still have some useful information. One example of this is shown in chapter 4.

2.6.2 Local Field Potential processing

The LFP signal is unusually broken down into different frequency bands. Delta: 0-5Hz, Theta: 6-12Hz, Beta: 10-20Hz, slow gamma 20-45Hz, medium gamma: 60-90Hz, fast Gamma: 100-200Hz. The energy in each frequency band in relation to each other can for example be used to determine which stage of sleep the rodent is in. Similarly, the EcoG signal can also be used for this. And the muscle tone that is picked up by the EMG can also give indications of sleep states. [10]

[10]For example a high theta-delta ratio indicates that the rodent is in REM sleep. In this sleep state the muscle tone is also reduced, meaning that the EMG signal also can be used as an indicator. On the other hand, a high power in the delta band indicates slow-wave sleep (deep sleep).

2.7 Signal Compression

The aim of signal compression is to encode the signal in such a way that reduces the data amount as much as possible and simultaneously allows for signal reconstruction with as little loss as possible. Lossless compression is the class of compression algorithms that theoretically allow for perfect reconstruction of the original signal. While lossy compression typically achieves higher compression at the cost of imperfect reconstruction. Two common ways of quantifying compression are compression ratio (CR) and relative compression ratio (RCR)

$$\mathrm{CR} = \frac{b}{\hat{b}}, \tag{2.1}$$

$$\mathrm{RCR} = \frac{b - \hat{b}}{b} \cdot 100, \tag{2.2}$$

where b is the number of bits used to send s, and $\hat{b}$ is the number of bits used to send $\hat{s}$. CR indicates by which factor the signal size has been reduced, while RCR gives a measure of how many percent of the original signal size has been effectively removed.

2.7.1 Nyquist-Shannon sampling theorem

The Nyquist-Shannon sampling theorem states that the sampling rate F_s must be at least twice the bandwidth of the signal B to avoid aliasing

$$B < \frac{F_s}{2}. \tag{2.3}$$

Figure 2.7.1 shows an example of the effects aliasing can have [20]. Here we see that the actual signal frequency becomes ambiguous when Fs is too low.

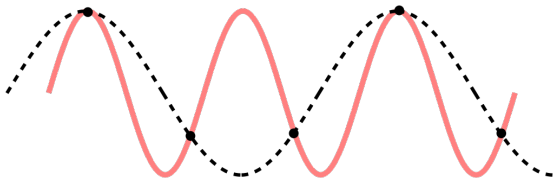


Figure 2.7.1: Example of aliasing effects. Taken from [20] and used creative commons CC BY-SA 4.0 license.

2.7.2 Compression Methods

To perform lossless compression one has to exploit the redundancies in the signal. These redundancies can be temporal, inter-channel, or in the way the bits in themself are represented. After performing an ideal lossless transform, the signal should be decorrelated in both the spatial and temporal domains and resemble noise.

2.7.2.1 Entropy coding

Entropy coding attempts to approach the lower bounds declared by Shannon's source coding theorem, which states that any lossless data compression method must have an expected code length greater or equal to the entropy of the source [21]. There are several possible entropy coding techniques, the most common being Huffman coding and Arithmetic coding. Arithmetic coding is much more computationally demanding than Huffman, and has been shown to have little improvement in compression results compared to Huffman coding when tested on EEG data [22]. A third alternative is Golomb coding, which requires very little computation and has hardware requirements lower than those of Huffman coding.

Huffman coding assigns variable length codes to input values based on frequency of occurrence. First, the probability of occurrence for different values is estimated using a dataset for training, then these probabilities are used to build a Huffman tree that determines which values get which binary code. A brief explanation of Huffman coding is included in chapter A of the appendix. When in use, the signal is first quantized, then this value is searched for and located in the Huffman tree and the corresponding code is sent. Thus, the method does not require an extensive amount of operations per sample, but needs to have the Huffman tree stored and available [23].

Golomb coding on the other hand does not require building or storing of a frequency table [24, 25]. Nevertheless, the coding is optimal if the data follows a geometric distribution, meaning that small values are much more likely than high values. In most cases this is only approximately true, in which case it performs almost as well as Huffman coding, but with lower computational requirements. Golomb-Rice coding (or Rice coding) is the subset of Golomb coding where the tunable parameter η is chosen to be a power of two. We define κ as

$$\kappa = log_2(\eta) \tag{2.4}$$

Rice coding takes $s_i[n]$, which denotes a single sample of a single channel and divides it by the integer η

$$q_i[n] = s_i[n]//\eta, \tag{2.5}$$

where // denotes integer division. The remainder $r_i[n]$ is then calculated

$$r_i[n] = s_i[n]\%\eta, \tag{2.6}$$

where % denotes the modulo operation.[11] The quotient $q_i[n]$ is then converted to unary code, while $r_i[n]$ is written in binary using κ bits. They are then concatenated such that $q_i[n]r_i[n]$ is sent. In this way, each encoded value will have a variable length, but the decoder will be able to locate where each new value starts thanks to the unary coding of $q_i[n]$. The ideal κ can be calculated as

$$\kappa = \log_2\left(\log_{10}(2) \cdot \overline{|\mathbf{s}_i|}\right). \tag{2.7}$$

[11] Since η is a power of two $q[n]$ can be calculated with a bitwise left shift operation, and $r[n]$ can be calculated using a bitwise OR operation. This makes the algorithm much less computationally demanding when implemented in hardware.

Where $\overline{|\mathbf{s}_i|}$ denotes the mean of the absolute value of $\mathbf{s}_i$ This can either be calculated adaptively, or estimated using training data.

2.7.3 Temporal Decorrelation

One simple method for temporal decorrelation is Differential Pulse Code Modulation (DPCM), which simply predicts the current sample $p_i[n]$ using the previous sample

$$p_i[n] = s_i[n-1], \qquad n = 1, ..., (N-1) \tag{2.8}$$

$$p_i[0] = 0. \tag{2.9}$$

To make this into a lossless compression method the difference between the prediction $p_i[n]$ and the true signal $s_i[n]$ is sent

$$u_i[n] = s_i[n] - p_i[n]. \tag{2.10}$$

This residual $u_i[n]$ can then be added on top of the previous sample to get the true signal at the output.

$$\hat{s}_i[n] = u_i[n] + \hat{s}_i[n-1], \qquad n = 1, ..., (N-1) \tag{2.11}$$

$$\hat{s}_i[0] = u_i[0] \tag{2.12}$$

Different transform coding methods can also be used to decorrelate the spatial dimension. Discrete Cosign Transform (DCT) is one of the most commonly used, and was shown to be the most effective out of four methods tested on LFP and EAP signals by Schmale et al. [26].[12] One way of computing the DCT coefficients is by building a DCT matrix (DCT basis) D

$$D_{l,k} = c_k \cos\left[\frac{(2l+1)k\pi}{2L}\right] \text{ where } c_k = \begin{cases} \sqrt{\frac{1}{L}} & \text{if } k = 0 \\ \sqrt{\frac{2}{L}} & \text{otherwise} \end{cases} \tag{2.13}$$

where $l = 0, 1, \ldots (L-1)$, and $k = 0, 1, \ldots (L-1)$ making an $L \cdot L$ DCT matrix [27]. Given a segment of the signal $\mathbf{s}_w$

$$\mathbf{a} = D^T \cdot \mathbf{s}_w \tag{2.14}$$

where $\mathbf{a}$ is a vector of the coefficients of that segment. The original signal can then be computed by computing the inverse

$$\hat{\mathbf{s}}_w = D \cdot \mathbf{a}. \tag{2.15}$$

Each DCT coefficient describes the contribution of a given frequency at the section of the signal we are analyzing. If most of the variation in a signal can be described using only a few DCT coefficients that signal is said to be sparse in DCT domain. This means that a signal that only consists of a few frequencies is sparse in DCT domain, and can thus be compressed with negligible loss by computing the DCT and discarding the least significant coefficients. Furthermore, one can also archive

[12] Here they test it on a method called Compressed Sensing (CS), but the knowledge is transferable.

lossless compression by transforming the signal to the DCT domain, keeping all the coefficients, and then applying entropy coding.

There are many possible transforms that have been derived, some of the most used ones being DCT, Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), and Karhunen-Loeve transform (KLT).[13] KLT is often also referred to as PCA, which will be used throughout the rest of this report. There are also several algorithms that can be used to build an optimized basis based on training data, e.g. K-SVD. [14] However, building a specialized basis needs a training set, and will only result in good compression given that the training set is sufficiently similar to the recorded data. It becomes clear how big the variation between different datasets is in chapter 4.

2.7.4 Inter-Channel Decorrelation

Since the tetrodes measure from points in space that are close to each other, the measured values will naturally be correlated. Similarly, the pairs of EcoG and EMG signals will also be correlated. It is therefore reasonable to use a method that exploits the inter-channel similarities to achieve compression. In audio signal compression one established method is M/S Stereo coding [28].[15] Given two correlated channels s_1 and s_2

$$h_{1,2}[n] = \frac{1}{2}(s_1[n] + s_2[n]), \tag{2.16}$$

and

$$o_{1,2}[n] = \frac{1}{2}(s_1[n] - s_2[n]). \tag{2.17}$$

Where the sum $h_{1,2}[n]$ and the difference $o_{1,2}[n]$ is then sent instead of the original channel values. If there is adequate similarity between the two channels the resulting $o_{1,2}[n]$ will have a lower amplitude than the original signals.[16] At the receiver end one can reconstruct the signal using

$$s_1[n] = h_{1,2}[n] + o_{1,2}[n], \tag{2.18}$$

and

$$s_2[n] = h_{1,2}[n] - o_{1,2}[n]. \tag{2.19}$$

Transform coding methods like DCT and PCA can also be used for inter-channel decorrelation. DCT will be effective given that the inter-channel variation can be described well with a limited sum of frequencies. Whilst PCA can decorrelate less obvious correlations between the channels. PCA is known to be optimal with respect to variance distribution, and outputs principle component vectors that are orthogonal [29]. The algorithm first removes the mean from the signal

$$\tilde{S}_w = S_w - \overline{S}_w, \tag{2.20}$$

[13]The details of each is beyond the scope of this report.

[14]The details of the algorithm are beyond the scope of this report.

[15]Also referred to as sum/difference coding.

[16]The output of MS Stereo can be coupled with entropy coding to get better lossless compression.

where $\bar{S}_w$ is the mean of the signal segment S_w.[17][18] Afterwards, the singular value decomposition (SVD) is computed

$$\text{SVD}(\tilde{S}_w \tilde{S}_w^T) = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T \cdot V\Sigma^T U^T = U\Sigma^2 U^T. \tag{2.21}$$

The columns of the resulting matrix U are the principal components of the signal and give the directions of most variance in descending order. While each component of the diagonal matrix Σ gives the variance of the corresponding principal component. With U we can compute

$$B = \tilde{S}_w \cdot U, \tag{2.22}$$

where B contains the PCA coefficients which can be sent, referred to as the principal components. The original signal can be reconstructed as

$$\hat{S}_w = B \cdot U^{-1} + \overline{S}_w. \tag{2.23}$$

Since most of the variation in the signal is described by the first principal components one can discard the lowest principal components without losing much information. It can also be coupled with entropy coding for lossless compression.[19]

[17] Upper case indicates that it is a matrix.

[18] The mean is removed to make sure that the direction of the non-zero mean is not interpreted as a direction of variance.

[19] It is worth noting that PCA is much more computationally demanding than the DCT, and will therefore require more energy from an embedded system compared to simpler methods.

CHAPTER

THREE

PREVIOUS WORK

This section will explore the existing literature on methods for compressing LFP and EAP signals. It will include an examination of various studies, highlighting significant contributions and discussing overarching patterns observed in the field. Chapter B of the appendix provides tables that summarize a large proportion of what has been published in the field. This analysis aims to offer insights into the current state of research and potential directions for future advancements.

3.1 Lossless compression

First, we will look at published methods that claim to be lossless. It is however worth noting that these methods still lose information to varying degrees.

SY. Park et al. (2017) achieve CR of 10.54 on EAPs and 5.35 on LFPs [30]. Furthermore, they achieve 89% reduced dynamic power consumption and state-of-the-art recording performance of 3.37 μW/Ch. The signal is first filtered into LFP and EAP bands and then sent into different ADCs and compression pipelines. However, this requires the production of an application-specific integrated circuit (ASIC). ASICs require major financial and time investments and are therefore out of the question for many projects.

O. Savolainen & T. Constandinou (2020) present a method for temporal decorrelation using a single neuron linear neural network (LNN) [31]. This is then followed by pre-trained Huffman coding. Their results from testing on different resolutions from 5 to 13-bit show that Delta encoding achieves similar results as LNN.

Y. Khazaei et al. (2020) present a method for spatial decorrelation. They take the inter-channel mean, remove this from all channels, send the mean as a separate channel, and adaptively allocate the bits. This method will be described further in chapter 5.

A. Cuevas-Lopez et al. (2022) use DPCM and Huffman coding. Additionally, they remove the three least significant bits, meaning that information is lost.[1] They

[1]Their reasoning behind this choice can be found in the article and is connected to the

have also implemented a version of Huffman coding that only needs to allocate 2^{n+1} bits for its Huffman tree given n bit samples (2^{2n} is standard). Furthermore, they use two tricks. The first is to only perform Huffman coding of the most significant bits. The second is to omit the sign bit when the sample value is zero. [2]

As opposed to LFP/EAP compression lossless EEG compression has a bit more published work, and can therefore be insightful to research given the signal similarities, as described in section 2.3. The following papers give a good overview of existing lossless methods used for EEG compression [32, 22, 33], published in 2015, 2018, and 2018 respectively. Additionally, Y. Wongsawat el. al. (2006) shows how DPCM, PCA, DCT, and Huffman can be combined for effective compression [34]. They use DPCM to reduce DC offset, PCA for inter-channel decorrelation, DCT for temporal correlation, and Huffman coding for entropy coding. "Although, the computational complexity of the proposed coder is highly consumed in the process of spatial decorrelation by calculating the eigenvalues and eigenvectors of PCA and by factorizing PCA matrix to form its reversible structure, this complex stage leads to very simple and effective temporal decorrelation (stereo IntDCT-IV) and entropy coding (Huffman)" [34].

3.2 Lossy compression

Looking at the work published on lossy compression of LFP and EAP signals it can be observed that it is typical to separate the signal into high- and low-frequency bands for lossy compression. This is because the LFP and EAP signals have distinctly different characteristics and intended use, as described in chapter 2. After separation, the LFP is typically down-sampled, since the lower frequencies require a lower sampling rate to fulfill the Nyquist-Shannon sampling theorem. Spike detection is commonly done on the EAP signal. This is done by thresholding the EAP signal and only keeping the parts of the signal where spikes are detected. The value of this threshold is commonly set using different adaptive methods. As mentioned, spike detection on the EAP signal is done in post-processing if it is not done locally, see section 2.6. However, local spike detection can result in more errors in the spike detection, meaning more spikes lost, especially if there are overlapping spikes.[3] Furthermore, some papers extract features from the spikes after detection, which is reasonable as the spikes are eventually clustered based on spike features in post-processing. However, methods like PCA can be used to extract higher-order features that can give better classification results compared to using a limited number of hand-crafted features. Additionally, it is useful to have the full EAP signal for research and publication purposes.

As an example G. Bilodeau et al. (2021) first filter the signal into LFP and EAP bands, and then downsample the LFP and do spike thresholding on the EAP [35].

inevitable noise floor of the ADC they use.

[2] The algorithm was implemented on an FPGA that has half the amount of look-up tables (LUTs) as the ICE40 and required less than 3 mW. Additionally, the system was implemented using the same INTAN RHD2132 acquisition chip as we use.

[3] This is when two spikes are recorded at the same time, and thus overlap.

Following this they then use DWT on the spikes for compression.[4] With this they get a CR of 9.38 on 32 channels of data, which achieves 30-60 minutes of activity on a 100mAh battery.

One method that is widely used, and is getting a lot of attention is compressed sensing (CS). This method has very low hardware demand and is easy to implement. The caveat is that it requires solving an under-determined system of equations on the reconstruction side, and is prone to producing artifacts. B. Sun and W. Zhao (2021) provide a very good review of the most successful attempts to implement CS on LFP, EAP, and EEG signals [36].

B. Sun et al. (2021) presented a CS method using a training-free deep neural network for re-construction [37]. W. Zhao et al. (2020) presented a CS method that exploits the a-priori information we have of the frequency bands that the LFP and EAP signals for re-construction [38]. Whilst B. Sun et al. (2021) presented a CS method that uses a new non-convex optimization method for re-construction [39]. These are some of the most promising attempts of CS compression on LFP and EAP signals. It is also worth noting that CS has its limitations, for example, "overlapping spikes are excluded and there is limited success for the signal reconstruction of overlapping spikes using CS methods " [36].

Apart from CS, one of the most successful methods presented is deep compressive autoencoders by T. Wu et al. (2018). This uses machine learning to compress their signal with a CR of 20 to 500.[5] Furthermore, the method has been made with hardware implementation in mind. The downside is that it requires data for training, meaning that it will be more adapted to the data it has been presented. This can be especially problematic for neural data since there can be a lot of variability in recorded neural data, both within and across measurement systems, which will become clear in chapter 4 of this report.

[4]They use the INTAN RHD2132 and use an FPGA with 15k LUTs for the compression algorithm.

[5]In this compression range the method has a Signal to Noise Distortion Ration of 14 dB to 8 dB.

CHAPTER

FOUR

DATASET

This chapter presents the datasets that have been used for testing and provides a frequency analysis of some of the data. It also gives useful background information to keep in mind and occasionally provides some comments on the presented data.

All of the data was recorded at Boccara laboratories [15], and were measured using wired headstages surgically implanted into the brains of rats. Drives with three tetrodes, two EcoG channels, and two EMG channels, resulting in 16 channels, were used. All in all the datasets have 60 seconds of recording in total.

When using a wired measurement system, as shown in figure 2.2.2, the gain of the different channels can be set individually, as described in section 2.2. And since the brain is constantly changing and there might develop scar tissue around the sensors, it is typical to set the gain for each channel before each experiment. Which makes sure that the signal from the channels have as high magnitudes as possible without being saturated.[1] As a result of this, the unit on the y-axis of the plots that will be shown is volts of arbitrary scale.

The signals can vary greatly depending on several factors. Such as how successful the surgery where the Drive is mounted goes, tetrode placement relative to neurons of interest, scar tissue development, the inherent noise in the brain, and the behavior of the rodent at that time. Moreover, the datasets are a mixture of sleep and wake rodent behavior. As a result, we can see great variation in the signals from the different datasets. This is part of what makes the compression of neural signals difficult.

Which datasets to use for method selection, training, and testing was chosen arbitrarily, and not after analyzing the data. Dataset 3 was used as a test set and was kept hidden away during all testing and method selection. Dataset 2 was used for training, meaning that it was used to train parameters or models that needed training data, e.g. Huffman coding or Rice coding. Using the same data for training as for compression would have given artificially high compression results.

[1] The higher the magnitude, the more bits are used for that signal, and thus the better the resolution of the signal.

4.1 Dataset 1

Contains 10 seconds of data sampled at 48kHz. Figure 4.1.1 shows the LFP/EAP channels, each column shows the signals from one tetrode. Figure 4.1.2 shows the EcoG channels in the first column and the EMG channels in the second. Similarly, figure 4.1.3 and 4.1.4 show the frequency power spectrum of the signal. Here the y-axis indicates the magnitude of the power in the different frequencies and is not in logarithmic scale. Plots of the frequency power spectrum with the y-axis in logarithmic scale are included in chapter I of the appendix.

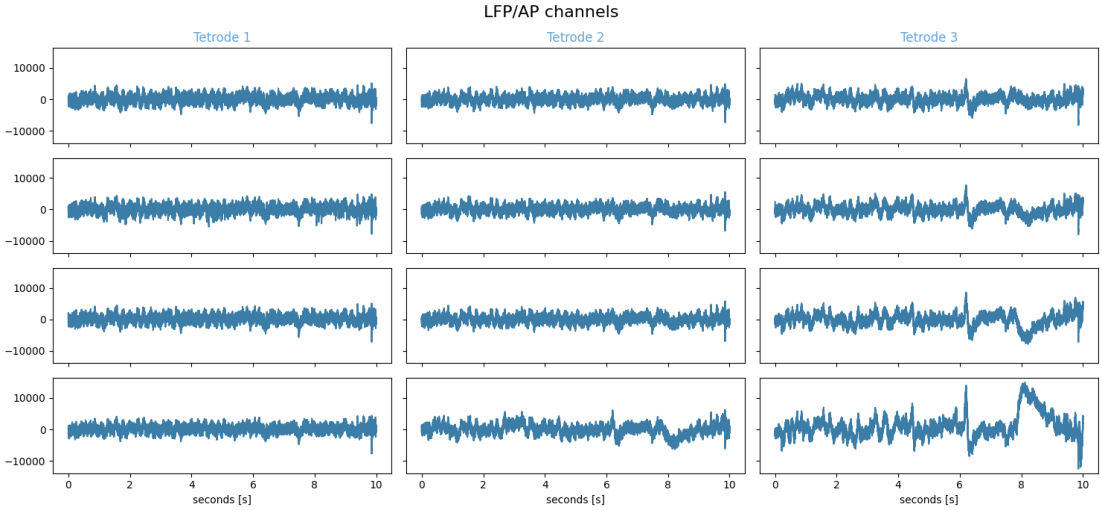


Figure 4.1.1: LFP/EAP channels of dataset 1.

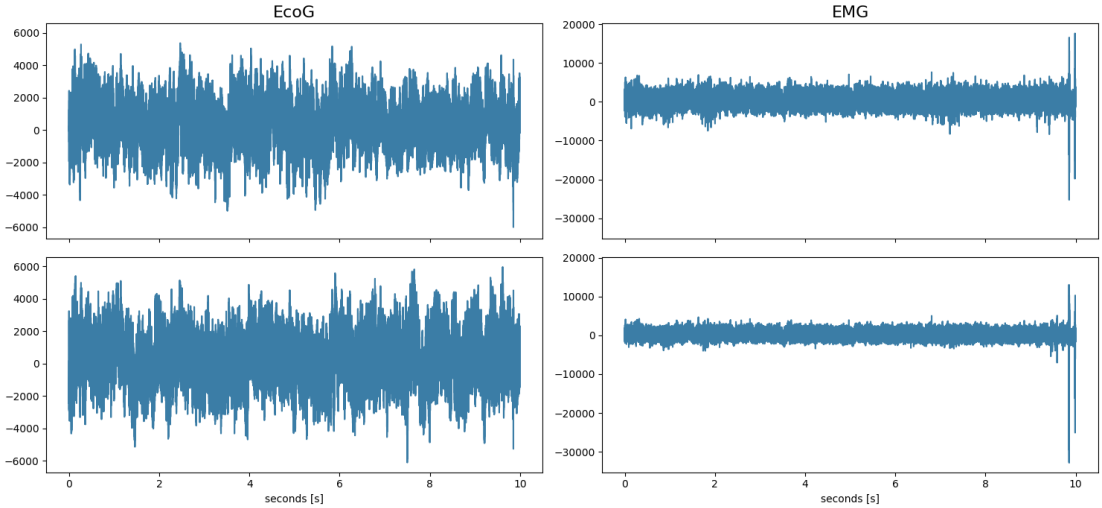


Figure 4.1.2: EcoG and EMG channels of dataset 1.

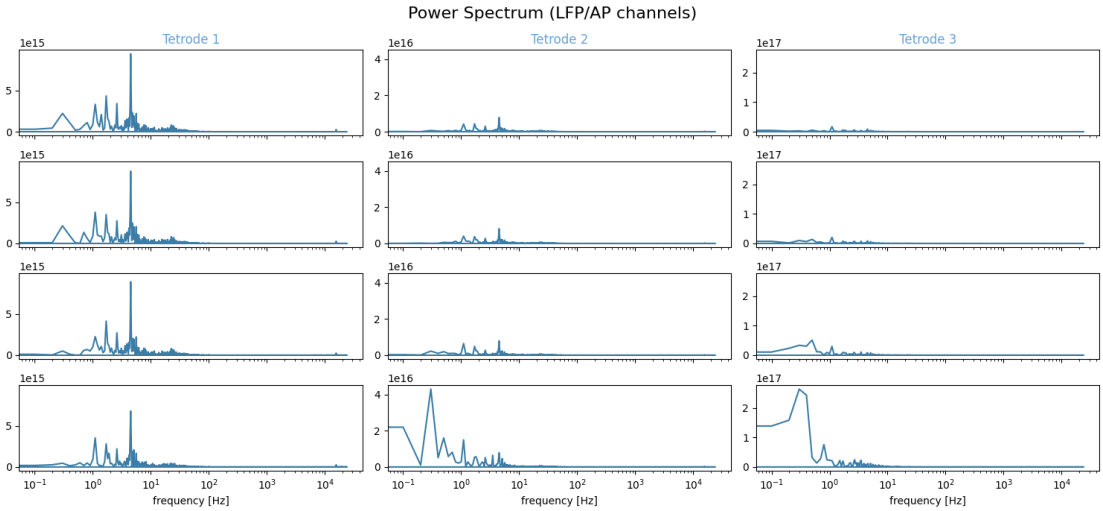


Figure 4.1.3: Power spectrum of LFP/EAP channels of dataset 1.

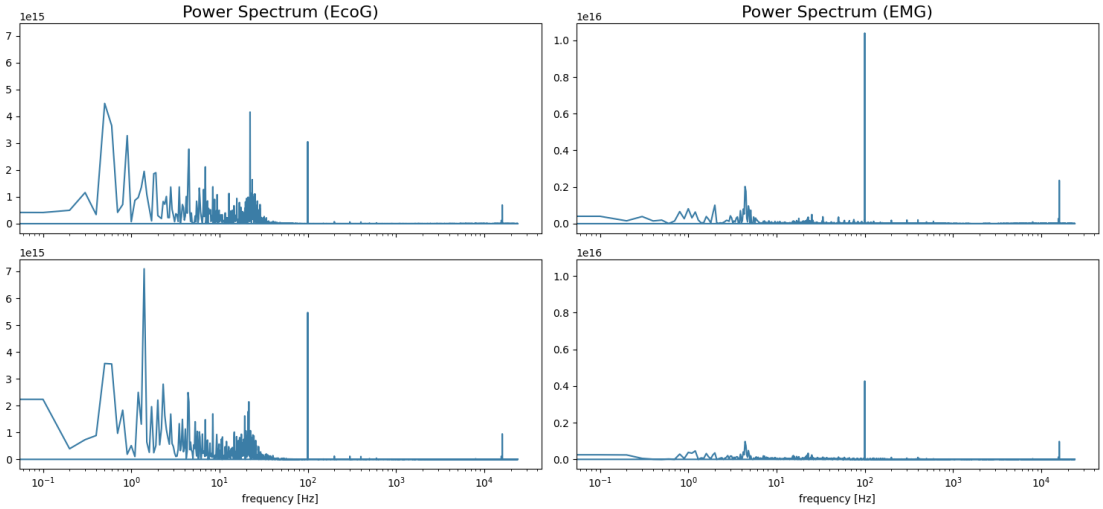


Figure 4.1.4: Power spectrum of EcoG and EMG channels of dataset 1.

4.2 Dataset 2

Contains 20 seconds of data sampled at 24kHz. Figure 4.2.1 shows the LFP/EAP channels, each column shows the signals from one tetrode. While figure 4.2.2 shows the EcoG channels in the first column and the EMG channels in the second. Plots of the frequency power spectrum of the signal with and without logarithmic y-axis are included in chapter I of the appendix.

We can observe that the signal saturates on several of the tetrode channels, and also on the EMG channels.

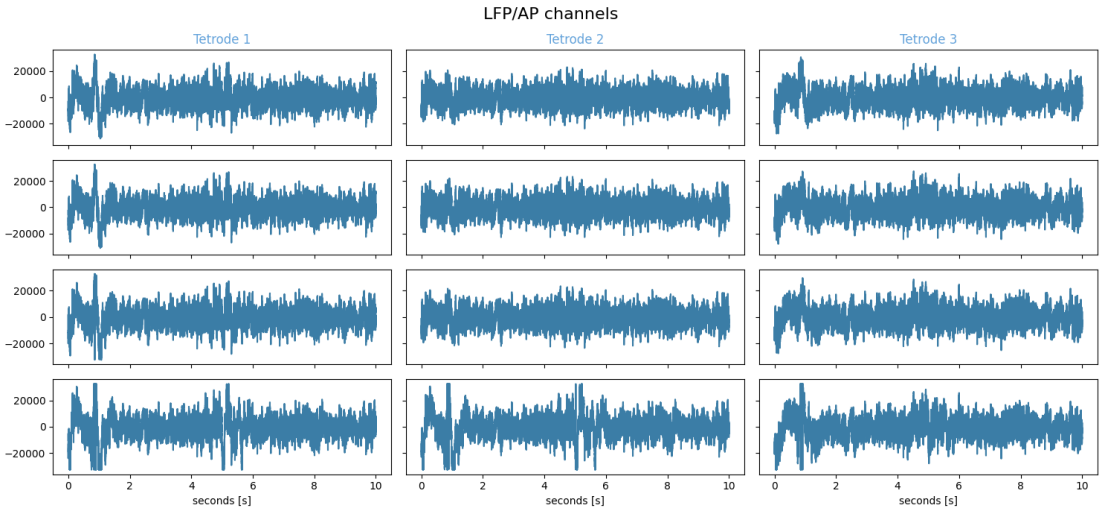


Figure 4.2.1: LFP/EAP channels of dataset 2.

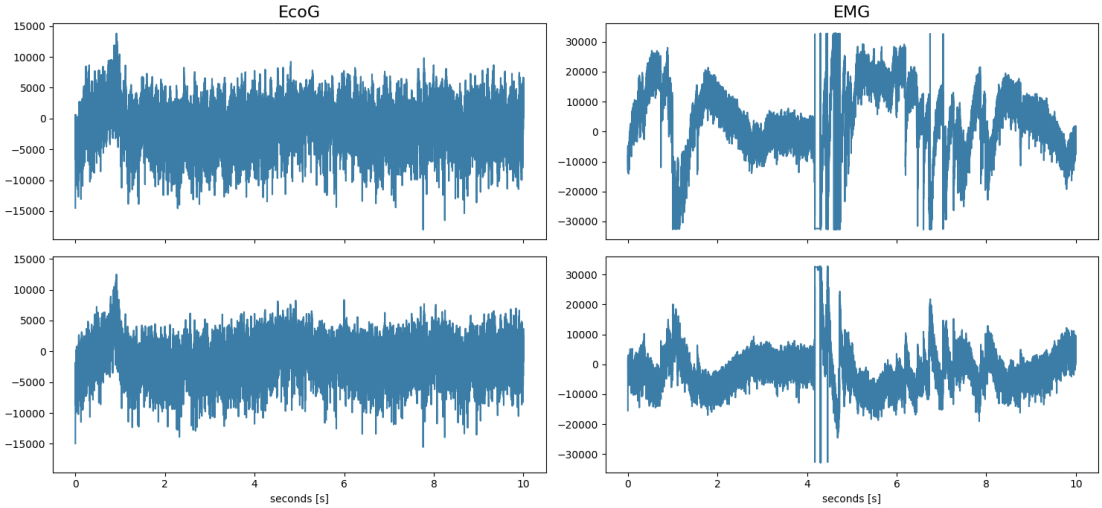


Figure 4.2.2: EcoG and EMG channels of dataset 2.

4.3 Dataset 3

Contains 20 seconds of data sampled at 24kHz. Figure 4.3.1 shows the LFP/EAP channels, each column shows the signals from one tetrode. While figure 4.3.2 shows the EcoG channels in the first column and the EMG channels in the second. Figure 4.3.3 shows the frequency power spectrum of the EcoG and EMG channels with a logarithmic y-axis. More plots of the frequency characteristics of dataset 3 are included in chapter I of the appendix.

In figure 4.3.2 we see that the EcoG channel in the top left corner resembles noise. We can also see that the frequency band of this channel is relatively flat in figure 4.3.3.

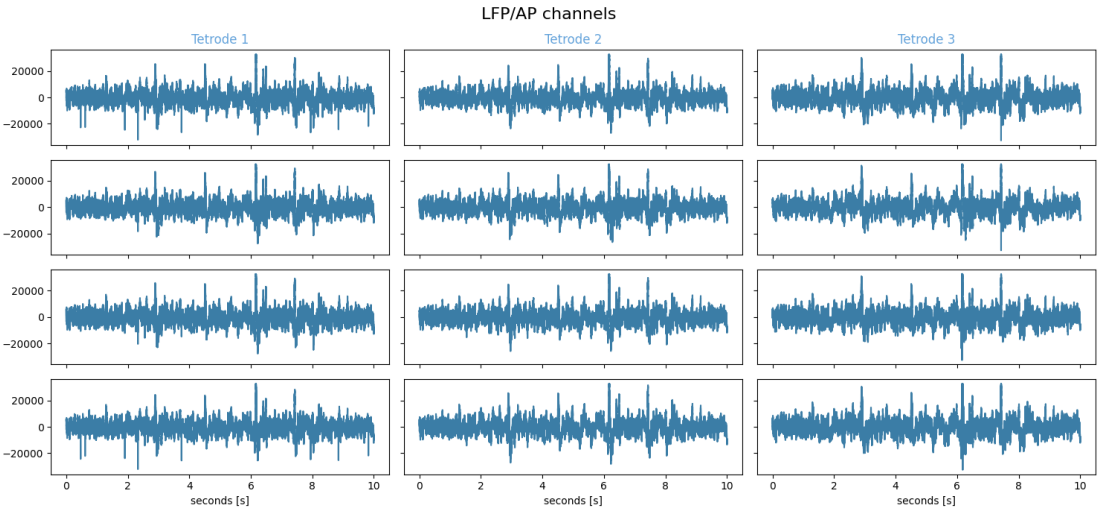


Figure 4.3.1: LFP/EAP channels of dataset 3.

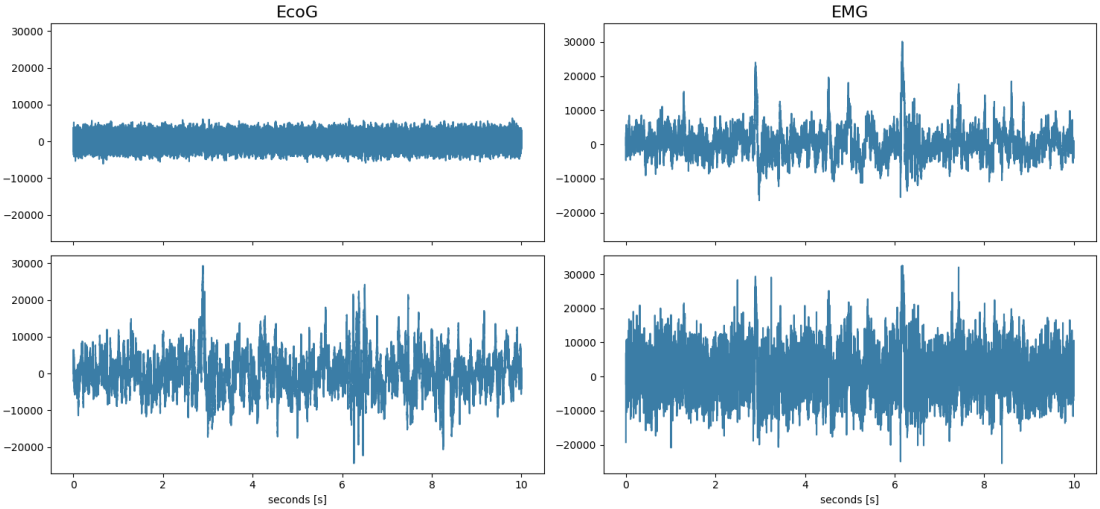


Figure 4.3.2: EcoG and EMG channels of dataset 3.

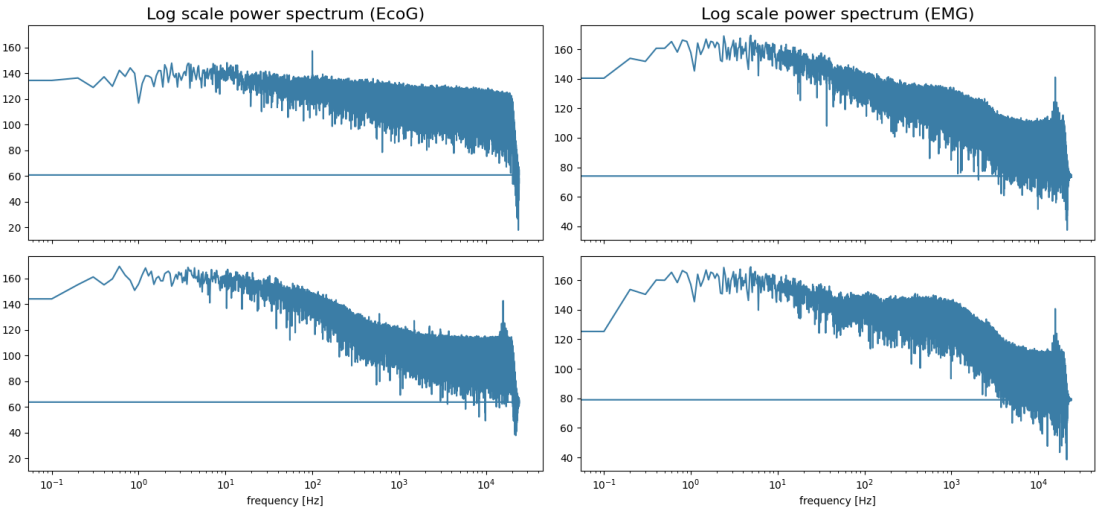


Figure 4.3.3: Power spectrum of EcoG and EMG channels of dataset 3.

4.4 Dataset 4

Contains 10 seconds of data sampled at 48kHz. Figure 4.4.1 shows the LFP/EAP channels, each column shows the signals from one tetrode. While figure 4.4.2 shows the EcoG channels in the first column and the EMG channels in the second. Similarly, figure 4.4.3 and 4.4.4 show the frequency power spectrum of the signal. Here the y-axis indicates the magnitude of the power in the different frequencies and is not in logarithmic scale. The frequency power spectrum with logarithmic y-axis is included in chapter I of the appendix.

In this dataset we see that the tetrode channels have a dominant frequency component compared to the others, and this frequency component is not as prominent in the EcoG and EMG channels. It is not known whether this is typical signal behavior or whether it is the result of an error in the experimental setup.[2]

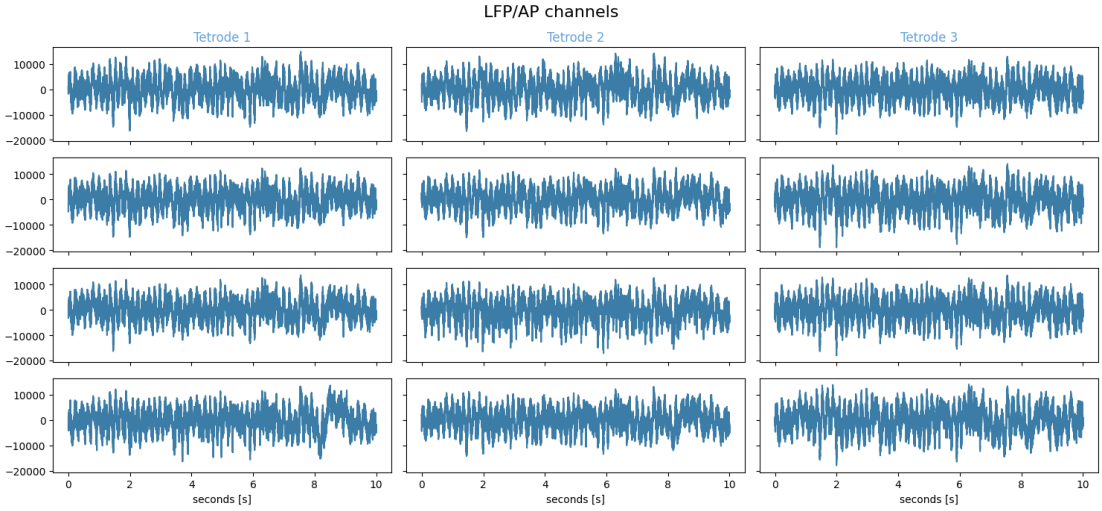


Figure 4.4.1: LFP/EAP channels of dataset 4.

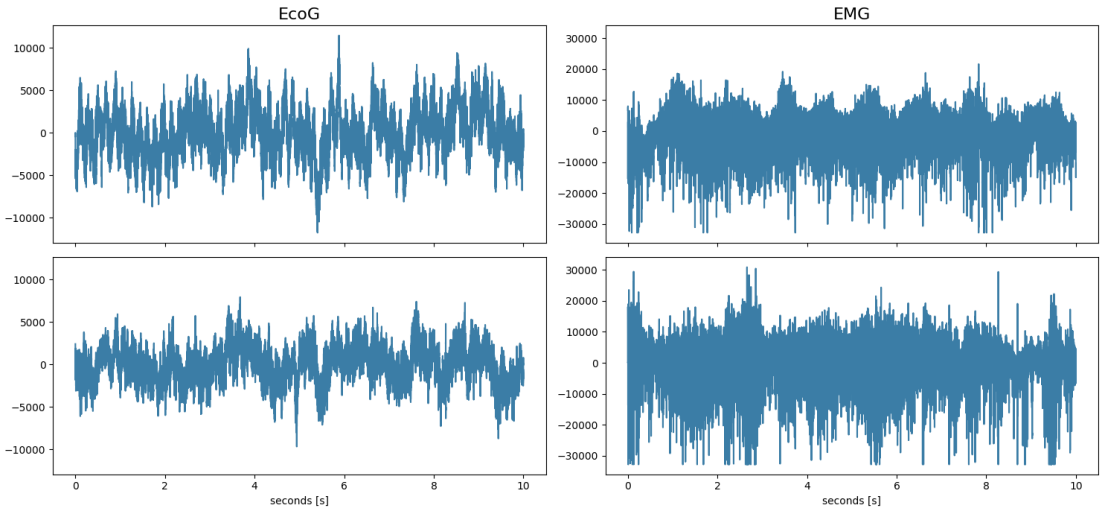


Figure 4.4.2: EcoG and EMG channels of dataset 4.

[2]This could be resolved through contact with the owners of the datasets but was not done due to time limitations.

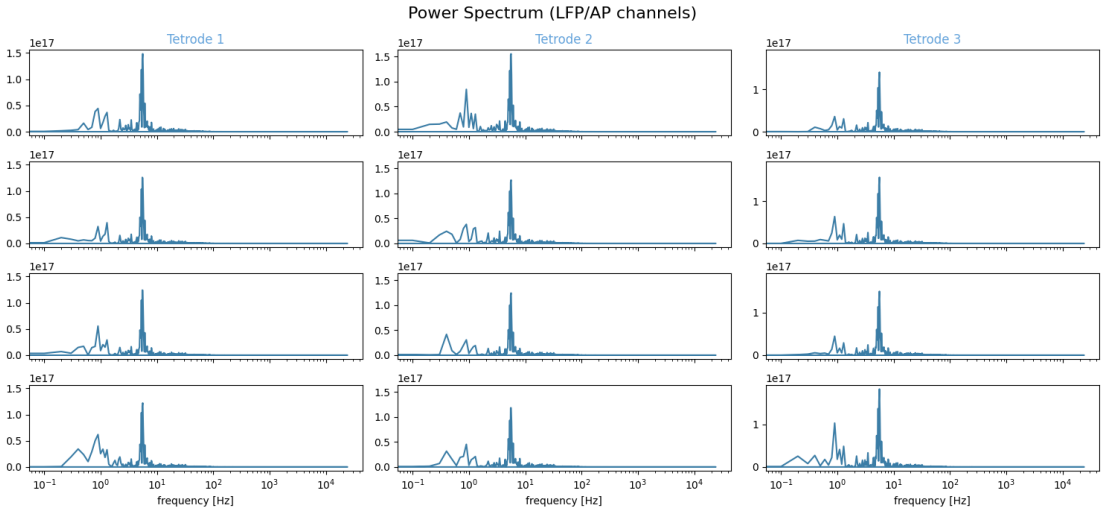


Figure 4.4.3: Power spectrum of LFP/EAP channels of dataset 4.

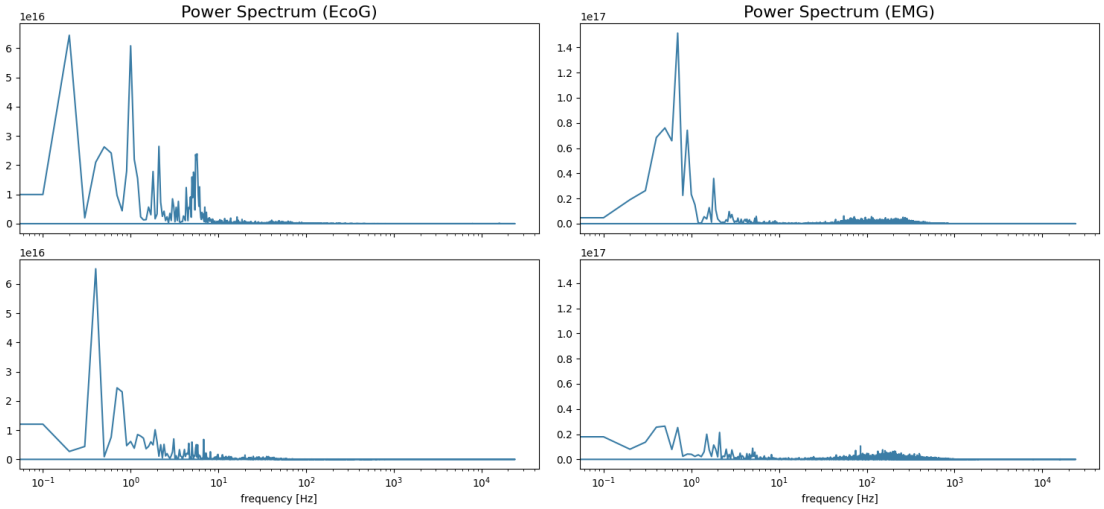


Figure 4.4.4: Power spectrum of EcoG and EMG channels of dataset 4.

CHAPTER

FIVE

METHODS

This chapter describes the implementation of some of the compression methods that will be tested and compared. It also presents some pre-processing steps that are taken to make the datasets that are used more similar to how the data from a wireless headstage might look.

All methods were implemented in Python. The GitHub repository with all the code is linked to in chapter C of the appendix. The appendix also includes pdfs of Python notebooks that illustrate more in-depth what the effects of the different methods are.

5.1 Pre-processing

The data is first band-pass filtered using a 3rd-order Butterworth low-pass filter, and a 1st-order Butterworth high-pass filter. Characteristics of this filter are illustrated in chapter D of the appendix. The bandwidth is set to 0.1 to 5000Hz. Afterwards, the signal is down-sampled to 20 kHz.[1] These steps are taken to make the signal resemble signals acquired from the wireless headstage design shown in figure 2.5.1 more closely.

5.2 FIR filtering

Finite Impulse Response (FIR) filtering can be implemented by taking the temporal mean of the four past samples of a signal

$$f_i[n] = \frac{1}{4}\sum_{k=0}^{3} s_i[n-k], \qquad n = 0, \ldots (N-1), \tag{5.1}$$

$$s_i[-k] = 0, \qquad k = 1, 2, 3 \tag{5.2}$$

[1] Low-pass filtering before down-sampling makes sure that the down-sampling does not cause aliasing.

where $f_i[n]$ is the temporal mean of $s_i[n]$ and it's 3 past samples.[2] The original signal can be retrieved by simply adding the past values to the current one

$$\hat{s}_i[n] = \sum_{k=0}^{3} f_i[n-k], \tag{5.3}$$

Additionally, every fifth sample is sent as its original, unprocessed.[3] This makes sure that potential errors are prevented from propagating.

This method low-pass filters the signal and removes some of the high-frequency variation. It is not designed to compress in itself, but rather to allow compression algorithms that follow it to perform better.

5.3 MS Quadro

This method builds upon MS Stereo but is specially adapted to 16-channel neural data with three tetrodes, two EcoG channels, and two EMG channels. Given four correlated channels $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ and $\mathbf{s}_4$

$$z_1[n] = \frac{1}{4}(s_1[n] + s_2[n] + s_3[n] + s_4[n]), \tag{5.4a}$$
$$z_2[n] = \frac{1}{4}(s_1[n] - s_2[n] - s_3[n] + s_4[n]), \tag{5.4b}$$
$$z_3[n] = \frac{1}{4}(s_1[n] - s_2[n] + s_3[n] - s_4[n]), \tag{5.4c}$$
$$z_4[n] = \frac{1}{4}(s_1[n] + s_2[n] - s_3[n] - s_4[n]), \tag{5.4d}$$

where $z_1[n]$ is the inter channel mean, and $z_2[n]$, $z_3[n]$ and $z_4[n]$ are difference signals. The method will result in three low-magnitude difference signals if the channels are similar. The original channel values can then be retrieved by solving the following set of equations

$$\hat{s_1}[n] = \frac{1}{4}(z_1[n] + z_2[n] + z_3[n] + z_4[n]), \tag{5.5a}$$
$$\hat{s_2}[n] = \frac{1}{4}(z_1[n] - z_2[n] - z_3[n] + z_4[n]), \tag{5.5b}$$
$$\hat{s_3}[n] = \frac{1}{4}(z_1[n] - z_2[n] + z_3[n] - z_4[n]), \tag{5.5c}$$
$$\hat{s_4}[n] = \frac{1}{4}(z_1[n] + z_2[n] - z_3[n] - z_4[n]). \tag{5.5d}$$

[2]The fraction $\frac{1}{4}$ is particularly appropriate as it is a power of 2. It can therefore be implemented as a left-shift operation in hardware, which simplifies the hardware implementation as opposed to an arbitrary division.

[3]This can be implemented in hardware with a simple counter and a multiplexer.

For channels that are correlated in pairs the MS Stereo method can be used. In its implementation, the above-presented method was used on the tetrode channels, while MS Stereo was used on the EcoG and EMG channels.

5.3.1 Channel Mean Removal

This method was taken from [40]. The mean of all channels is found

$$\overline{s}[n] = \frac{1}{16}\sum_{i=1}^{16} s_i[n]. \tag{5.6}$$

This is then removed from all channels

$$m_i[n] = s_i[n] - \overline{s}[n], \qquad i = 1, ..., 16, \tag{5.7a}$$
$$m_{17}[n] = \overline{s}[n]. \tag{5.7b}$$

Each output channel $m_i[n]$ is then sent. The original channel values can then be re-constructed

$$\hat{s_i}[n] = m_i[n] + m_{17}[n], \qquad i = 1, ..., 16. \tag{5.8}$$

5.3.2 Temporal DCT

The temporal DCT coefficients were calculated using equation 2.14 by taking the 500 past samples for one channel at a time, $w = [500, 1]$

$$\mathbf{a}_{[500,1]} = D^T \cdot \mathbf{s}_{[500,1]} \tag{5.9}$$

where D is of size 500x500.

5.3.3 Spatial DCT

The spatial DCT coefficients were calculated using equation 2.14 by taking one sample form each channel at time, $w = [1, 16]$

$$\mathbf{a}_{[1,16]} = D^T \cdot \mathbf{s}_{[1,16]} \tag{5.10}$$

where D is of size 16x16.

5.3.4 PCA

The PCA was taken of 500 sample windows across all channels, resulting in 500x16 sample windows, $w = [500, 16]$

$$\mathbf{b}_{[500,16]} = \tilde{\mathbf{s}}_{[500,16]} \cdot U \tag{5.11}$$

where U is of size 16x16. [4][5]

5.3.5 Huffman Coding

The Huffman tree was built using a training set. This Huffman tree was then used to encode the signals of the other datasets.[6]

5.3.6 Adaptive Rice Coding

For Rice coding κ can be calculated using a training set, similar to the case of Huffman coding. However, since the calculation of κ is reasonably simple, one can also develop an adaptive calculation of κ to increase compression results. The initial κ is first calculated using all of the data in a training set, and then it is adaptively updated using

$$\kappa = \alpha \log_2(\log_{10}(2) \cdot \overline{|s_{[40,1]}|}) + (1 - \alpha)\kappa \tag{5.12}$$

where the segment size $w = [40, 1]$ means 40 past samples of one channel at a time. Furthermore, α is a tunable parameter. Rice coding is then performed on the next 40 samples of s using the κ that was calculated.[7] The κ is sent at the start of each 40 sample window using 4 bits to make sure that the right κ is used when decoding.

[4]The mean $\overline{s}$, in equation 2.20, is computed across each channel, resulting in a vector of length 16.

[5]From equation 2.23 it is clear that both U and $\overline{s}$ need to be known by the receiver, meaning that they should be sent in addition to the coefficients with every window. This was not implemented, making the results from PCA a bit better than they should be. Since U is 16x16 elements and $\overline{s}$ is 16 elements, the total amount of extra values would be 272 compared to the 8000 elements in each PCA 500x16 window, which would not make a too drastic difference.

[6]Adaptive Huffman coding was not developed since this would require a significant amount of computation in its implementation.

[7]In this way calculations are reduced compared to calculating κ for each sample.

CHAPTER

SIX

RESULTS

In this chapter, the results from different compression methods will be presented. Spatial, temporal, and entropy coding methods will be compared against each other to start with, and then results from different combinations of methods will be shown. From these results, the best methods will be tested further, and finally, the method with the best results will be selected.

Datasets 1 and 2 were used to test and compare different methods. While dataset 4 was used to get less biased results on the method with the best results from the previous testing. Additionally, dataset 3 was used for training and was therefore not used for testing.

6.1 Entropy testing

Table 6.1.1 shows the results from testing Rice coding with pre-trained parameter κ, adaptive Rice coding with $\alpha = 0.5$, and pre-trained Huffman coding. The initial κ value for the adaptive Rice coding was set using pre-training.

Method	RCR - Dataset 1	RCR - Dataset 2
Rice Coding (pre-trained)	19.41%	12.46%
Adaptive Rice Coding ($\alpha = 0.5$)	**25.91**%	**14.58**%
Huffman coding (pre-trained)	17.55%	10.55%

Table 6.1.1: Results from testing entropy coding methods.

From table 6.1.1 we see that Rice coding performs better than the Huffman coding. This can be explained by the fact that the datasets are quite different from each other, see chapter 4. It seems that a Huffman tree built for dataset 3 is not optimal for datasets 1 and 2. This difference in the datasets also explains why the results from datasets 1 and 2 differ by this extent, and why this difference is most pronounced on the Huffman coding and least pronounced on the adaptive Rice coding. We can also see that adaptive Rice coding performs significantly better than pre-trained Rice coding, thus the adaptive adjustment is a successful

modification to the method.

6.2 Spatial testing

Table 6.2.1 shows the results from testing different methods that do spatial decorrelation on the 16-channel data. All methods were followed by adaptive Rice coding using $\alpha = 0.5$. The different methods were implemented as described in chapter 5.

Method	RCR - Dataset 1	RCR - Dataset 2
MS Stereo	32.38%	23.87%
MS Quadro	35.52%	28.09%
DCT (1x16)	27.81%	18.55%
PCA (500x16)	**39.58**%	**33.36**%
Channel Mean removal	29.96%	20.37%

Table 6.2.1: Results from testing spatial decorrelation methods. All methods were followed by adaptive Rice coding with $\alpha = 0.5$.

From table 6.2.1 we can notice that PCA gives very good results. This is reasonable since PCA finds the ideal principal components that are entirely decorrelated. Chapter E of the appendix illustrates how much information lies in the different coefficients when using PCA on the whole signal and just the tetrode channels. The results from spatial DCT indicate that the variation between the different channels is not particularly sparse in frequency domain. Which makes sense when looking at the data presented in chapter 4. From looking at the data it is reasonable to attribute these poor results to the big difference between the tetrode signals, EcoG signals, and EMG signals. Chapter F of the appendix illustrates how much better DCT works when only performed on the tetrode signals, and also illustrates more in-depth the effects of spatial DCT on the signal.[1]

One can also discuss whether Rice coding is particularly well suited for coding the output of PCA and DCT. In principle, given that PCA and DCT perform well, they should output few large coefficients that constitute the majority of the variance, and many small coefficients. Rice coding would then automatically use few bits on the small coefficients and many on the few large coefficients, resulting in effective compression. However, one could also have used methods like adaptive bit allocation instead, which might have given better compression results.

[1] One solution could be to only compute the DCT coefficients on the tetrode channels and process the EcoG and EMG channels differently.

6.3 Temporal testing

Table 6.3.1 presents the result after testing DPCM and temporal DCT followed by adaptive Rice coding with $\alpha = 0.5$.

Method	RCR - Dataset 1	RCR - Dataset 2
DPCM	39.70%	33.39%
DCT (500)	**41.12**%	**34.94**%

Table 6.3.1: Results from testing spatial decorrelation methods. All methods were followed by adaptive Rice coding with $\alpha = 0.5$.

The results in table 6.3.1 indicate that temporal DCT is better suited for the data than spatial DCT. Looking at the frequency plots of datasets 1 and 2 in section 4 we see that a handful of frequencies are more prominent than the rest. This results in few large DCT coefficients and many small, which in turn allows for good Rice coding compression. Chapter G of the appendix illustrates more in-depth the effect of temporal DCT on the signal.

6.4 Combination testing

Table 6.4.1 shows the results from testing different combinations of the methods that have been presented, here all methods have been given letter codes to make the table more compact. The bottom part of table 6.4.1 shows which code corresponds to which method.

Method	RCR - Dataset 1	RCR - Dataset 2	Avg.
DE	44.82%	40.66%	42.74%
DJ	47.56%	43.68%	45.62%
DG	42.22%	38.43%	40.33%
DH	44.19%	40.12%	42.16%
DM	42.90%	38.92%	40.91%
FE	46.21%	41.23%	43.72%
FG	43.20%	38.96%	41.08%
FM	43.98%	39.40%	41.69%
FHD	42.48%	37.74%	40.11%
DFH	45.66%	40.97%	43.32%
DHF	45.57%	41.17%	43.37%
DEF	46.40%	41.53%	43.97%
JF	48.89%	44.41%	46.65%
DJF	**49.07**%	**44.67**%	**46.87**%

Short Hand Notations	
C	FIR filter
D	DPCM
E	MS Stereo
F	DCT (temporal)
G	DCT (spatial)
H	PCA
J	MS Quadro
M	Channel Mean Removal

Table 6.4.1: Results from testing different combinations of the presented methods and short hand notation for different methods. All combinations were followed by adaptive Rice coding with $\alpha = 0.5$.

From the first portion of table 6.4.1 we can observe that adding DPCM (D) to the start of the spatial methods improves all of the results. Then we can notice that adding D to the start makes MS Stereo (E) and MS Quadro (J) perform better than PCA (H), $DH < DE < DJ$. One reason can be that D results in a signal that is closer to zero centered, meaning that E and J get lower difference values when taking the difference between the channels. On the other hand, H might not benefit from the D transform as much since it does not necessarily make the different channels more correlated. It is even possible that D makes the channels less correlated since the main correlation between the channels comes from the LFP signal, which is made less pronounced by D. Chapter H of the appendix in-

cludes plots of how the output signal after D looks.[2] In the next section of table 6.4.1 we observe that adding temporal DCT (F) to the start of spatial methods improves the results even more than D does.

In the last section of table 6.4.1 we can observe how the same methods in different order can give fairly different results. Combining temporal DCT, PCA, and DPCM in different orders gives different results, where stating with DPCM and ending with DCT (*DHF*) gives the best results compared to *FHD* and *DFH*. Intuitively it makes sense to start with the simplest method and end with the most complex one. However, this should have resulted in the best results for *DFH*.

6.5 FIR filtering

The three best combinations from table 6.4.1 were tested again with the FIR filter presented in chapter 5 included at the start. These results are shown in table 6.5.1.

Method	RCR - Dataset 1	RCR - Dataset 2	Avg.
CJF	51.09%	45.61%	48.35%
CDJF	51.28%	45.97%	48.63%
CDJ	**51.83**%	**46.86**%	**49.35**%

Table 6.5.1: Results from testing the three best methods in 6.4.1 again with the FIR filter at the start. All combinations were followed by adaptive Rice coding with

Table 6.5.1 indicates that FIR filtering before the three best methods improves them even further. This can be because the result after FIR filtering has a slower variation, which in turn allows DPCM to predict the next sample more accurately using the previous sample. Thus, using FIR filtering before DPCM results in better overall results at little computational cost. We also see that *CDJ* performs the best out of all combinations. This indicates that MS Quadro is a good method for compression of multichannel neurological data and can outperform more complex and computationally demanding methods like PCA.

6.6 Method with best results

As mentioned the method; FIR filter + DPCM + MS Quadro + Adaptive Rice Coding (*CDJ*) has yielded the best results. To get more reliable results on this method we test how well it does on the previously hidden dataset 4. This gives a RCR of **45.35**% (equivalent to CR = 1.83). With this result, it is reasonable to assume that the method will give around 40% to 50% compression on new unseen data.

[2]It also includes more in-depth analyses of other methods.

Figure 6.6.1 shows the original and reconstructed signal when using *CDJ* and rounding the signal to remove decimal numbers before Rice coding, on dataset 1. The top plot shows 0.8 ms of data (20 samples) while the bottom plot shows the whole signal with 20 seconds of data.

Figure 6.6.1: Original signal compared to re-constructed signal on different scales when compressing with *CDJ*, on dataset 1. The top plot shows 0.8 ms of data while the bottom plot shows the whole signal with 20 seconds of data.

.

CHAPTER

SEVEN

DISCUSSION

This section discusses implications of the results presented in the previous section. It also contextualizes the results in the frame of the report as a whole and underlines possible flaws and uncertainties. To end with it also discusses natural extensions and possibilities for future work.

7.1 Discussion of results

The most important findings from the results are that a combination of simple methods can be used to get around 40 - 50% compression without compromising information. And that the new method MS Quadro can outperform more complex methods like PCA.

7.2 Reliability of results

The datasets constitute 60 seconds of neurological recordings and include typical deviations from perfect recordings such as clipping of the data, and faulty channels. They also contain variations that come from different rat activity such as sleep and awake activity. This can explain the big difference between the compression results on the different datasets. In general, the compression methods perform worse on dataset 2 than on dataset 1, which might be because of the clipping that is present in the dataset. But the important takeaway is that there can be a lot of inter-dataset variability when working with this type of data. Which is why measures were taken to make sure that the final result did not include biases that were based on the attributes of that specific dataset. However, given more time and data one should also calculate the variance in the results and use this to give a quantitative measure of the reliability of the final result.

It is also worth underlining that the data in the datasets comes from a different system than the one that the compression algorithm will be implemented on. Even though measures were taken to mitigate the effects of this, as described in section 5.1, this will still have an effect.

7.3 Information loss

Since the data acquired from the system that will be designed, see figure 1.1.2, will be used for research it is very important to have as reliable data as possible. Furthermore, it is preferable to have data that has as little bias as possible. The advantage of using a lossless method is that it does not add any distortion to the signal, e.g. by adding spurious elements that can be misleading and increase uncertainty. Which methods like CS or machine learning variants can add.[1] This being said, all electronic systems have several inevitable sources of error and noise, including rounding error, quantization error in the ADC, inherent noise in the brain, transmission error, and more. And even though all of the methods used are theoretically reversible without loss they will still result in a small amount of information loss due to effects like rounding error. However, this error can often be negligible since it can be assumed to not add much extra uncertainty compared to the uncertainty that is already present in the system.

7.4 Usefulness

The end goal is to allow for a reduction in power consumption, which can in turn allow for the implementation of a wireless headstage. The two major factors that account for the power saved will be the amount of power used to compute the algorithm and the amount of power reduction as a result of reduced data transmission. One big advantage with *CDJ* is that it is a simple method with low hardware requirements, and that also needs very little computation. Although not directly transferable, Cuevas-López et al. show how their method can be implemented on an FPGA half the size of the ICE40, and that the power needed for computation of the algorithm is negligible compared to the power saved in transmission [41]. However, it is not possible to say how much power will be saved by using the method *CDJ* without implementing it in hardware and testing.

7.5 Future work

One useful next step is to implement the method *CDJ* on an FPGA to attain more reliable numbers on its performance, and power consumption. Part of this will also be to optimize its implementation to maximize power savings and to analyze what proportion of the total power consumption that can be removed using the method.

From the papers presented in chapter 3 and the tables in chapter B of the appendix two natural methods to investigate further are CS and machine learning, with their limitations kept in mind. Furthermore, a natural next step is to accommodate for the differences between the LFP, EAP, EcoG, and EMG signals, and also exploit the intended signal uses described in section 2.6. One possibility is to only send EAP signals on three of the tetrode channels and send the whole LFP+EAP

[1] Additionally, even the best CS methods can have problems when there are several overlapping AP spikes [36].

signal on the last tetrode channel. This is reasonable since it is the inter-tetrode LFP variation that is of interest as opposed to the intra-tetrode LFP variation. Furthermore, to accommodate for channels that might become faulty, as the one in figure 4.3.2, one can adaptively choose which channel sends the full signal based on a local faulty channel detector. The EAP signals can then be compressed using methods like CS, or run through local spike detection as in [35]. For the EAP signal, a lossy method is more justifiable since the performance of the method can be evaluated based on how well the AP signals are clustered, which would make methods that reduce noise but keep information advantageous. Furthermore, the EcoG and EMG channels can be filtered into their respective frequency ranges and compressed using methods that account for their specific attributes and use cases. Lastly, the full LFP+EAP channel can be compressed using a lossless method such as *CDJ*.

CHAPTER

EIGHT

CONCLUSIONS

This report has addressed the need for advances in brain monitoring systems, as neurological disorders pose significant health challenges globally. This is done through exploring signal compression as one possible way of reducing their power consumption, to enable the development of a wireless brain monitoring system. The study successfully demonstrates that the methods of FIR filtering, DPCM, MS Quadro, and Adaptive Rice coding can be combined to compress neural signals such as LFP, EAP, EcoG, and EMG, by 40-50% without compromising information integrity.

REFERENCES

[1] Trokel Hafting. "Project Proposal Wireless Neuroprobe-On-A-Chip". I dont know if I am refering to this correctly.

[2] *BrainChip - Institute of Basic Medical Sciences*. URL: https://www.med.uio.no/imb/english/people/aca/torkelh/brainchip/index.html (visited on 10/25/2023).

[3] *Wireless Neuroprobe-On-A-Chip - Prosjektbanken*. URL: https://prosjektbanken.forskningsradet.no/en/project/FORISS/332004?Kilde=FORISS&distribution=Ar&chart=bar&calcType=funding&Sprak=no&sortBy=date&sortOrder=desc&resultCount=30&offset=0&Prosjektleder=Richard+Bischof (visited on 11/02/2023).

[4] *THE 17 GOALS | Sustainable Development*. URL: https://sdgs.un.org/goals (visited on 12/13/2023).

[5] *Neurological Disorders: Public Health Challenges*. URL: https://www.who.int/publications-detail-redirect/9789241563369 (visited on 11/10/2023).

[6] *The Nobel Prize in Physiology or Medicine 2014*. NobelPrize.org. URL: https://www.nobelprize.org/prizes/medicine/2014/summary/ (visited on 11/10/2023).

[7] Roddy M. Grieves et al. "The place-cell representation of volumetric space in rats". In: *Nature Communications* 11.1 (Feb. 7, 2020). Number: 1 Publisher: Nature Publishing Group, p. 789. ISSN: 2041-1723. DOI: 10.1038/s41467-020-14611-7. URL: https://www.nature.com/articles/s41467-020-14611-7 (visited on 12/12/2023).

[8] *What Is a Neuron?* Nov. 22, 2016. URL: https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron (visited on 12/02/2023).

[9] *Background | National Institute on Drug Abuse (NIDA) - ClipArt Best - ClipArt Best*. URL: https://www.clipartbest.com/clipart-KijXKe5zT (visited on 12/13/2023).

[10] *Action Potential*. In: *Wikipedia*. Nov. 13, 2023. URL: https://en.wikipedia.org/w/index.php?title=Action_potential&oldid=1184911660 (visited on 12/02/2023).

[11] Marie Engelene J. Obien et al. "Revealing Neuronal Function through Microelectrode Array Recordings". In: *Frontiers in Neuroscience* 8 (2015). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2014.00423 (visited on 12/02/2023).

[12] Alain Destexhe and Claude Bedard. "Local Field Potential". In: *Scholarpedia* 8.8 (Aug. 26, 2013), p. 10713. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.10713. URL: http://www.scholarpedia.org/article/Local_field_potential (visited on 12/02/2023).

[13] *Brain Anatomy and How the Brain Works*. July 14, 2021. URL: https://www.hopkinsmedicine.org/health/conditions-and-diseases/anatomy-of-the-brain (visited on 12/02/2023).

[14] *Local Field Potential*. In: *Wikipedia*. Sept. 5, 2023. URL: https://en.wikipedia.org/w/index.php?title=Local_field_potential&oldid=1174028534 (visited on 09/18/2023).

[15] *Boccara Lab*. URL: https://boccaralab.com/ (visited on 10/18/2023).

[16] *Intan Technologies*. Intan Technologies. URL: https://www.intantech.com (visited on 11/09/2023).

[17] *iCE40 UltraPlus | AI/ML Low Power FPGA | Lattice Semiconductor*. URL: https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus#_1583858FEF1D4406B570F0CACD485268 (visited on 11/09/2023).

[18] *nRF52811 - Bluetooth 5.4 SoC*. URL: https://www.nordicsemi.com/Products/nRF52811 (visited on 11/09/2023).

[19] Rafael Bretas et al. "Neural Representation of Overlapping Path Segments and Reward Acquisitions in the Monkey Hippocampus". In: *Frontiers in Systems Neuroscience* 13 (Sept. 1, 2019), p. 48. DOI: 10.3389/fnsys.2019.00048.

[20] *Nyquist–Shannon Sampling Theorem*. In: *Wikipedia*. Oct. 14, 2023. URL: https://en.wikipedia.org/w/index.php?title=Nyquist%E2%80%93Shannon_sampling_theorem&oldid=1180085259 (visited on 11/07/2023).

[21] *Entropy Coding*. In: *Wikipedia*. Aug. 11, 2023. URL: https://en.wikipedia.org/w/index.php?title=Entropy_coding&oldid=1169842099 (visited on 09/18/2023).

[22] Laxmi Shaw, Daleef Rahman, and Aurobinda Routray. "Highly Efficient Compression Algorithms for Multichannel EEG". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.5 (May 2018), pp. 957–968. ISSN: 1558-0210. DOI: 10.1109/TNSRE.2018.2826559.

[23] *Huffman Coding*. In: *Wikipedia*. Sept. 18, 2023. URL: https://en.wikipedia.org/w/index.php?title=Huffman_coding&oldid=1175907130 (visited on 11/10/2023).

[24] *Golomb Coding*. In: *Wikipedia*. Sept. 20, 2023. URL: https://en.wikipedia.org/w/index.php?title=Golomb_coding&oldid=1176279887 (visited on 12/05/2023).

[25] *Rice (Golomb) Coding Encoding Discussion and Implementation*. URL: https://michaeldipperstein.github.io/rice.html (visited on 12/05/2023).

[26] Sebastian Schmale et al. "Joint Compression of Neural Action Potentials and Local Field Potentials". In: *2013 Asilomar Conference on Signals, Systems and Computers*. 2013 Asilomar Conference on Signals, Systems and Computers. Nov. 2013, pp. 1823–1827. DOI: 10.1109/ACSSC.2013.6810617.

[27] *3.8.2: Discrete Cosine Transformation*. Engineering LibreTexts. May 2, 2021. URL: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Signal_Processing_and_Modeling/Information_and_Entropy_(Penfield)/03%3A_Compression/3.08%3A_Detail-_2-D_Discrete_Cosine_Transformation/3.8.02%3A_Discrete_Cosine_Transformation (visited on 11/29/2023).

[28] Yuli You. "Joint Channel Coding". In: *Audio Coding: Theory and Applications*. Ed. by Yuli You. Boston, MA: Springer US, 2010, pp. 231–234. ISBN: 978-1-4419-1754-6. DOI: 10.1007/978-1-4419-1754-6_12. URL: https://doi.org/10.1007/978-1-4419-1754-6_12 (visited on 09/18/2023).

[29] N. Ahmed, T. Natarajan, and K.R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* C-23.1 (Jan. 1974), pp. 90–93. ISSN: 1557-9956. DOI: 10.1109/T-C.1974.223784. URL: https://ieeexplore.ieee.org/document/1672377 (visited on 11/10/2023).

[30] Sung-Yun Park, Jihyun Cho, and Euisik Yoon. "3.37 μW/Ch Modular Scalable Neural Recording System with Embedded Lossless Compression for Dynamic Power Reduction". In: *2017 Symposium on VLSI Circuits*. 2017 Symposium on VLSI Circuits. June 2017, pp. C168–C169. DOI: 10.23919/VLSIC.2017.8008468. URL: https://ieeexplore.ieee.org/abstract/document/8008468 (visited on 11/07/2023).

[31] Oscar W. Savolainen and Timothy G. Constandinou. "Lossless Compression of Intracortical Extracellular Neural Recordings Using Non-Adaptive Huffman Encoding". In: *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). July 2020, pp. 4318–4321. DOI: 10.1109/EMBC44109.2020.9176352.

[32] Lei Lin et al. "Multichannel EEG Compression Based on ICA and SPIHT". In: *Biomedical Signal Processing and Control* 20 (July 1, 2015), pp. 45–51. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2015.04.001. URL: https://www.sciencedirect.com/science/article/pii/S174680941500049X (visited on 12/03/2023).

[33] Guillermo Dufort y Álvarez et al. "Wireless EEG System Achieving High Throughput and Reduced Energy Consumption Through Lossless and Near-Lossless Compression". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (Feb. 2018), pp. 231–241. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2017.2779324.

[34] Y. Wongsawat et al. "Lossless Multi-Channel EEG Compression". In: *2006 IEEE International Symposium on Circuits and Systems*. 2006 IEEE International Symposium on Circuits and Systems. May 2006, 4 pp.–1614. DOI: 10.1109/ISCAS.2006.1692909.

[35] Guillaume Bilodeau et al. "A Wireless Electro-Optic Platform for Multimodal Electrophysiology and Optogenetics in Freely Moving Rodents". In: *Frontiers in Neuroscience* 15 (2021). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2021.718478 (visited on 09/18/2023).

[36] Biao Sun and Wenfeng Zhao. "Compressed Sensing of Extracellular Neurophysiology Signals: A Review". In: *Frontiers in Neuroscience* 15 (2021). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2021.682063 (visited on 09/16/2023).

[37] Biao Sun et al. "Training-Free Deep Generative Networks for Compressed Sensing of Neural Action Potentials". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.10 (Oct. 2022), pp. 5190–5199. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3069436.

[38] Wenfeng Zhao et al. "Block-Sparse Modeling for Compressed Sensing of Neural Action Potentials and Local Field Potentials". In: *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. 2019 53rd Asilomar Conference on Signals, Systems, and Computers. Nov. 2019, pp. 2097–2100. DOI: 10.1109/IEEECONF44664.2019.9048841.

[39] Biao Sun et al. "Compressed Sensing of Large-Scale Local Field Potentials Using Adaptive Sparsity Analysis and Non-Convex Optimization". In: *Journal of Neural Engineering* 18.2 (Feb. 2021), p. 026007. ISSN: 1741-2552. DOI: 10.1088/1741-2552/abd578. URL: https://dx.doi.org/10.1088/1741-2552/abd578 (visited on 09/18/2023).

[40] Yousef Khazaei, Ali Abbasi Shahkooh, and Amir M. Sodagar. "Spatial Redundancy Reduction in Multi-Channel Implantable Neural Recording Microsystems". In: *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). July 2020, pp. 898–901. DOI: 10.1109/EMBC44109.2020.9175732.

[41] Aarón Cuevas-López et al. "Low-Power Lossless Data Compression for Wireless Brain Electrophysiology". In: *Sensors* 22.10 (10 Jan. 2022), p. 3676. ISSN: 1424-8220. DOI: 10.3390/s22103676. URL: https://www.mdpi.com/1424-8220/22/10/3676 (visited on 11/07/2023).

[42] *Huffman Coding Algorithm*. URL: https://www.programiz.com/dsa/huffman-coding (visited on 11/10/2023).

[43] Arda Uran et al. "A 16-Channel Neural Recording System-on-Chip With CHT Feature Extraction Processor in 65-Nm CMOS". In: *IEEE Journal of Solid-State Circuits* 57.9 (Sept. 2022), pp. 2752–2763. ISSN: 1558-173X. DOI: 10.1109/JSSC.2022.3161296.

[44] G. Bilodeau et al. "A Wireless Electro-Optic Headstage with Digital Signal Processing and Data Compression for Multimodal Electrophysiology and Optogenetic Stimulation". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). Oct. 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180912.

[45] Tao Xiong et al. "An Unsupervised Compressed Sensing Algorithm for Multi-Channel Neural Recording and Spike Sorting". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.6 (June 2018), pp. 1121–1130. ISSN: 1558-0210. DOI: 10.1109/TNSRE.2018.2830354. URL: https://ieeexplore.ieee.org/abstract/document/8350329 (visited on 11/07/2023).

[46] Takayuki Okazawa and Ippei Akita. "A Time-Domain Analog Spatial Compressed Sensing Encoder for Multi-Channel Neural Recording". In: *Sensors* 18.1 (1 Jan. 2018), p. 184. ISSN: 1424-8220. DOI: 10.3390/s18010184. URL: https://www.mdpi.com/1424-8220/18/1/184 (visited on 11/07/2023).

[47] Tong Wu et al. "Deep Compressive Autoencoder for Action Potential Compression in Large-Scale Neural Recording". In: *Journal of Neural Engineering* 15.6 (Oct. 2018), p. 066019. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aae18d. URL: https://dx.doi.org/10.1088/1741-2552/aae18d (visited on 09/23/2023).

[48] Tong Wu et al. "A Lightweight Deep Compressive Model for Large-Scale Spike Compression". In: *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS). Oct. 2018, pp. 1–4. DOI: 10.1109/BIOCAS.2018.8584752. URL: https://ieeexplore.ieee.org/abstract/document/8584752 (visited on 11/07/2023).

[49] Tong Wu et al. "A Streaming PCA VLSI Chip for Neural Data Compression". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.6 (Dec. 2017), pp. 1290–1302. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2017.2717281. URL: https://ieeexplore.ieee.org/abstract/document/8010351 (visited on 11/07/2023).

[50] Gabriel Gagnon-Turcotte et al. "A Wireless Headstage for Combined Optogenetics and Multichannel Electrophysiological Recording". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.1 (Feb. 2017), pp. 1–14. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2016.2547864. URL: https://ieeexplore.ieee.org/abstract/document/7494925 (visited on 11/07/2023).

[51] Tong Wu et al. "A Streaming PCA Based VLSI Chip for Neural Data Compression". In: *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS). Oct. 2016, pp. 192–195. DOI: 10.1109/BioCAS.2016.7833764. URL: https://ieeexplore.ieee.org/abstract/document/7833764 (visited on 11/07/2023).

[52] G. Gagnon-Turcotte et al. "A Wireless Optogenetic Headstage with Multichannel Neural Signal Compression". In: *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2015 IEEE Biomedical Circuits and Systems Conference (BioCAS). Oct. 2015, pp. 1–4. DOI: 10.1109/BioCAS.2015.7348342. URL: https://ieeexplore.ieee.org/abstract/document/7348342 (visited on 11/07/2023).

[53] Russell Dodd, Bruce F. Cockburn, and Vincent Gaudet. "Neural Spike Compression Using Feature Extraction and a Fuzzy C-Means Codebook". In: *2014 IEEE 44th International Symposium on Multiple-Valued Logic*. 2014 IEEE 44th International Symposium on Multiple-Valued Logic. May 2014, pp. 44–48. DOI: 10.1109/ISMVL.2014.16. URL: https://ieeexplore.ieee.org/abstract/document/6844994 (visited on 11/07/2023).

[54] Jie Zhang et al. "An Efficient and Compact Compressed Sensing Microsystem for Implantable Neural Recordings". In: *IEEE Transactions on Biomedical Circuits and Systems* 8.4 (Aug. 2014), pp. 485–496. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2013.2284254. URL: https://ieeexplore.ieee.org/abstract/document/6693746 (visited on 11/07/2023).

[55] Lei Liu et al. "Neural Recording Front-End IC Using Action Potential Detection and Analog Buffer with Digital Delay for Data Compression". In: *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). July 2013, pp. 747–750. DOI: 10.1109/EMBC.2013.6609608. URL: https://ieeexplore.ieee.org/abstract/document/6609608 (visited on 11/07/2023).

[56] Jie Zhang et al. "Reconstruction of Neural Action Potentials Using Signal Dependent Sparse Representations". In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2013 IEEE International Symposium on Circuits and Systems (ISCAS). May 2013, pp. 1520–1523. DOI: 10.1109/ISCAS.2013.6572147. URL: https://ieeexplore.ieee.org/abstract/document/6572147 (visited on 11/07/2023).

[57] Shuai Zhou et al. "Compressive Sensing of Neural Action Potentials Using Nano Platinum Black Modified Microelectrode Array". In: *The 8th Annual IEEE International Conference on Nano/Micro Engineered and Molecular Systems*. The 8th Annual IEEE International Conference on Nano/Micro Engineered and Molecular Systems. Apr. 2013, pp. 432–435. DOI: 10.1109/NEMS.2013.6559765. URL: https://ieeexplore.ieee.org/abstract/document/6559765 (visited on 11/07/2023).

[58] Christoph Bulach, Ulrich Bihr, and Maurits Ortmanns. "Evaluation Study of Compressed Sensing for Neural Spike Recordings". In: *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Aug. 2012, pp. 3507–3510. DOI: 10.1109/EMBC.2012.6346722. URL: https://ieeexplore.ieee.org/abstract/document/6346722 (visited on 11/07/2023).

[59] Stefan Craciun et al. "Wireless Transmission of Neural Signals Using Entropy and Mutual Information Compression". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 19.1 (Feb. 2011), pp. 35–44. ISSN: 1558-0210. DOI: 10.1109/TNSRE.2010.2070078. URL: https://ieeexplore.ieee.org/abstract/document/5560866 (visited on 11/07/2023).

[60] A. Bonfanti et al. "A Wireless Microsystem with Digital Data Compression for Neural Spike Recording". In: *Microelectronic Engineering*. Proceedings of the 36th International Conference on Micro- and Nano-Engineering (MNE) 88.8 (Aug. 1, 2011), pp. 1672–1675. ISSN: 0167-9317. DOI: 10.1016/j.mee.2011.01.024. URL: https://www.sciencedirect.com/science/article/pii/S0167931711000359 (visited on 11/07/2023).

[61] Stefan Craciun et al. “Compression of Neural Signals Using Discriminative Coding for Wireless Applications”. In: *2009 4th International IEEE/EMBS Conference on Neural Engineering*. 2009 4th International IEEE/EMBS Conference on Neural Engineering. Apr. 2009, pp. 629–632. DOI: 10.1109/NER.2009.5109375. URL: https://ieeexplore.ieee.org/abstract/document/5109375 (visited on 11/07/2023).

[62] Seetharam Narasimhan et al. “Neural Data Compression with Wavelet Transform: A Vocabulary Based Approach”. In: *2007 3rd International IEEE/EMBS Conference on Neural Engineering*. 2007 3rd International IEEE/EMBS Conference on Neural Engineering. May 2007, pp. 666–669. DOI: 10.1109/CNE.2007.369760. URL: https://ieeexplore.ieee.org/abstract/document/4227365 (visited on 11/07/2023).

[63] R.H. Olsson and K.D. Wise. “A Three-Dimensional Neural Recording Microsystem with Implantable Data Compression Circuitry”. In: *IEEE Journal of Solid-State Circuits* 40.12 (Dec. 2005), pp. 2796–2804. ISSN: 1558-173X. DOI: 10.1109/JSSC.2005.858479. URL: https://ieeexplore.ieee.org/abstract/document/1546254 (visited on 11/07/2023).

APPENDICES

APPENDIX

A

HUFFMAN CODING

The first step in Huffman coding is to predict the frequency of occurrence of each value that a signal can take. These probabilities of occurrence can be estimated using previously recorded signal data. The number of occurrences for each signal value is first counted, then the Huffman tree is built based on this. Using the Huffman tree one can then determine what value gets what binary code. An example of encoding characters using Huffman is shown in figure A.0.1, A.0.2 and A.0.3. Here we see that the most frequently occurring symbol gets the shortest code.

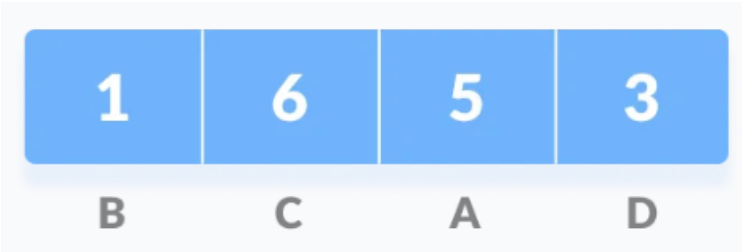


Figure A.0.1: Frequency of occurrence of characters, taken from [42].

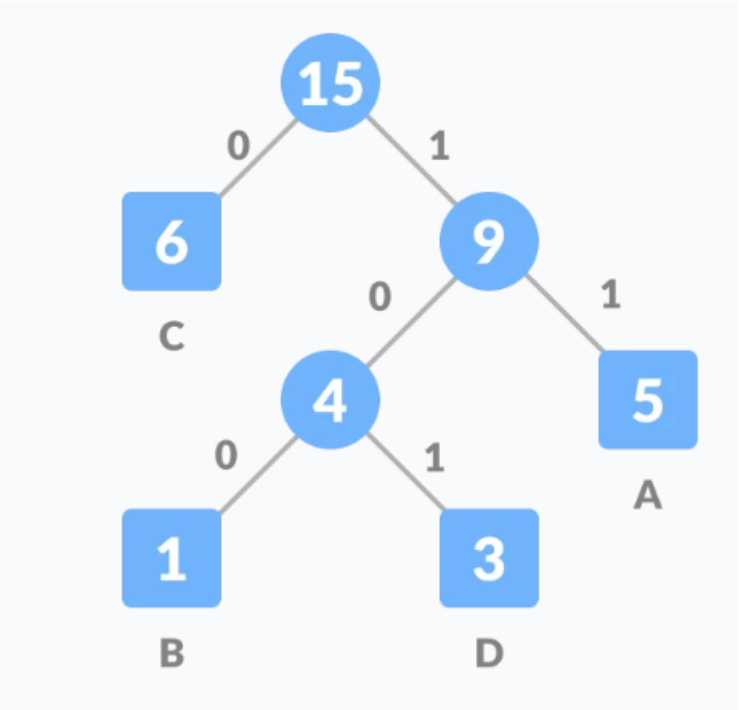


Figure A.0.2: Huffman tree built from A.0.1, taken from [42].

Character	Frequency	Code
A	5	11
B	1	100
C	6	0
D	3	101

Figure A.0.3: Table of resulting codewords using A.0.2, taken from [42].

APPENDIX

B

PREVIOUS WORK - TABLES

Tables B.0.1 to B.0.7 give an overview of methods that have been used to compress LFP and EAP signals. The compression results from each paper are deliberately not included because different papers have used different datasets and have taken validation measures to different extents.

References	Method	Comments	Full title
A. Cuevas-Lopez et al. - 2022 [41]	Remove 3bits + DPCM + Huffman	Uses INTAN RHD2132, and FPGA with 2.5k LUTs	Low-Power Lossless Data Compression for Wireless Brain Electrophysiology
Y. Khazaei et al. - 2020 [40]	Remove inter-channel avg. + adaptive bit allocation	Only Spatial (not temporal)	Spatial Redundancy Reduction in Multi-Channel Implantable Neural Recording Microsystems
O. Savolainen & T. Constandinou - 2020 [31]	Linear Neural Network (LNN) + Huffman		Lossless Compression of Intracortical Extracellular Neural Recordings using Non-Adaptive Huffman Encoding
SY. Park et al. - 2017 [30]	Utilizing spatiotemporal correlation and sparsity of neural signals	"reduced P-D consumption by 89% while achieving the state-of-the-art recording performance of 3.37 μW/Ch, 5.18 μV-rms input-referred noise"	3.37 mu W/Ch Modular Scalable Neural Recording System with Embedded Lossless Compression for Dynamic Power Reduction

Table B.0.1: Published work on compression of LFP and EAP signals that claim to be lossless.

References	Method	Comments	Full title
A. Uran et al. - 2022 [43]	Feature extraction with compressed Hadamard transform (CHT), selection of the features to be computed is tailored through a machine learning algorithm		A 16-Channel Neural Recording System-on-Chip With CHT Feature Extraction Processor in 65-nm CMOS
G. Bilodeau et al. - 2021 [35]	LFP & EAP separation, AP detection, and DWT compression	Uses INTAN RHD2132, and FPGA with 15k LUTs	A Wireless Electro-Optic Platform for Multimodal Electrophysiology and Optogenetics in Freely Moving Rodents
B. sun et al. - 2021 [37]	Deep neural network (DNN) for CS reconstruction of EAP signals	DNN does not need to be trained	Training-Free Deep Generative Networks for Compressed Sensing of Neural Action Potentials
B. sun & W. Zhao - 2021 [36]	CS Review for LFP and EAP signals		Compressed Sensing of Extracellular Neurophysiology Signals: A Review
B. Sun et al. - 2021 [39]	CS on LFP with non-convex optimizer		Compressed sensing of large-scale local field potentials using adaptive sparsity analysis and non-convex optimization

Table B.0.2: Published work on compression of LFP and EAP signals. Table 1 of 6.

References	Method	Comments	Full title
G. Bilodeau et al. - 2020 [44]	LFP & EAP separation, EAP detection, and DWT compression		A wireless electro-optic headstage with digital signal processing and data compression for multimodal electrophysiology and optogenetic stimulation
W. Zhao et al. - 2019 [38]	CS on EAP & LFP using DCT and binary-weighted l1-minimization	Exploits the fact that the frequency bands of the LFP and EAP signals are known when re-constructing	Block-Sparse Modeling for Compressed Sensing of Neural Action Potentials and Local Field Potentials
T. Xiong et al. - 2018 [45]	Unsupervised CS, with group sparsity and template matching		An Unsupervised Compressed Sensing Algorithm for Multi-Channel Neural Recording and Spike Sorting
T. Okazawa & I. Akita - 2018 [46]	Analog CS for EAP		A Time-Domain Analog Spatial Compressed Sensing Encoder for Multi-Channel Neural Recording
T. Wu et al. - 2018 [47]	Deep compressive autoencoder (CAE) with embedding space updated via Vector Quantization (VQ) + entropy coding	"15×better than PCA and over 70×better than DWT, DCT", for 500x CR. Most significant improvements at high CR.	Deep compressive autoencoder for action potential compression in large-scale neural recording

Table B.0.3: Published work on compression of LFP and EAP signals. Table 2 of 6.

References	Method	Comments	Full title
T. Wu et al. - 2018 [48]	Same as paper below	Same as paper below	A Lightweight Deep Compressive Model for Large-Scale Spike Compression
T. Wu et al. - 2017 [49]	LFP and EAP split, PCA feature extraction,	Implemented on integrated circuit (IC), 144-nW/channel power consumption (LFP compression), 3.05-μW/channel power consumption (EAP compression)	A Streaming PCA VLSI Chip for Neural Data Compression
G. Gagnon-Turcotte et al. - 2017 [50]	AP detection, DWT compression with dynamic coefficient re-quantization		A Wireless Headstage for Combined Optogenetics and Multichannel Electrophysiological Recording
T. Wu et al. - 2016 [51]	PCA for LFP compression	144nW per channel at a 0.5V power supply	A Streaming PCA based VLSI Chip for Neural Data Compression
G. Gagnon-Turcotte et al. - 2015 [52]	AP detection + DWT		A wireless optogenetic headstage with multichannel neural signal compression

Table B.0.4: Published work on compression of LFP and EAP signals. Table 3 of 6.

References	Method	Comments	Full title
R. Dodd et al. - 2014 [53]	AP feature extraction using fuzzy c-means codebook	"improves neural signal compression by 87% compared to spike detection alone"	Neural Spike Compression Using Feature Extraction and a Fuzzy C-Means Codebook
J. Zhang et al. – 2014 [54]	Signal-dependent CS on EAP	"0.27 μW operating at 20 KHz" for the compression implementation.	An Efficient and Compact Compressed Sensing Microsystem for Implantable Neural Recordings
S. Schmale et al. - 2013 [26]	Review paper	Concludes that DCT is the best dictionary for CS	Joint Compression of Neural Action Potentials and Local Field Potentials
L. Liu et al. - 2013 [55]	Neural recording front end, detects AP and low-pass filters		Neural Recording Front-End IC Using Action Potential Detection and Analog Buffer with Digital Delay for Data Compression
J. Zhang et al. - 2013 [56]	CS using K-SVD & DWT	Argues for signal-dependent CS, "6 dB better SNDR"	Reconstruction of Neural Action Potentials Using Signal Dependent Sparse Representations

Table B.0.5: Published work on compression of LFP and EAP signals. Table 4 of 6.

References	Method	Comments	Full title
S. Zhou et al. - 2013 [57]	EAP detection + CS (DWT basis & Bayesian CS algorithm)		Compressive Sensing of Neural Action Potentials Using Nano Platinum Black Modified Microelectrode Array
C. Bulach et al. - 2012 [58]	Evaluation study CS for EAP	"CS is shown to work for the compression of low-noise synthesized neural spike signals with a compression rate of 2.05, but cannot be recommended for the compression of neural spike signals in general."	Evaluation study of compressed sensing for neural spike recordings
S. Craciun et al. - 2011 [59]	Vector quantization	EAP compression without thresholding	Wireless Transmission of Neural Signals Using Entropy and Mutual Information Compression
A. Bonfanti et al. - 2011 [60]	EAP detection and storing up to 20 points per waveform		A wireless microsystem with digital data compression for neural spike recording
S. Craciun et al. - 2009 [61]	Vector quantization (discriminative Linde-Buzo-Gray algorithm (DLBG)) on EAP		Compression of Neural Signals using Discriminative Coding for Wireless Applications

Table B.0.6: Published work on compression of LFP and EAP signals. Table 5 of 6.

References	Method	Comments	Full title
S. Narasimhan et al - 2007 [62]	Vocabulary-based Wavelet Transform		Neural data compression with wavelet transform: A vocabulary based approach
RH. Olsson & KD. Wise – 2005 [63]	Spike detection + key feature sent	Implemented on ASIC	A three-dimensional neural recording microsystem with implantable data compression circuitry

Table B.0.7: Published work on compression of LFP and EAP signals. Table 6 of 6.

APPENDIX

C

GITHUB REPOSITORY

All of the code used for this project is included in the Github repository linked below. The CombineMethods.ipynb file was used for all of the testing.

Github repository link

- `https://github.com/ariaalinejad/NauralSignalCompression`

APPENDIX

D

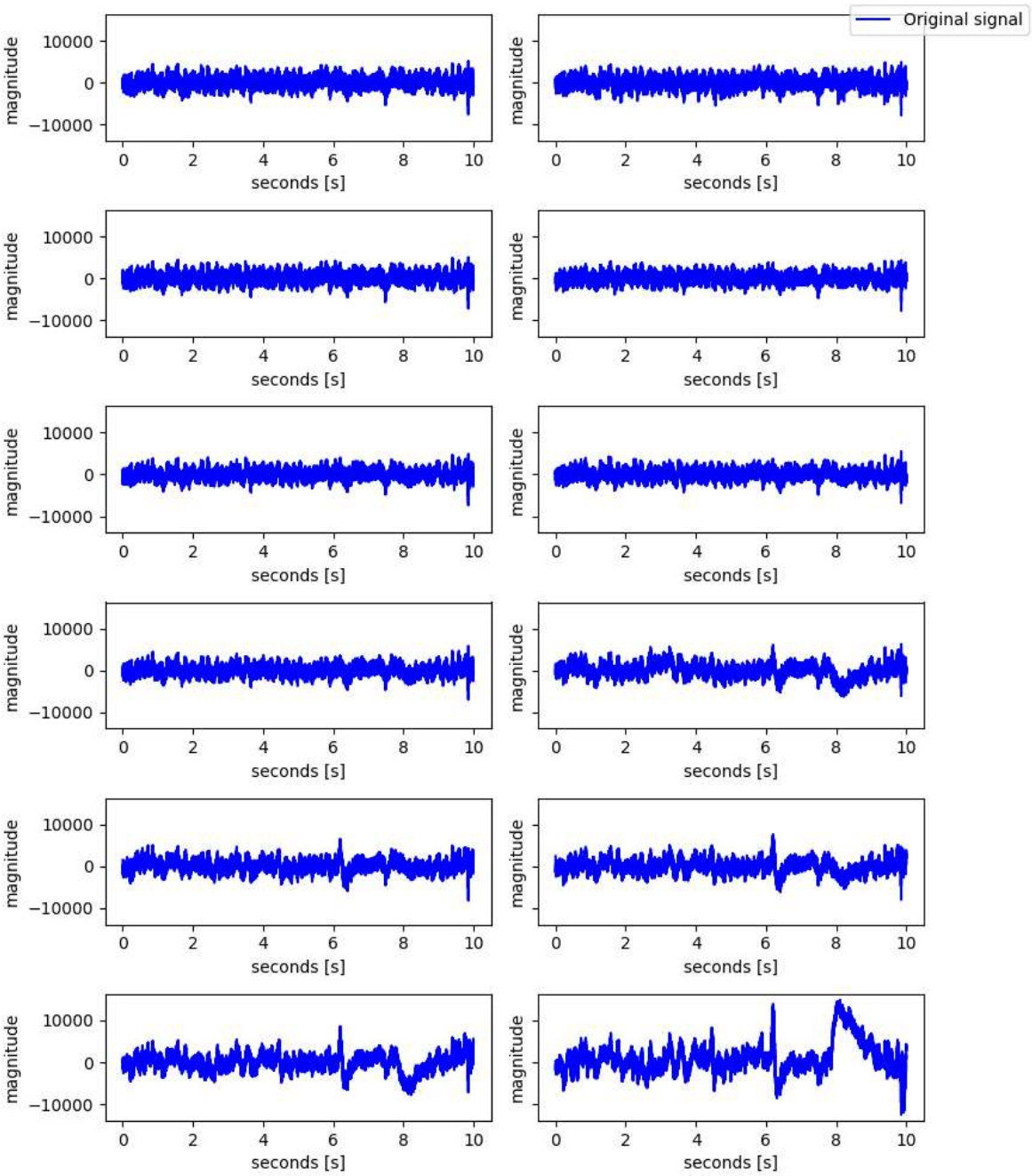
ADDITIONAL SIGNAL ANALYSIS

Here I will emphesize the different frequency components in the signal and how the LFP and AP signal differ

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, freqz, lfilter, filtfilt
import warnings
warnings.simplefilter("ignore")

s = np.load('../test_data_sintef.npy') # 48kHz (Dataset 1)
#s = np.load('../52728_2021-09-25_01_24KHz_test_data_sintef.npy') # 24kHz (Datas
#s = np.load('../56180_2021-11-05_06_24KHz_test_data_sintef.npy') # 24kHz (Datas
#s = np.load('../61467_2022-09-16_01_48KHz_test_data_sintef.npy') # 48kHz (Datas
```

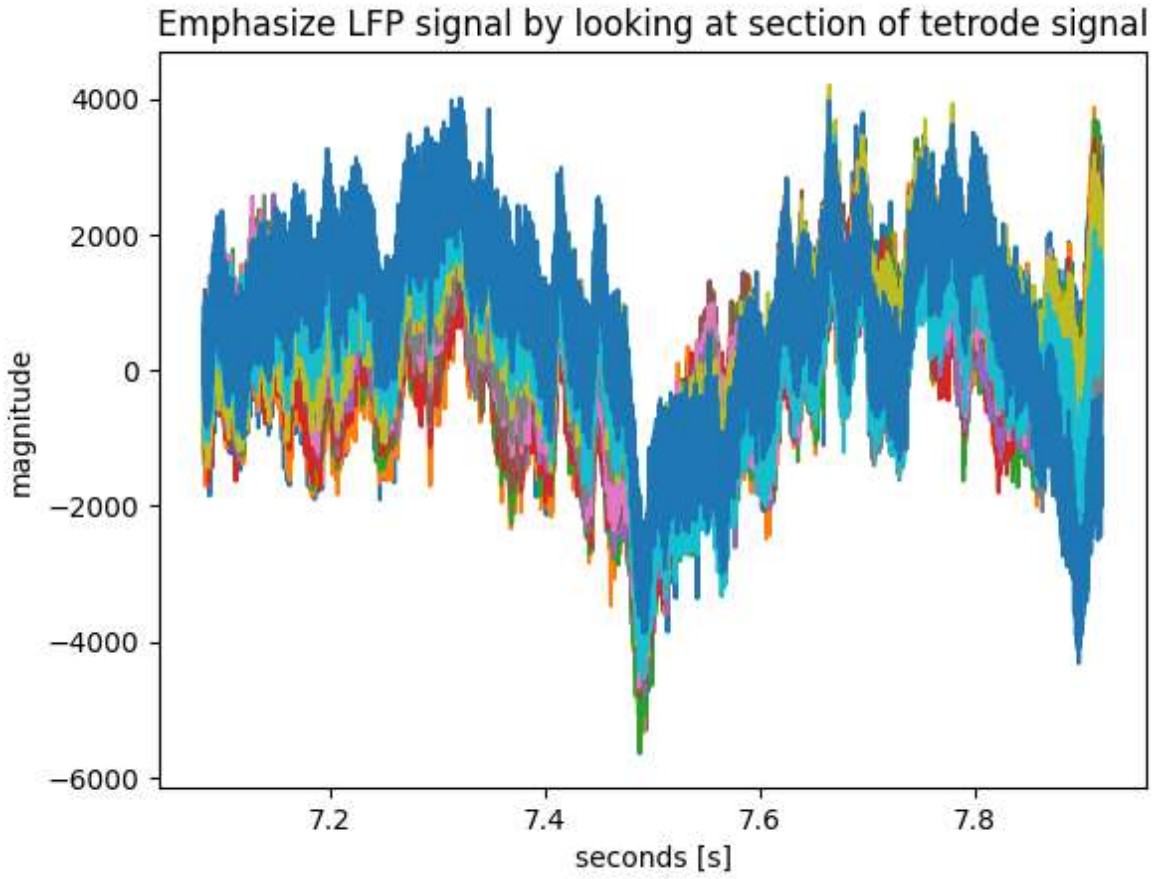
Visualize

In [ ]:
```
# Visualize the signals
ts = np.linspace(0,10, int(s.shape[0])) # time

# Visualize the whole signals with 3 bits less
fig, axs = plt.subplots(6, 2, figsize=(8, 10), sharey=True)
for i in range(s.shape[1] - 4):
    axs.flatten()[i].plot(ts, s[:,i], 'b')
    axs.flatten()[i].set_xlabel('seconds [s]')
    axs.flatten()[i].set_ylabel('magnitude')
fig.legend(['Original signal'], loc='upper right', bbox_to_anchor=(1.1, 1))
plt.tight_layout()
plt.show()
```



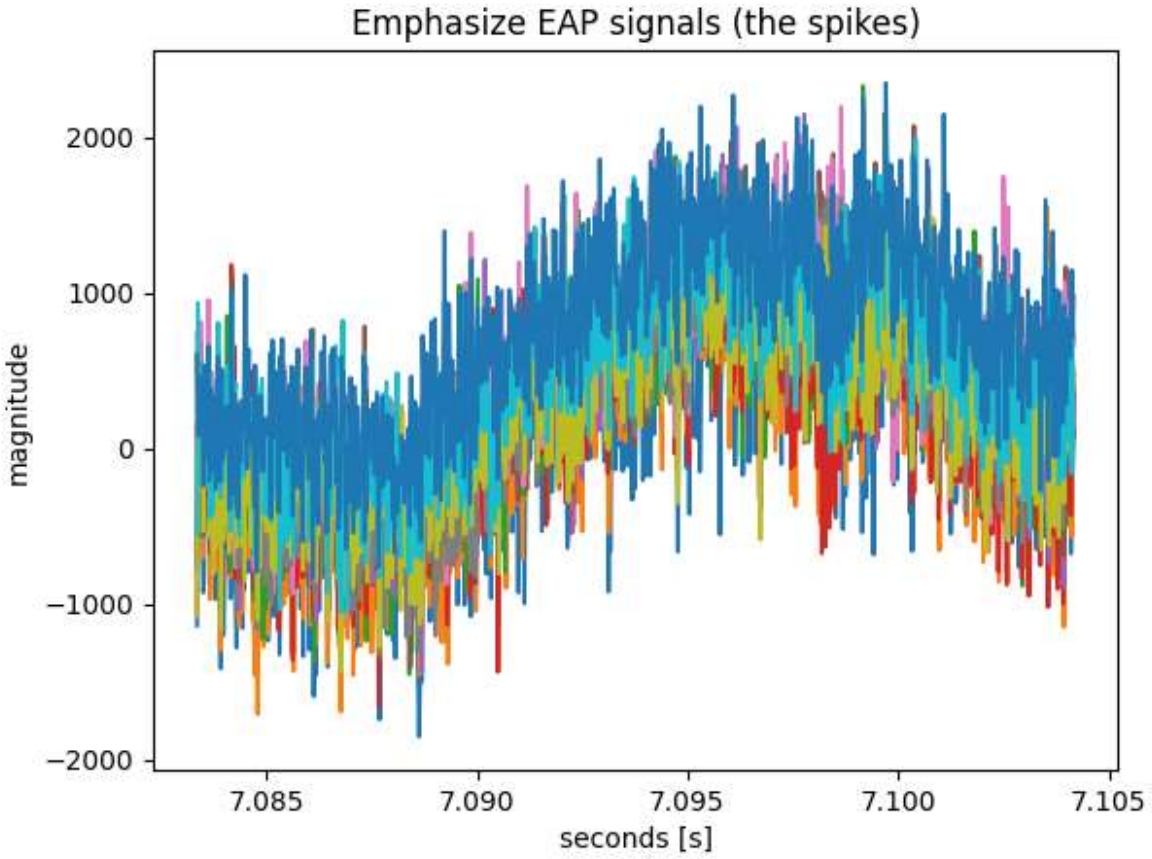
In [ ]:
```
# Emphasize the LFP signals (the oscillation seen in several channels)
plt.plot(ts[340000:380000], s[340000:380000,:11])
plt.xlabel('seconds [s]')
plt.ylabel('magnitude')
plt.title('Emphasize LFP signal by looking at section of tetrode signal')
```

Out[ ]:
```
Text(0.5, 1.0, 'Emphasize LFP signal by looking at section of tetrode signal')
```



In [ ]:
```
# Emphasize the EAPsignals (the spikes)
plt.plot(ts[340000:341000], s[340000:341000,:11])
plt.xlabel('seconds [s]')
plt.ylabel('magnitude')
plt.title('Emphasize EAP signals (the spikes)')
```

Out[ ]:
```
Text(0.5, 1.0, 'Emphasize EAP signals (the spikes)')
```



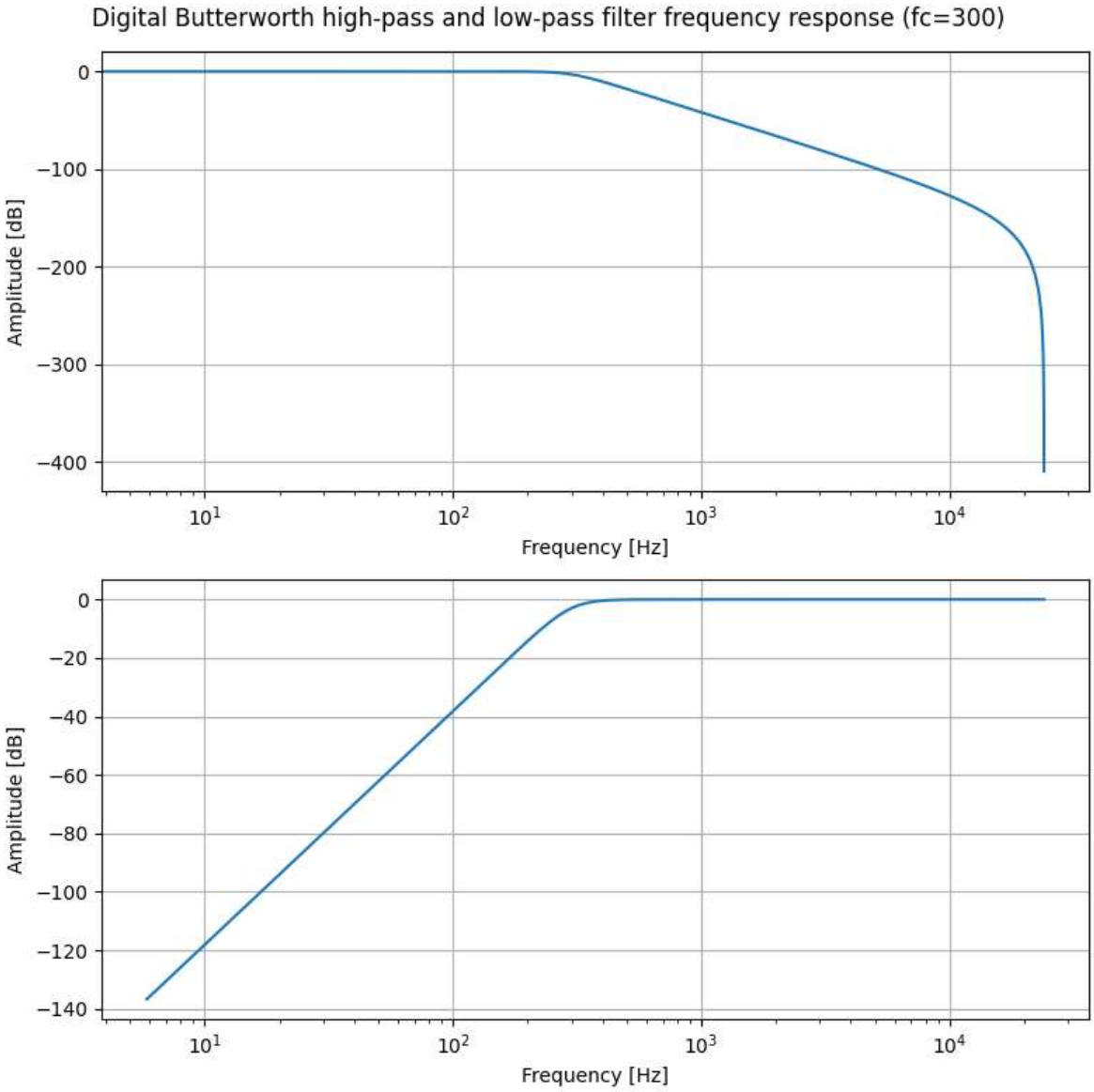
Filter

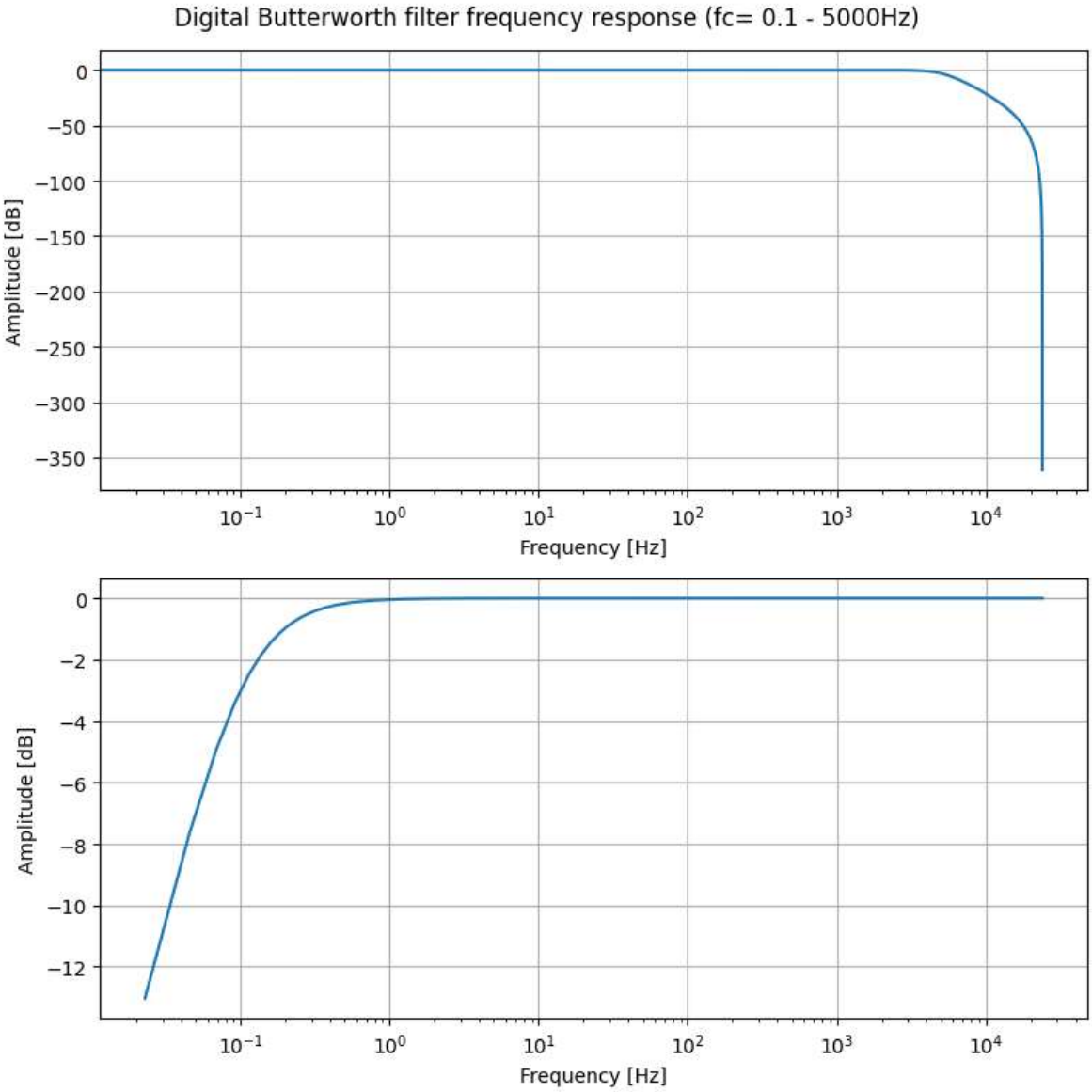
Visualize the Digital Butterworth lowpass and highpass filters used

Two of the digital filters are used to replicate the internal analog filters in the INTAN chip

In [ ]:
```
# Digital Low pass and high pass filters
fs = 48000
fc = 300
Wn = fc / (fs/2)
b, a = butter(N=4, Wn=Wn, btype='low', analog=False, output='ba')
b2, a2 = butter(N=4, Wn=Wn, btype='high', analog=False, output='ba')

# compute the frequency response
w, h = freqz(b, a, 2**12)
f = w/(2*np.pi) *fs
w2, h2 = freqz(b2, a2, 2**12)
f2 = w2/(2*np.pi) *fs

# plot the magnitude response
fig, axs = plt.subplots(2,1,figsize=(8,8))
axs[0].semilogx(f, 20 * np.log10(abs(h)))
axs[1].semilogx(f2, 20 * np.log10(abs(h2)))
axs[0].set_xlabel('Frequency [Hz]')
axs[0].set_ylabel('Amplitude [dB]')
axs[1].set_xlabel('Frequency [Hz]')
axs[1].set_ylabel('Amplitude [dB]')
axs[0].grid(True)
axs[1].grid(True)
```

```
fig.suptitle('Digital Butterworth high-pass and low-pass filter frequency respon
plt.tight_layout()
plt.show()


#----


# Digital Low pass and high pass filters
fs = 48000
fc1 = 5000
fc2 = 0.1
Wn1 = fc1 / (fs/2)
Wn2 = fc2 / (fs/2)
b3, a3 = butter(N=3, Wn=Wn1, btype='low', analog=False, output='ba')
b4, a4 = butter(N=1, Wn=Wn2, btype='high', analog=False, output='ba')

# compute the frequency response
w3, h3 = freqz(b3, a3, 2**20)
f3 = w3/(2*np.pi) *fs
w4, h4 = freqz(b4, a4, 2**20)
f4 = w4/(2*np.pi) *fs

# plot the magnitude response
fig, axs = plt.subplots(2,1, figsize=(8,8))
axs[0].semilogx(f3, 20 * np.log10(abs(h3)))
axs[1].semilogx(f4, 20 * np.log10(abs(h4)))
axs[0].set_xlabel('Frequency [Hz]')
axs[0].set_ylabel('Amplitude [dB]')
axs[1].set_xlabel('Frequency [Hz]')
axs[1].set_ylabel('Amplitude [dB]')
axs[0].grid(True)
axs[1].grid(True)
fig.suptitle('Digital Butterworth filter frequency response (fc= 0.1 - 5000Hz)')
plt.tight_layout()
plt.show()
```


Digital Butterworth high-pass and low-pass filter frequency response (fc=300)
0
−100
−200
−300
−400
Amplitude [dB]
10^1
10^2
10^3
10^4
Frequency [Hz]
0
−20
−40
−60
−80
−100
−120
−140
Amplitude [dB]
10^1
10^2
10^3
10^4
Frequency [Hz]



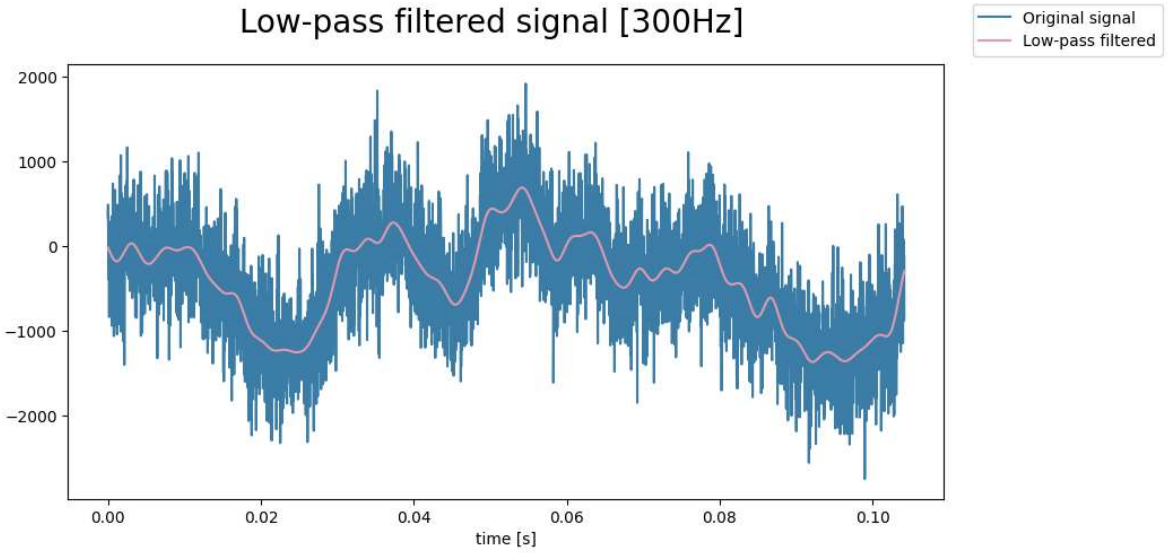
Filter the signal

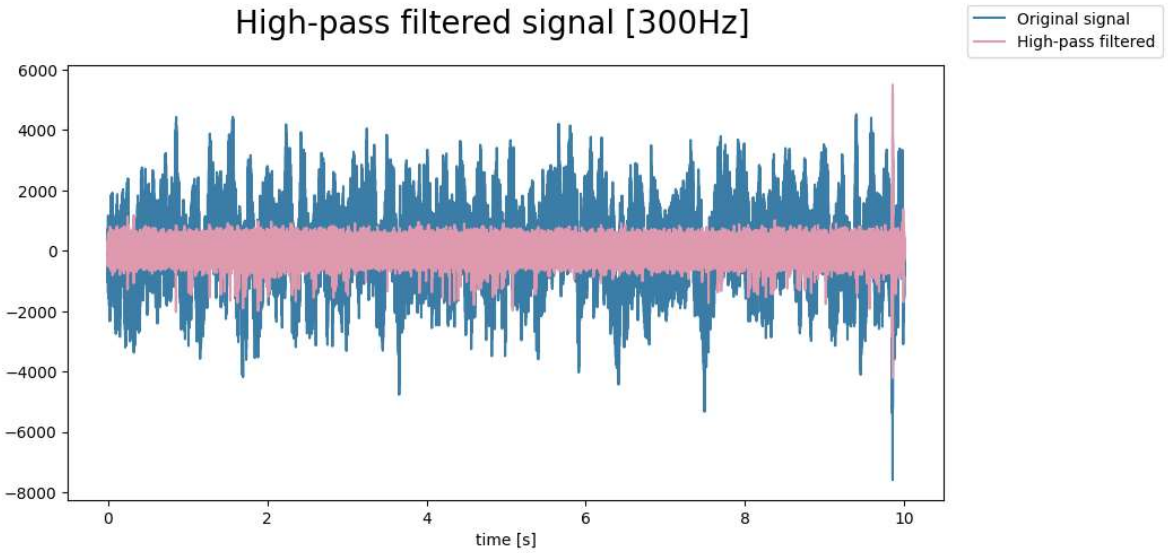
In [ ]:

```
x = s[:,:]
j = 0 # channel we plot
time = 10*x.shape[0]/s.shape[0]
tx = np.linspace(0,time, int(x.shape[0])) # time


#first band pass filter the signal (using what resembles the analog filters in t
lp = np.zeros_like(x)
bp = np.zeros_like(x)
AP= np.zeros_like(x)
LFP = np.zeros_like(x)
for i in range(x.shape[1]):
    lp[:,i] = lfilter(b3,a3, x[:,i])
    bp[:,i] = lfilter(b4,a4,lp[:,i])

    AP[:,i] = filtfilt(b2,a2, bp[:,i])
    LFP[:,i] = filtfilt(b,a,  bp[:,i])


# Illustrate the LFP components of the signal
fig = plt.figure(figsize=(10, 5))
```

```
plt.plot(tx[:5000],   x[:5000,j], color='#3B7DA6')
plt.plot(tx[:5000], LFP[:5000,j], color='#DF9AAF')
plt.xlabel('time [s]')
#plt.tight_layout()
fig.suptitle('Low-pass filtered signal [300Hz]', fontsize=20)
fig.legend(['Original signal', 'Low-pass filtered'], loc='upper right', bbox_to_


# Illustrate the EAPcomponents of the signal
fig = plt.figure( figsize=(10, 5))
plt.plot(tx[:],  x[:,j], color='#3B7DA6')
plt.plot(tx[:], AP[:,j], color='#DF9AAF')
plt.xlabel('time [s]')
#axs[1].set_ylabel('magnitude')
#plt.tight_layout()
fig.suptitle('High-pass filtered signal [300Hz]', fontsize=20)
fig.legend(['Original signal', 'High-pass filtered'], loc='upper right', bbox_to
```

Out[ ]: <matplotlib.legend.Legend at 0x1ee9c6b2390>

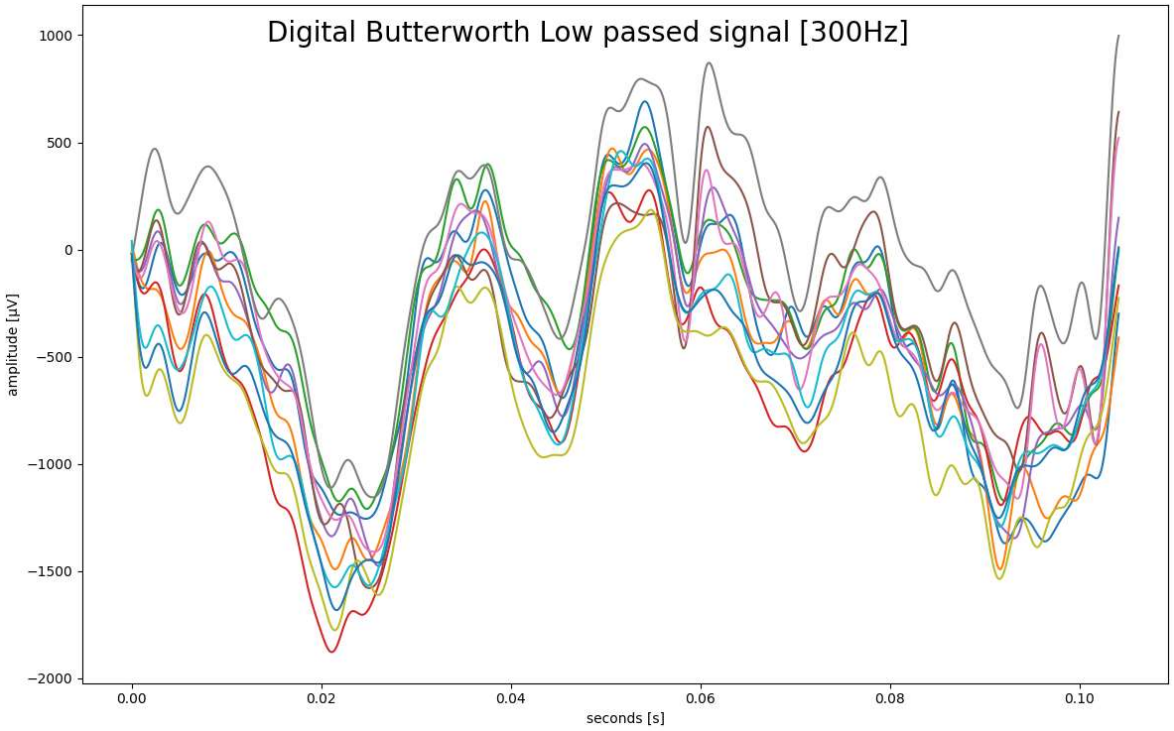


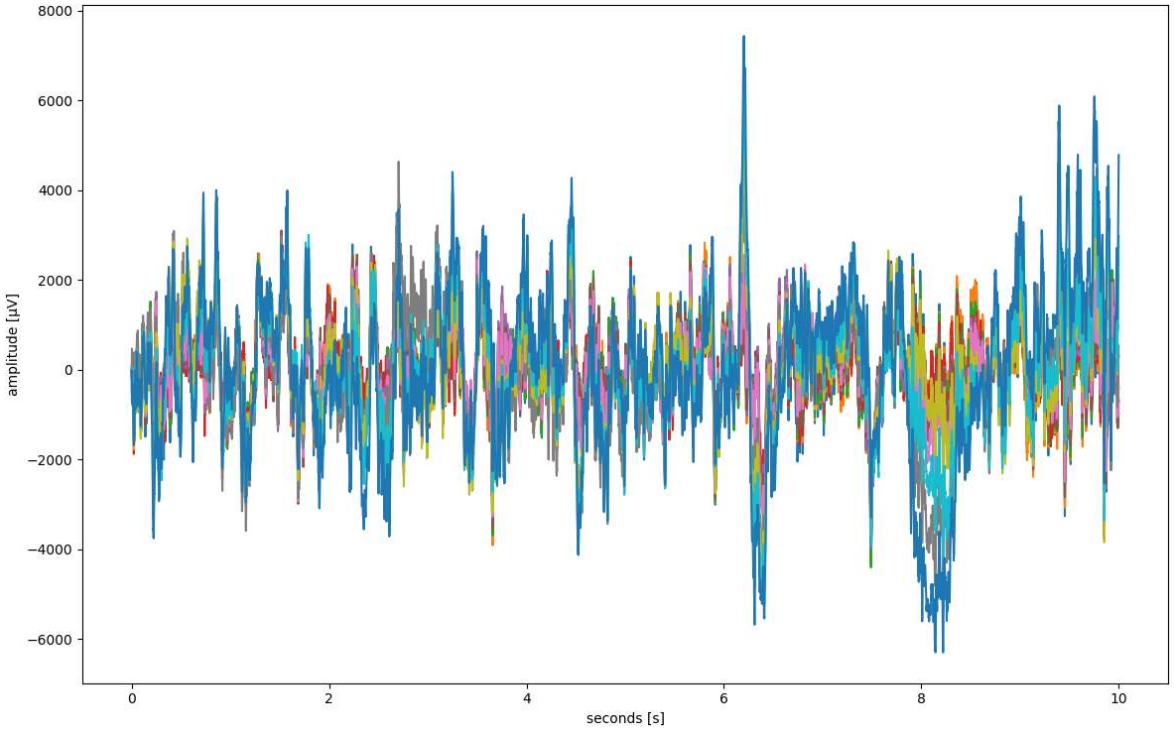


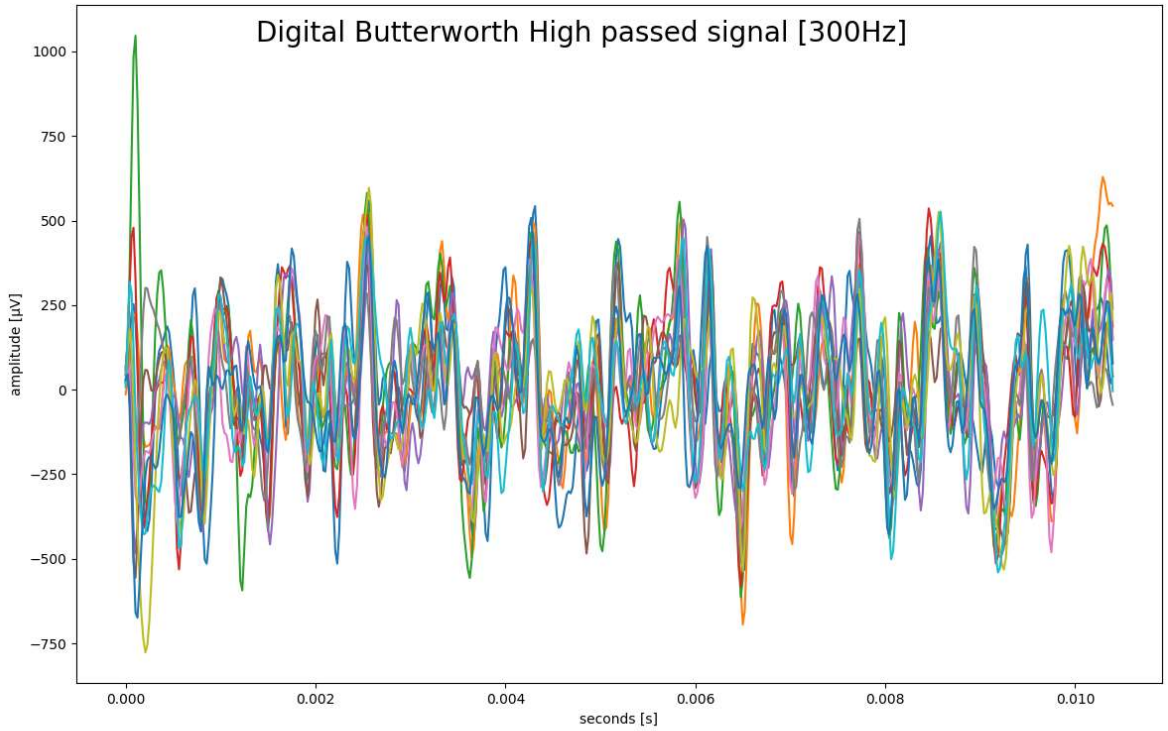
In [ ]:
```
# illustrate the effects on all tetrode channels

# Illustrate the LFP components of the signal
fig, axs = plt.subplots(2,1, figsize=(12, 15))
axs[0].plot(tx[:5000], LFP[:5000,:11])
axs[0].set_xlabel('seconds [s]')
```

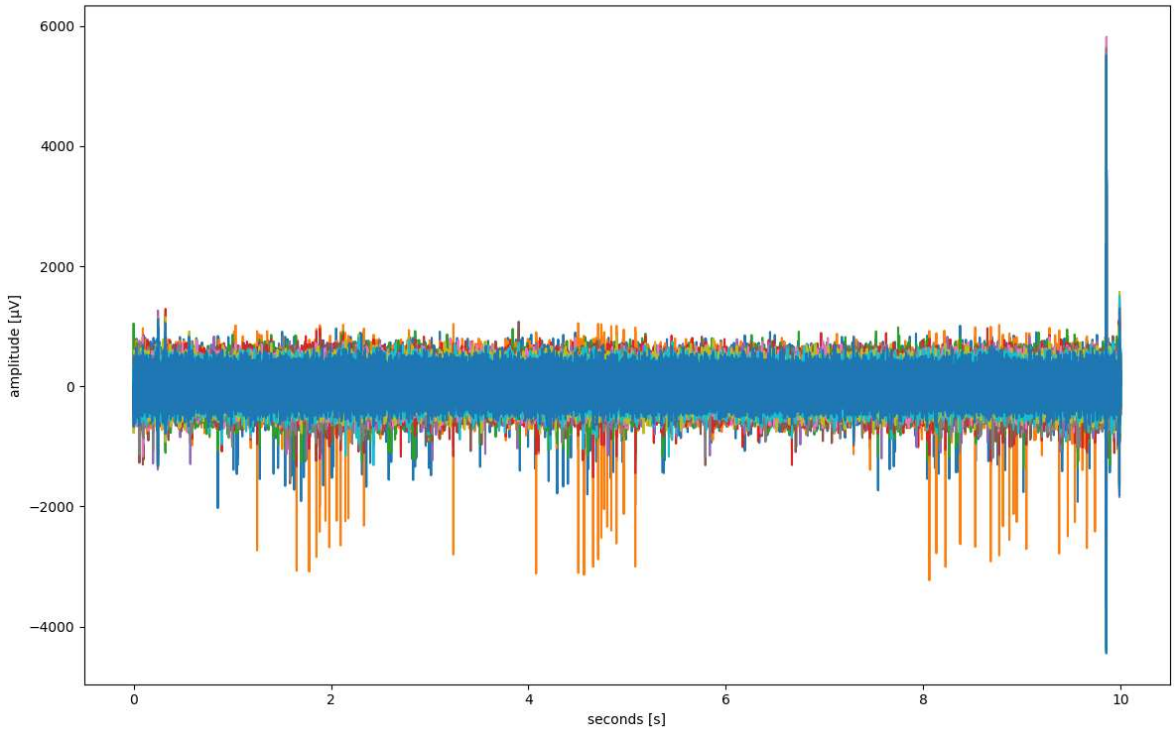
```
axs[0].set_ylabel('amplitude [µV]')
axs[1].plot(tx[:], LFP[:,:11])
axs[1].set_xlabel('seconds [s]')
axs[1].set_ylabel('amplitude [µV]')
plt.tight_layout()
fig.suptitle('Digital Butterworth Low passed signal [300Hz]', fontsize=20)


# Illustrate the AP components of the signal
fig, axs = plt.subplots(2,1, figsize=(12, 15))
axs[0].plot(tx[:500], AP[:500,:11])
axs[0].set_xlabel('seconds [s]')
axs[0].set_ylabel('amplitude [µV]')
axs[1].plot(tx[:], AP[:,:11])
axs[1].set_xlabel('seconds [s]')
axs[1].set_ylabel('amplitude [µV]')
plt.tight_layout()
fig.suptitle('Digital Butterworth High passed signal [300Hz]', fontsize=20)
```

Out[ ]: Text(0.5, 0.98, 'Digital Butterworth High passed signal [300Hz]')


Digital Butterworth Low passed signal [300Hz]
amplitude [µV]
1000
500
0
−500
−1000
−1500
−2000
0.00
0.02
0.04
0.06
0.08
0.10
seconds [s]


amplitude [µV]
8000
6000
4000
2000
0
−2000
−4000
−6000
0
2
4
6
8
10
seconds [s]





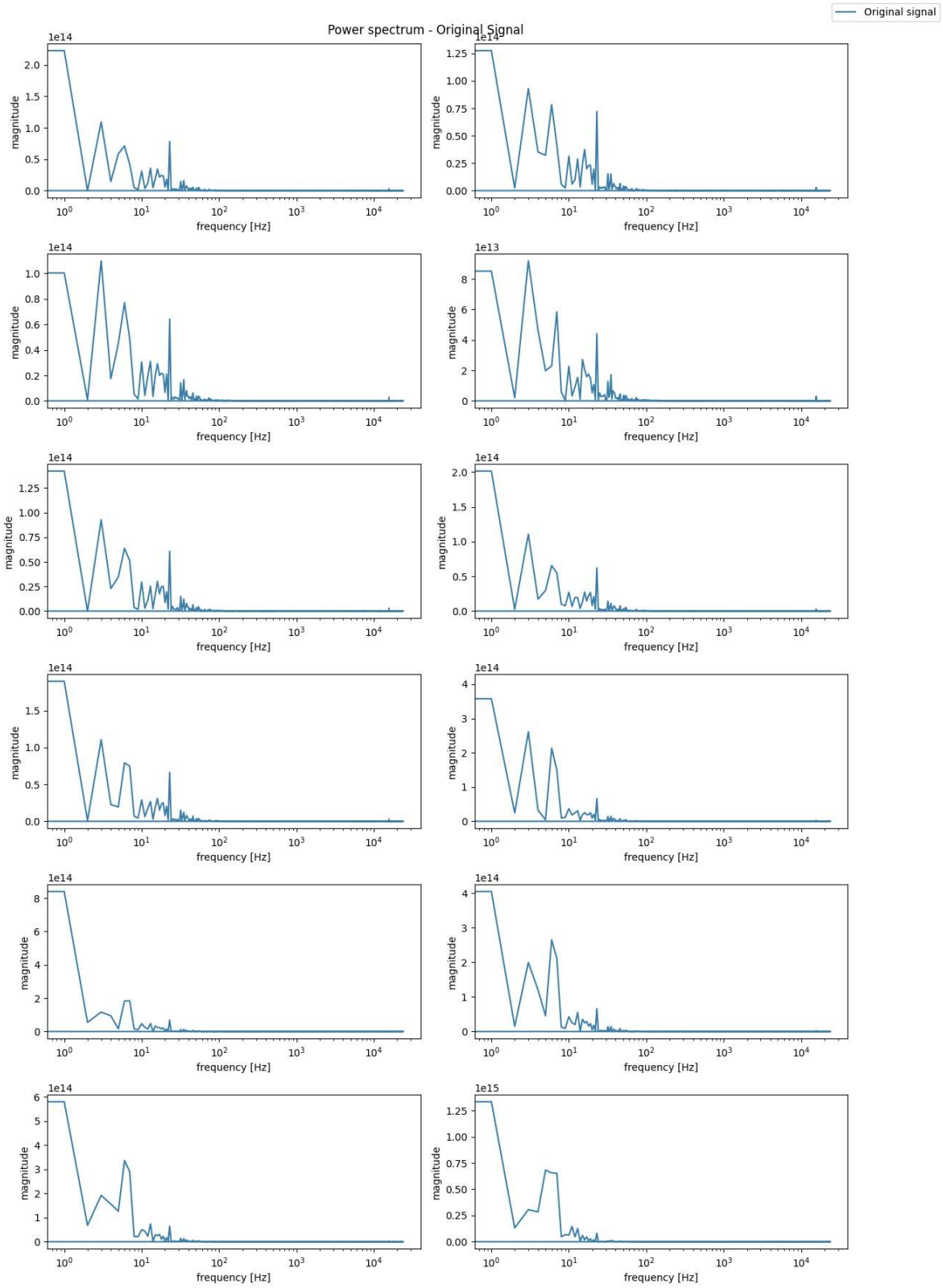
We can see how there is a clear shared shape for the LFP signals of the different channels, but that the AP signals differes more from one channel to another

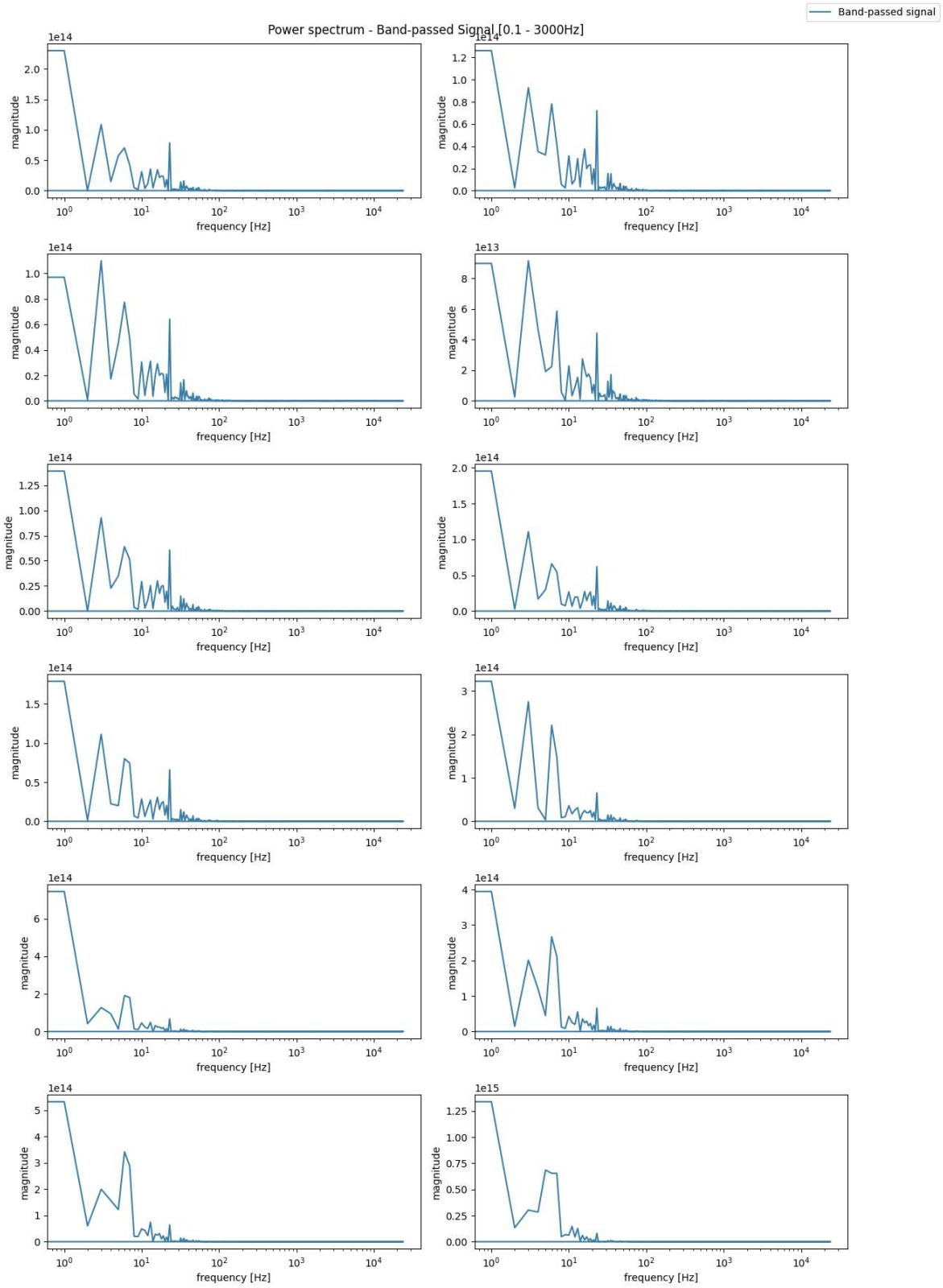
Frequency components:

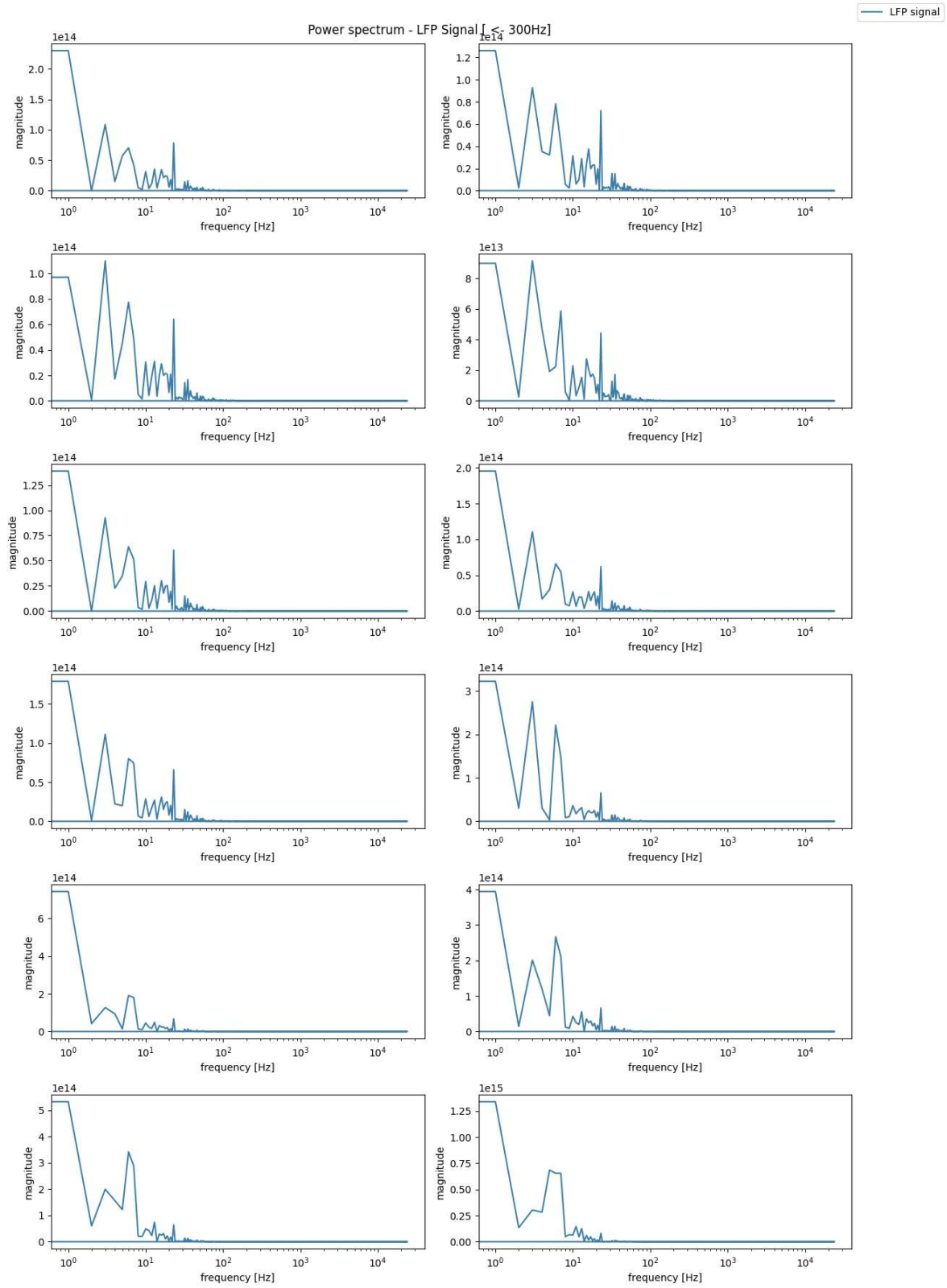
In [ ]:
```
t = np.linspace(0,1,48000) # use one second

fig1, axs1 = plt.subplots(6, 2, figsize=(12, 18))
fig2, axs2 = plt.subplots(6, 2, figsize=(12, 18))
fig3, axs3 = plt.subplots(6, 2, figsize=(12, 18))
fig4, axs4 = plt.subplots(6, 2, figsize=(12, 18))
for i in range(s.shape[1] - 4 ):
    fft_s = np.fft.fft(s[:48000,i])
```

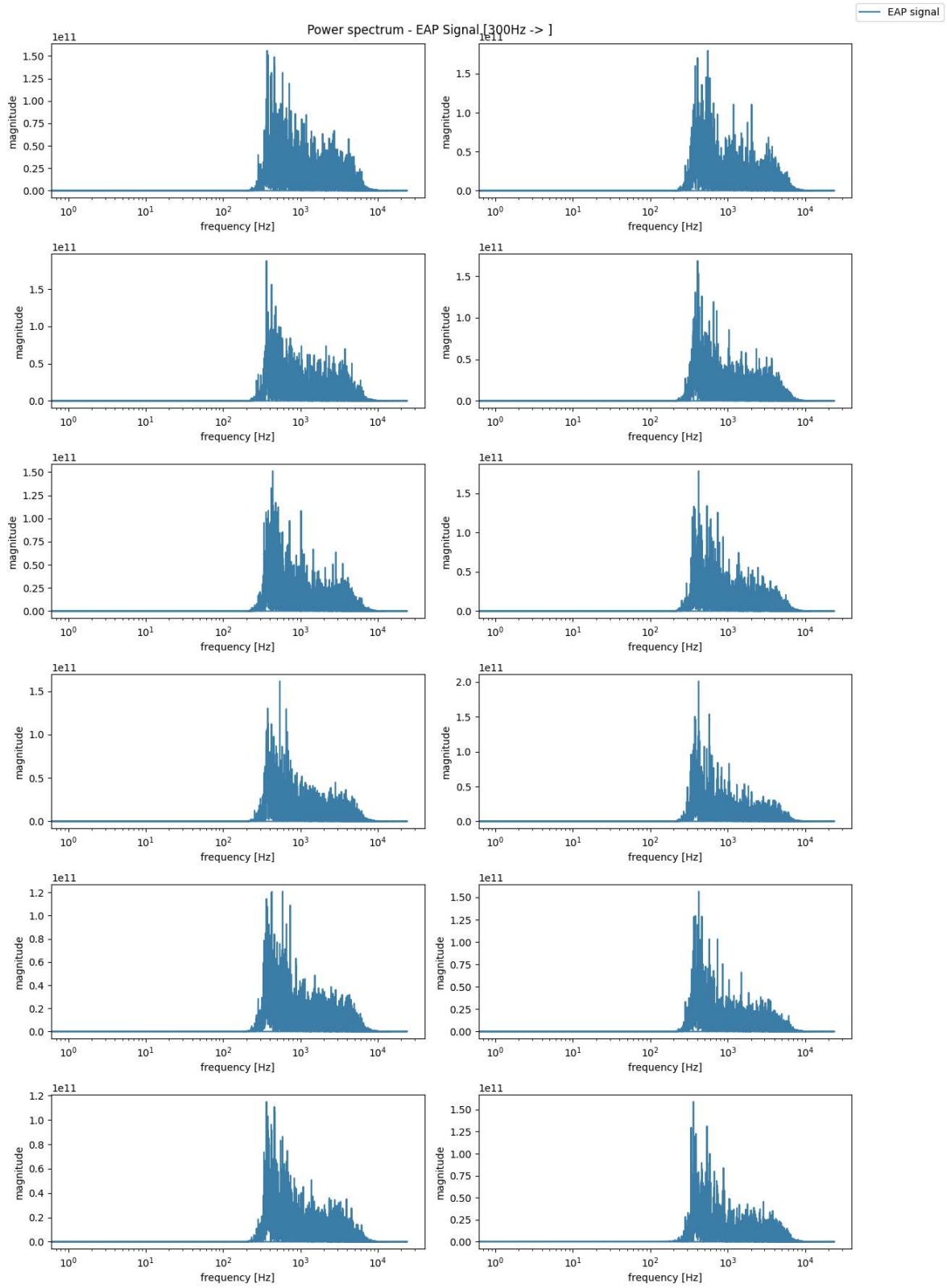
```
    P_s = np.abs(fft_s) ** 2
    freqss = np.fft.fftfreq(len(s[:48000,i]), t[1] - t[0])
    fft_bp = np.fft.fft(bp[:48000,i])
    P_bp = np.abs(fft_bp) ** 2
    fft_LFP = np.fft.fft(LFP[:48000,i])
    P_LFP = np.abs(fft_LFP) ** 2
    fft_AP= np.fft.fft(AP[:48000,i])
    P_AP= np.abs(fft_AP) ** 2

    #axs1.flatten()[i].plot(freqss[:], 20*np.log10(P[:]), 'b')
    axs1.flatten()[i].semilogx(freqss[:], P_s[:], color='#3B7DA6')
    axs1.flatten()[i].set_xlabel('frequency [Hz]')
    axs1.flatten()[i].set_ylabel('magnitude')

    axs2.flatten()[i].semilogx(freqss[:], P_bp[:], color='#3B7DA6')
    axs2.flatten()[i].set_xlabel('frequency [Hz]')
    axs2.flatten()[i].set_ylabel('magnitude')

    axs3.flatten()[i].semilogx(freqss[:], P_LFP[:], color='#3B7DA6')
    axs3.flatten()[i].set_xlabel('frequency [Hz]')
    axs3.flatten()[i].set_ylabel('magnitude')

    axs4.flatten()[i].semilogx(freqss[:], P_AP[:], color='#3B7DA6')
    axs4.flatten()[i].set_xlabel('frequency [Hz]')
    axs4.flatten()[i].set_ylabel('magnitude')

fig1.suptitle('Power spectrum - Original Signal')
fig2.suptitle('Power spectrum - Band-passed Signal [0.1 - 3000Hz]')
fig3.suptitle('Power spectrum - LFP Signal [ <- 300Hz]')
fig4.suptitle('Power spectrum - EAP Signal [300Hz -> ]')
fig1.tight_layout()
fig2.tight_layout()
fig3.tight_layout()
fig4.tight_layout()
fig1.legend(['Original signal'], loc='upper right', bbox_to_anchor=(1.1, 1))
fig2.legend(['Band-passed signal'], loc='upper right', bbox_to_anchor=(1.1, 1))
fig3.legend(['LFP signal'], loc='upper right', bbox_to_anchor=(1.1, 1))
fig4.legend(['EAP signal'], loc='upper right', bbox_to_anchor=(1.1, 1))
```

Out[ ]: <matplotlib.legend.Legend at 0x1ef467a1350>


Original signal
Power spectrum - Original Signal
magnitude
frequency [Hz]
1e14
1e13
1e15


Band-passed signal
Power spectrum - Band-passed Signal [0.1 - 3000Hz]
magnitude
frequency [Hz]
1e14
1e13
1e15


LFP signal
Power spectrum - LFP Signal [<- 300Hz]
magnitude
frequency [Hz]
1e14
1e13
1e15



Notice the 10kHz noise that is present in the orignal signal is supressed after band pass filtering. Furthermore, we can also see that the EAP signals have relatively similar frequency components even though the APs happen at different times in the different signals

APPENDIX

E

PCA IMPLEMENTATION AND ANALYSIS

In this notebook I will illustrate how PCA can be computed and how much information lies in the different coefficients

About PCA:

- PCA garantees output vectors that are un-correlated with each other
- SVD automatically outputs the sorted eigenvectors (principal components)
- We remove the mean to make sure that the offset does not get interpreted as a direction of variance

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

s = np.load('../test_data_sintef.npy')
```

PCA

In [ ]:
```
# Find the principal components using three different methods and compare

temp = s[:500,:].astype(int)

# Normalize the data:
mean = np.mean(temp, axis=0)

mean_temp = temp - mean

# find PCA using eigenvalues (built with the help of: https://towardsdatascience
cov = np.cov(mean_temp.T)
eig_val, eig_vec = np.linalg.eig(cov)

# sort eigenvalues
indices = np.arange(0,len(eig_val),1)
indices = ([x for _,x in sorted(zip(eig_val, indices))])[::-1]
eig_val = eig_val[indices]
eig_vec = eig_vec[:,indices]

pca_temp_1 = mean_temp@eig_vec


# find PCA using SVD
U,sigma,V = np.linalg.svd(mean_temp.T@mean_temp)
np.allclose(V.T, U) # this should be true, if not something is incorrect
pca_temp_2 = mean_temp@U



# find PCA using library
pca = PCA()
pca_temp_3 = pca.fit_transform(temp)
```

```
# compare resulting coefficients on three coefficients
print(f'Method 1 gives same as method 2: {np.allclose(np.abs(pca_temp_1), np.abs
print(f'Method 1 gives same as method 3: {np.allclose(np.abs(pca_temp_1), np.abs
```

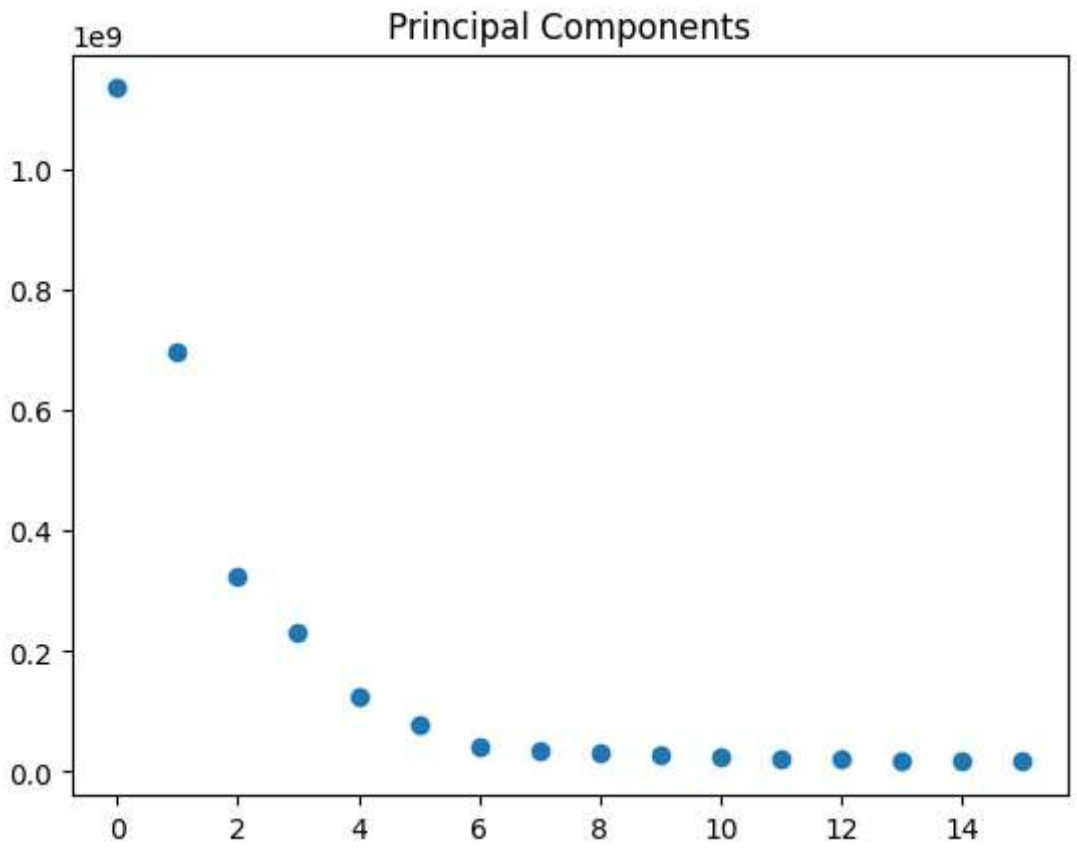
```
Method 1 gives same as method 2: True
Method 1 gives same as method 3: True
```

We see that all methods give the same result. Next we check how much each principal components contributes

In [ ]:
```
# Display contribution of each singular value

sum_sigma = np.sum(sigma)
explained_variance = sigma/ sum_sigma
print(f'explained variance: \n{explained_variance} \n')
cumulative_variance = np.cumsum(explained_variance)
print(f'cumulative variance: \n{cumulative_variance} \n')

plt.title('Principal Components')
plt.scatter(np.arange(0,len(sigma)),sigma)
```

```
explained variance:
[0.39915587 0.2454873  0.11401362 0.08079757 0.04374768 0.02732161
 0.01494193 0.01185748 0.01090962 0.0098645  0.0085061  0.00742084
 0.00698536 0.00665654 0.00651502 0.00581893]

cumulative variance:
[0.39915587 0.64464317 0.75865679 0.83945436 0.88320205 0.91052366
 0.92546559 0.93732308 0.9482327  0.9580972  0.9666033  0.97402415
 0.9810095  0.98766605 0.99418107 1.        ]
```

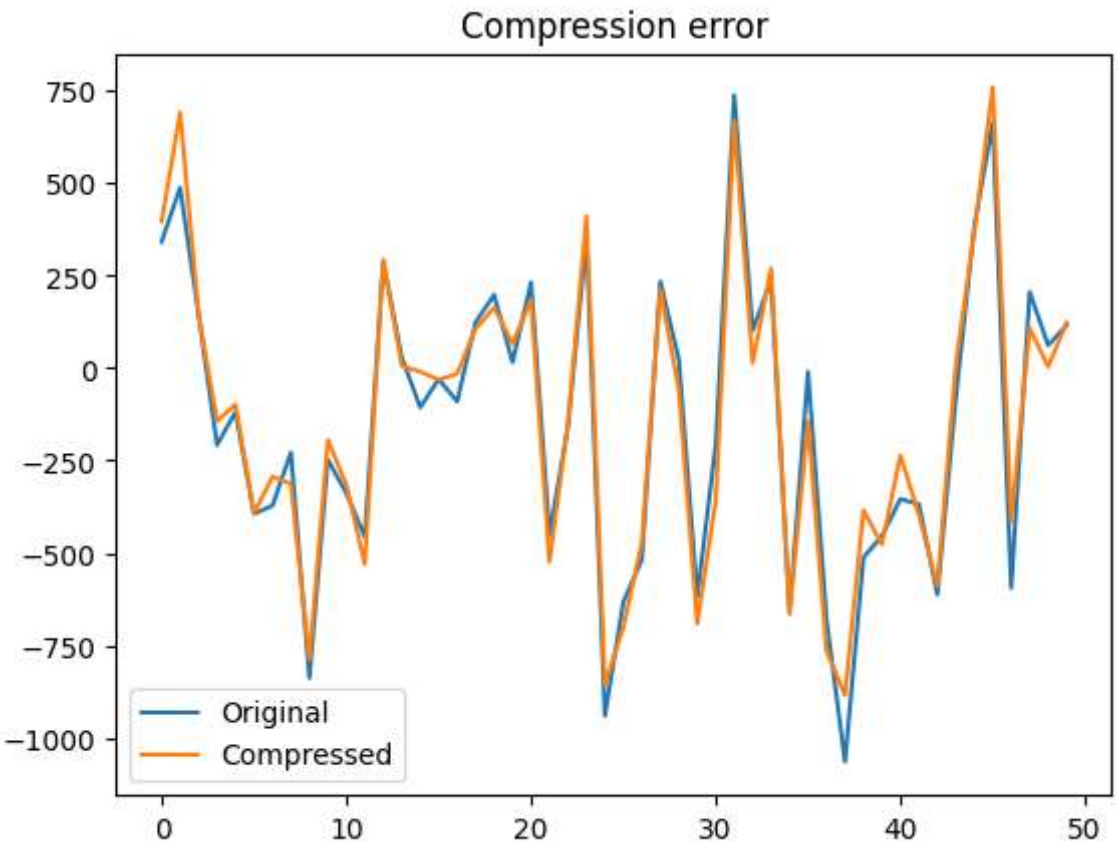
Out[ ]: <matplotlib.collections.PathCollection at 0x2ac9772f650>



The results indicate that the PCA does not compress particularly well, since we need 10 of 16 coefficients to maintain 95% of the variance. Next we se how well we can reconstruct the signal without these six coefficients

In [ ]:
```
# remove coefficients and reconstruct the signal:
n_remove = 6

# compute PCA and remove n_remove coefficients
pca_th = np.pad(mean_temp@U[:,:-n_remove], ((0,0), (0,n_remove)), 'constant', cc

# re-construct signal
temp_hat = pca_th@U.T + mean

# plot result and display SNDR
plt.title('Compression error')
plt.plot(temp[:50,0])
plt.plot(temp_hat[:50,0])
plt.legend(['Original', 'Compressed'])

SNDR = 0
for i in range(temp.shape[1]):
    SNDR += 1/(temp.shape[1]) * 20*np.log10(np.linalg.norm(temp[:,i])/np.linalg.


print(f'SNDR: {SNDR:.2f} dB \n')
```

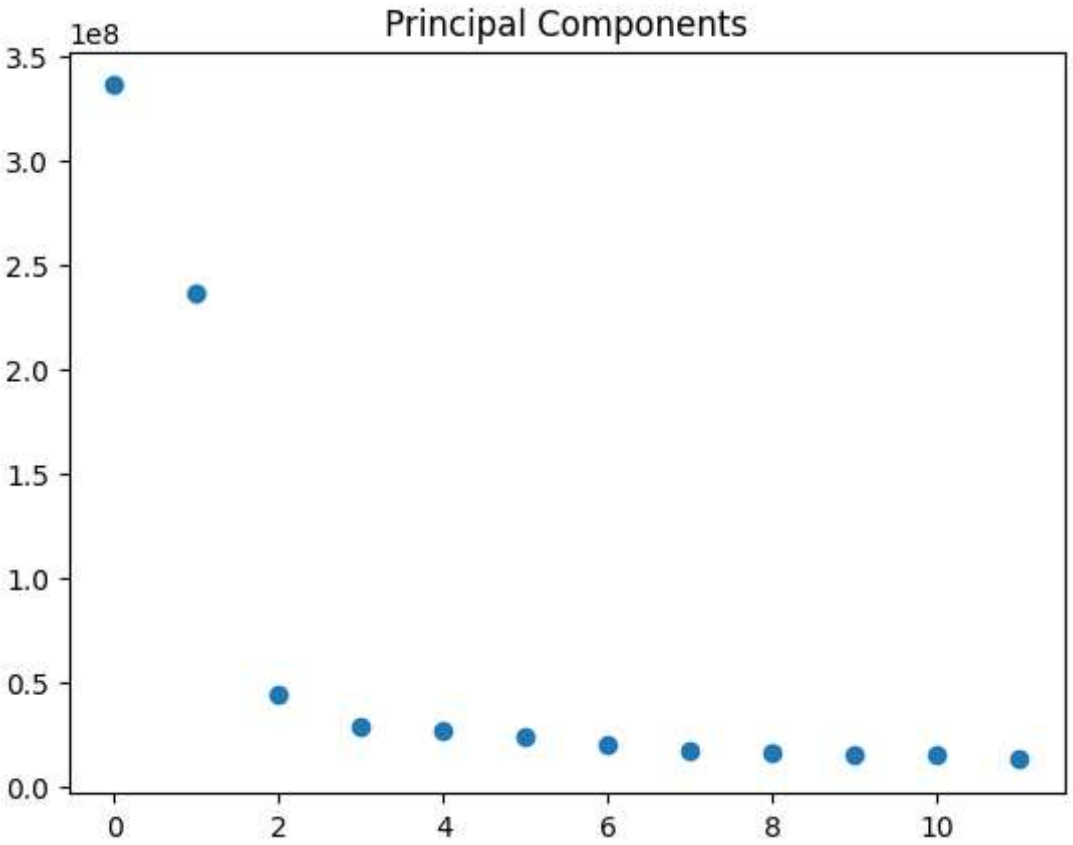
```
SNDR: 11.96 dB
```

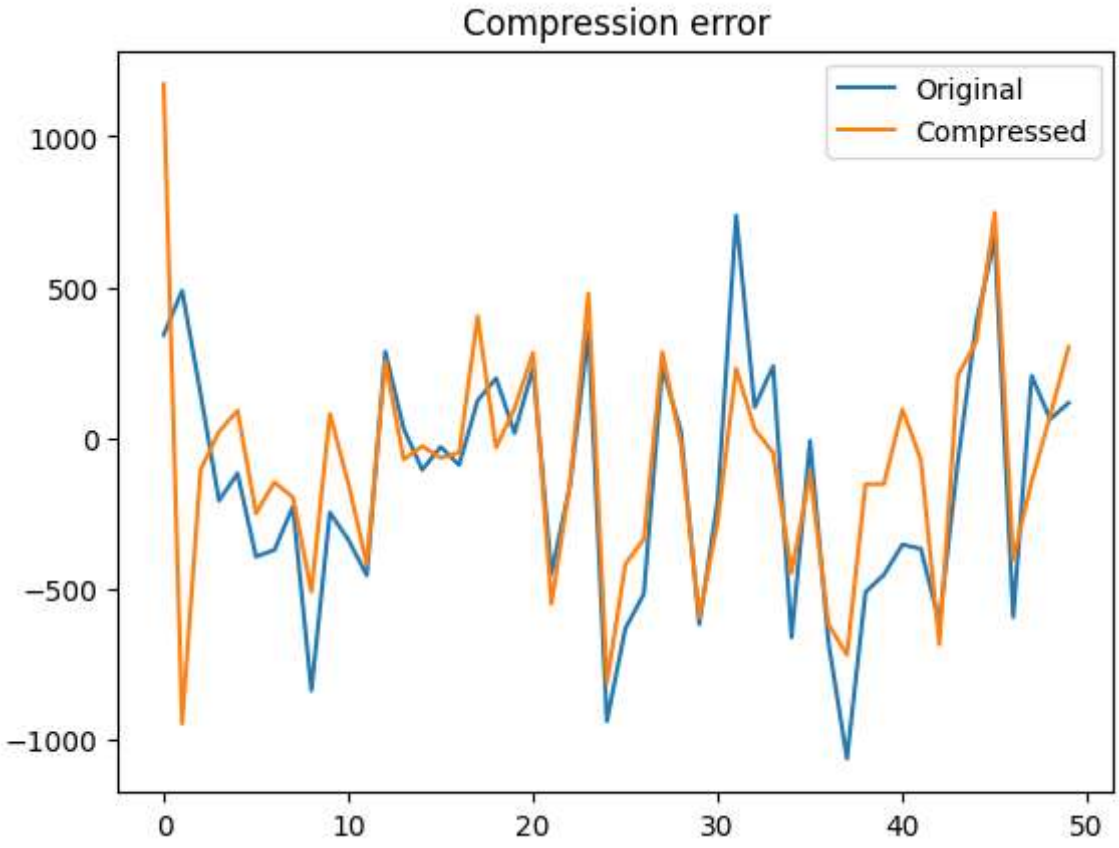


This indicates that the channels are not particularly correlated, if we compute the PCA on only the tetrode channels we get the following results:

In [ ]:
```python
# Find the principal components using three different methods and compare

temp = s[:400,:12].astype(int)

# Normalize the data:
mean = np.mean(temp, axis=0)

mean_temp = temp - mean

# find PCA using SVD
U,sigma,V = np.linalg.svd(mean_temp.T@mean_temp)
np.allclose(V.T, U) # this should be true, if not something is incorrect
pca_temp_2 = mean_temp@U


sum_sigma = np.sum(sigma)
explained_variance = sigma/ sum_sigma
print(f'explained variance: \n{explained_variance} \n')
cumulative_variance = np.cumsum(explained_variance)
print(f'cumulative variance: \n{cumulative_variance} \n')

plt.title('Principal Components')
plt.scatter(np.arange(0,len(sigma)),sigma)

# remove coefficients and reconstruct the signal:
n_remove = temp.shape[1] - 2

# compute PCA and remove n_remove coefficients
pca_th = np.pad(mean_temp@U[:,:-n_remove], ((0,0), (0,n_remove)), 'constant', cc

# re-construct signal
temp_hat = pca_th@U.T + mean

# plot result and display SNDR
plt.figure()
plt.title('Compression error')
plt.plot(temp[:50,0])
plt.plot(temp_hat[:50,0])
plt.legend(['Original', 'Compressed'])

SNDR = 0
for i in range(temp.shape[1]):
    SNDR += 1/(temp.shape[1]) * 20*np.log10(np.linalg.norm(temp[:,i])/np.linalg.
print(f'SNDR: {SNDR:.2f} dB \n')
```

```
explained variance:
[0.42258149 0.29778246 0.05545045 0.03619654 0.03362738 0.03014262
 0.0255747  0.02182578 0.02111957 0.01954323 0.01948524 0.01667055]

cumulative variance:
[0.42258149 0.72036395 0.7758144  0.81201094 0.84563832 0.87578093
 0.90135564 0.92318141 0.94430098 0.96384421 0.98332945 1.        ]

SNDR: 8.03 dB
```





Here we can see a clear elbow after two coefficients, so we can get decent reconstuction using only two coefficients. But this error is most likely more then what the researchers would like

APPENDIX

F

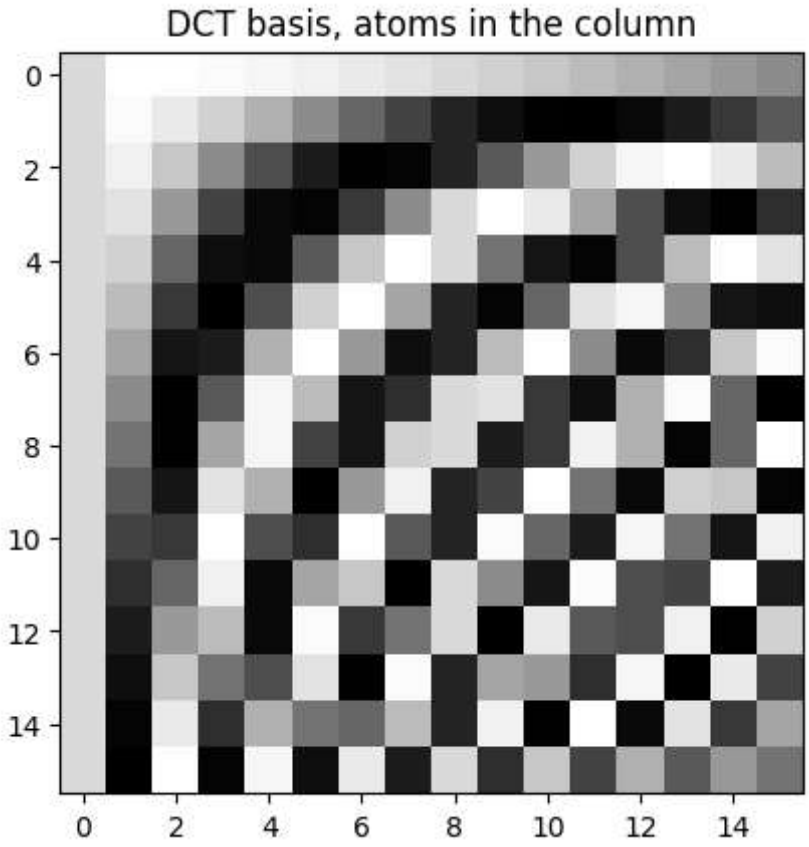
SPATIAL DCT IMPLEMENTATION AND ANALYSIS

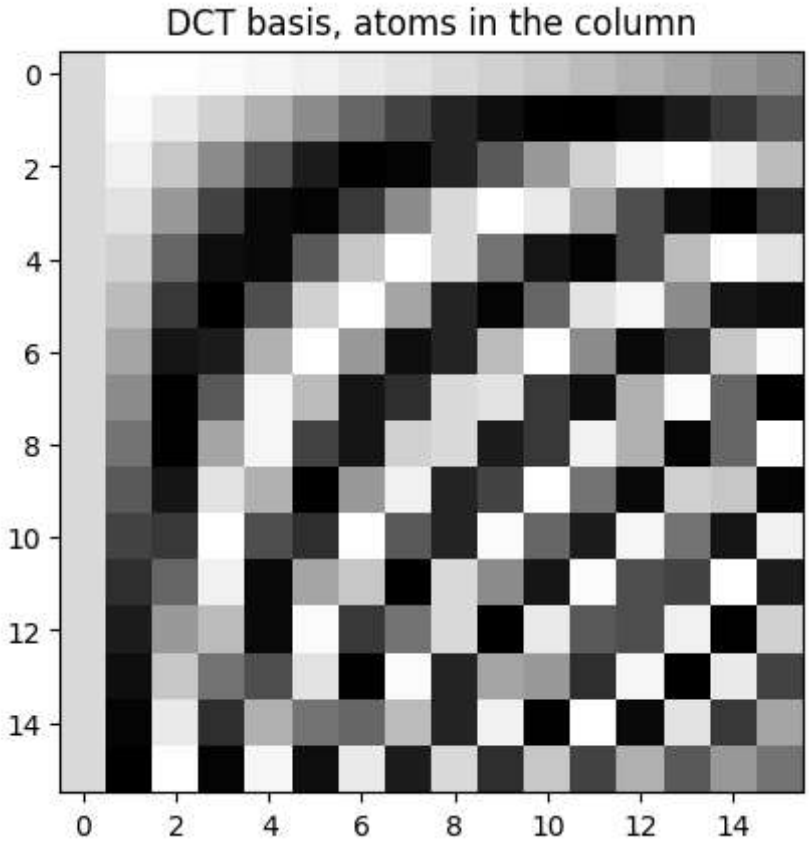
Here I will show how well DCT performs on spatial decorrelation using one segment of the signal. I also illustrate the effects of removing the coefficients of the highest frequencies.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import idct
import warnings
warnings.simplefilter("ignore")

s = np.load('../test_data_sintef.npy')
```

In [ ]:
```
# generate DCT basis
pp = s.shape[1]

D = np.zeros((pp, pp))

# generate DCT basis using library
for k in range(pp):
    a = np.zeros(pp)
    a[k] = 1
    D[:, k] = idct(a, norm='ortho') # use idct on the atom

plt.figure()
plt.imshow(D, cmap='gray')
plt.title('DCT basis, atoms in the column')


# generate DCT basis using equation
DCT = np.zeros((pp, pp))

c_0 = np.sqrt(1/pp)
c_k = np.sqrt(2/pp)
for k in range(pp):
    if (k == 0):
        DCT[:, k] = [c_0*np.cos(k*np.pi*(2*n+1)/(2*pp)) for n in range(pp)]
    else:
        DCT[:, k] = [c_k*np.cos(k*np.pi*(2*n+1)/(2*pp)) for n in range(pp)]
    DCT[:, k] = DCT[:, k] / np.linalg.norm(DCT[:, k], ord = 2)

plt.figure()
plt.imshow(DCT, cmap='gray')
plt.title('DCT basis, atoms in the column')
```

Out[ ]:
```
Text(0.5, 1.0, 'DCT basis, atoms in the column')
```





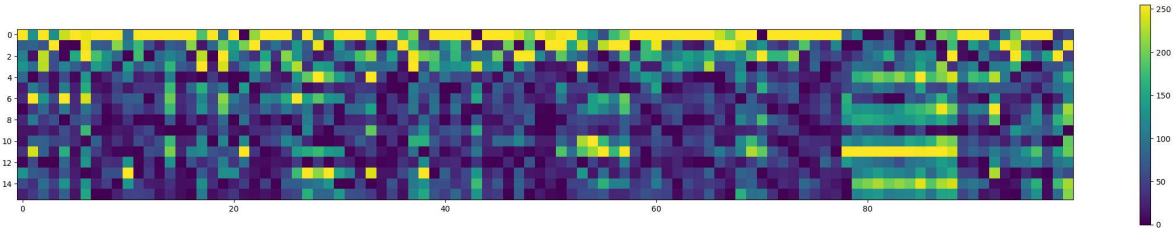
We see that the two DCT basis are identical. Next we compute the coefficients

In [ ]:
```
# this takes the DCT across the channels
def getDCT(temp, D):
    c = np.zeros((temp.shape[1], temp.shape[0]))

    for i in range(temp.shape[0]):
```

```
        c[:, i] = D.T@temp[i,:]

    return c
```

In [ ]:
```
temp = s.astype(int)

# function to normalize each column of the dct coefficients
def visualize_dct(c):
    for i in range(c.shape[0]):
        c[i,:] = 255* (c[i,:] - c[i,:].min()) / (c[i,:].max() - c[i,:].min())
    return c

# perform DCT
c = getDCT(temp, DCT)

# reduce the number of columns for better visualization
M = 100
c_new = np.zeros((c.shape[0], M))
step = c.shape[1] // M
for i in range(M):
    c_new[:,i] = np.mean(c[:,i*step:i*step + step], axis=1)

# visulaize the DCT coefficients
plt.figure(figsize=[30,5])
plt.imshow(visualize_dct(abs(c_new).T).T)
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x215ff65f490>
```



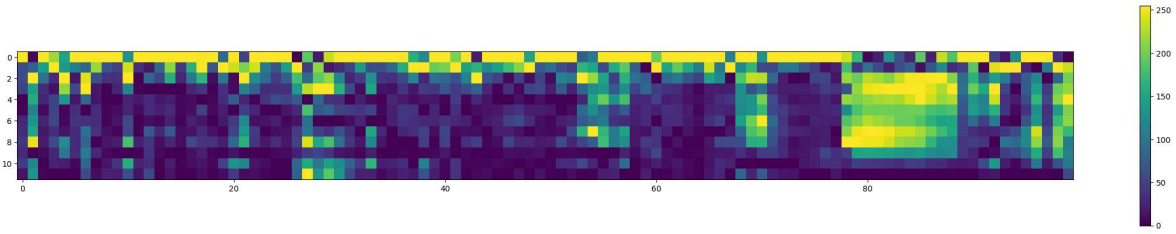
We see that most of the information is contained in the lowest freqencies, but that there still is some important information in higher freqencies too.

We then test on just the tetrode channels:

In [ ]:
```
temp = s[:,:12].astype(int)

# generate DCT basis
pp = temp.shape[1]

# compute new DCT basis that is 12x12
DCT = np.zeros((pp, pp))
c_0 = np.sqrt(1/pp)
c_k = np.sqrt(2/pp)
for k in range(pp):
    if (k == 0):
        DCT[:, k] = [c_0*np.cos(k*np.pi*(2*n+1)/(2*pp)) for n in range(pp)]
    else:
        DCT[:, k] = [c_k*np.cos(k*np.pi*(2*n+1)/(2*pp)) for n in range(pp)]
    DCT[:, k] = DCT[:, k] / np.linalg.norm(DCT[:, k], ord = 2)

# perform DCT
```

```
c = getDCT(temp, DCT)

# reduce num columns
M = 100
c_new = np.zeros((c.shape[0], M))
step = c.shape[1] // M
for i in range(M):
    c_new[:,i] = np.mean(c[:,i*step:i*step + step], axis=1)

# visulaize the DCT coefficients
plt.figure(figsize=[30,5])
plt.imshow(visualize_dct(abs(c_new).T).T)
plt.colorbar()
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x215ff6b5c10>



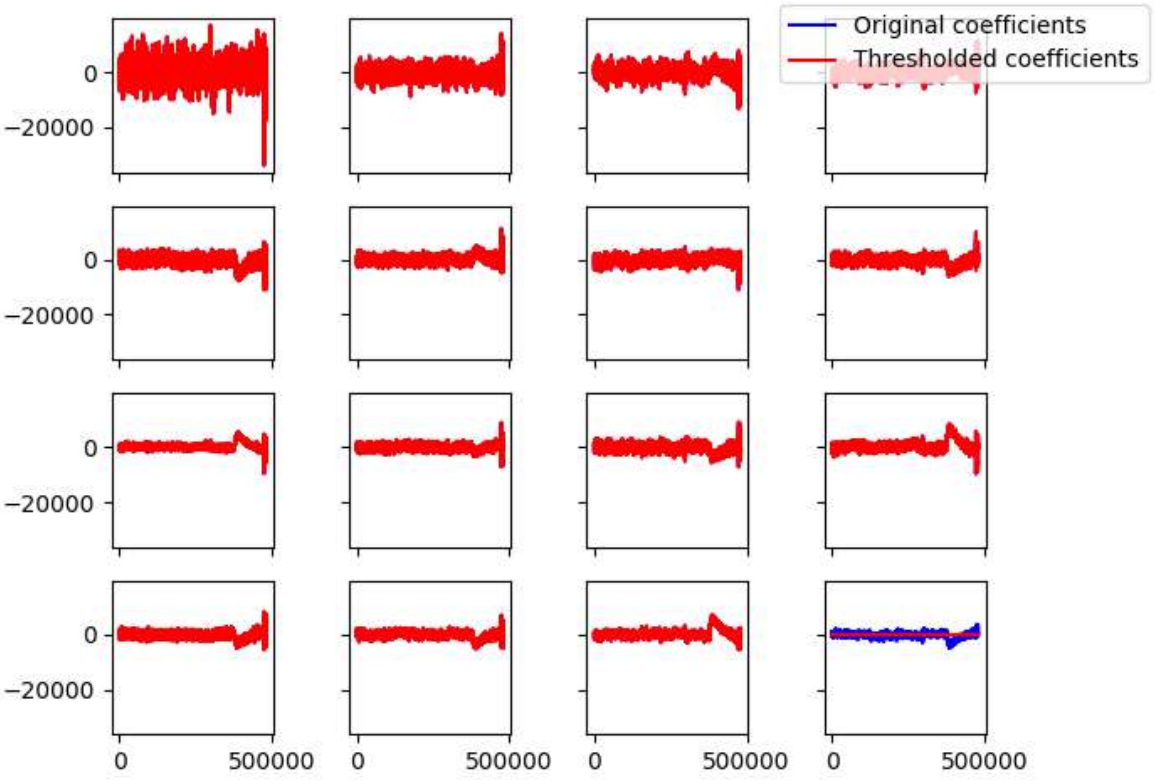
We see that the result is more sparse, as expected

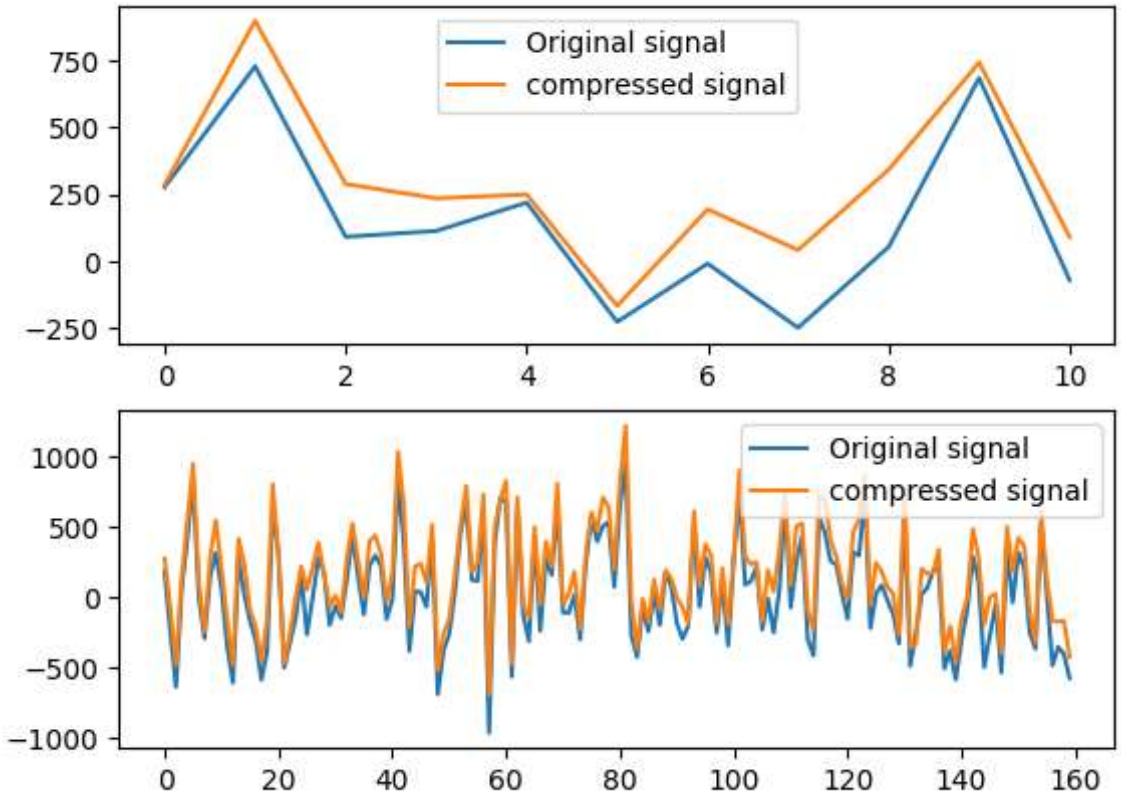
Threshold coefficients

Threshold to see how much loss we get from removing the highest frequency coefficients

In [ ]:
```
# threshold coefficients
n_remove = 1

temp = s.astype(int)

# threshold coefficients
c = getDCT(temp, D)

# remove the n_remove highest frequencies
c_th = c[:-n_remove, :]
c_th_full = np.pad(c_th, ((0,n_remove), (0,0)), 'constant', constant_values=(0))

# Visualize the coefficients
fig, axs = plt.subplots(4, 4, sharey=True, sharex=True)
for i in range(temp.shape[1]):
    axs.flatten()[i].plot(c[i, :], 'b')
    axs.flatten()[i].plot(c_th_full[i,:], 'r')
fig.legend(['Original coefficients', 'Thresholded coefficients'], loc='upper rig
plt.tight_layout()
plt.show()

# reconstuct signal
temp_hat = (D@c_th_full).T

# plot error
fig2, axs2 = plt.subplots(2,1)
axs2[0].plot(temp[140:151,5])
axs2[0].plot(temp_hat[140:151,5])
```

```
axs2[0].legend(['Original signal', 'compressed signal'])
axs2[1].plot(temp[40:200,5])
axs2[1].plot(temp_hat[40:200,5])
axs2[1].legend(['Original signal', 'compressed signal'])
plt.show()

SNDR = 0
for i in range(temp.shape[1]):
    P_d = np.linalg.norm(temp[:,i])
    P_error = np.linalg.norm(temp[:,i] - temp_hat[:,i])

    SNDR += 1/(temp.shape[1]) * 20*np.log10(P_d/P_error)
print(f'Mean SNDR across channels: {SNDR:.2f} dB')
```






```
Mean SNDR across channels: 16.90 dB
```

APPENDIX

G

TEMPORAL DCT IMPLEMENTATION AND ANALYSIS

Here I will show how well DCT performs on temporal decorrelation using one segment of the signal. I also illustrate the effects of removing the coefficients of the highest frequencies.

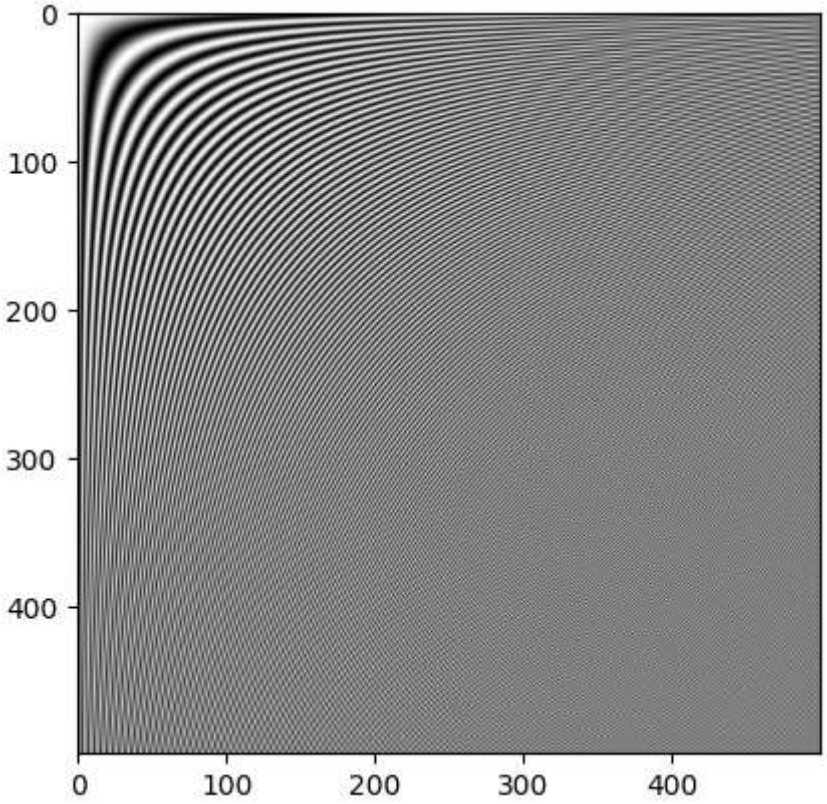
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import idct

s = np.load('../test_data_sintef.npy')
```

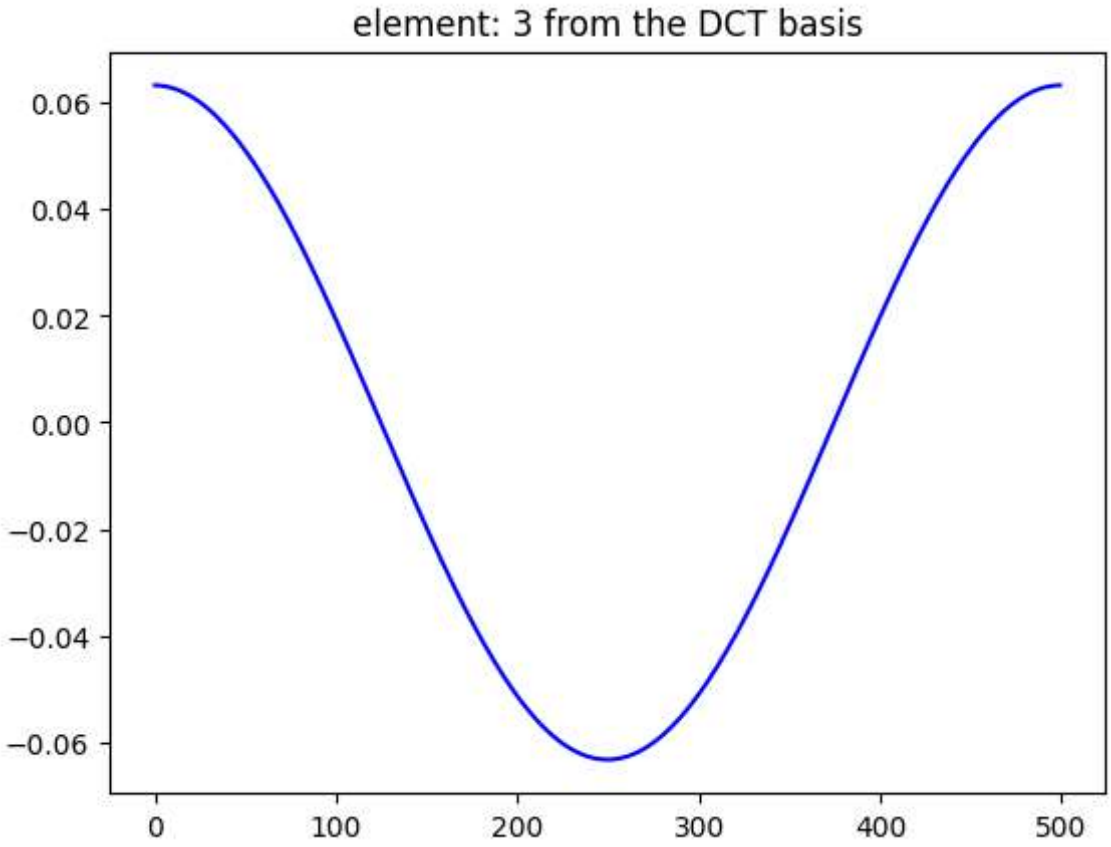
DCT temporal coding

setup

In [ ]:
```
# generate DCT basis
pp = 500

D = np.zeros((pp, pp))

# generate DCT basis
for k in range(pp):
    a = np.zeros(pp)
    a[k] = 1
    D[:, k] = idct(a, norm='ortho') # use idct on the atom

# plot DCT matrix
plt.figure()
plt.imshow(D, cmap='gray')

# plot one element from the DCT matrix
k = 2
plt.figure()
plt.plot(D[:, k], 'b')
plt.title(f'element: {k+1} from the DCT basis')
```

Out[ ]:
```
Text(0.5, 1.0, 'element: 3 from the DCT basis')
```





In [ ]:

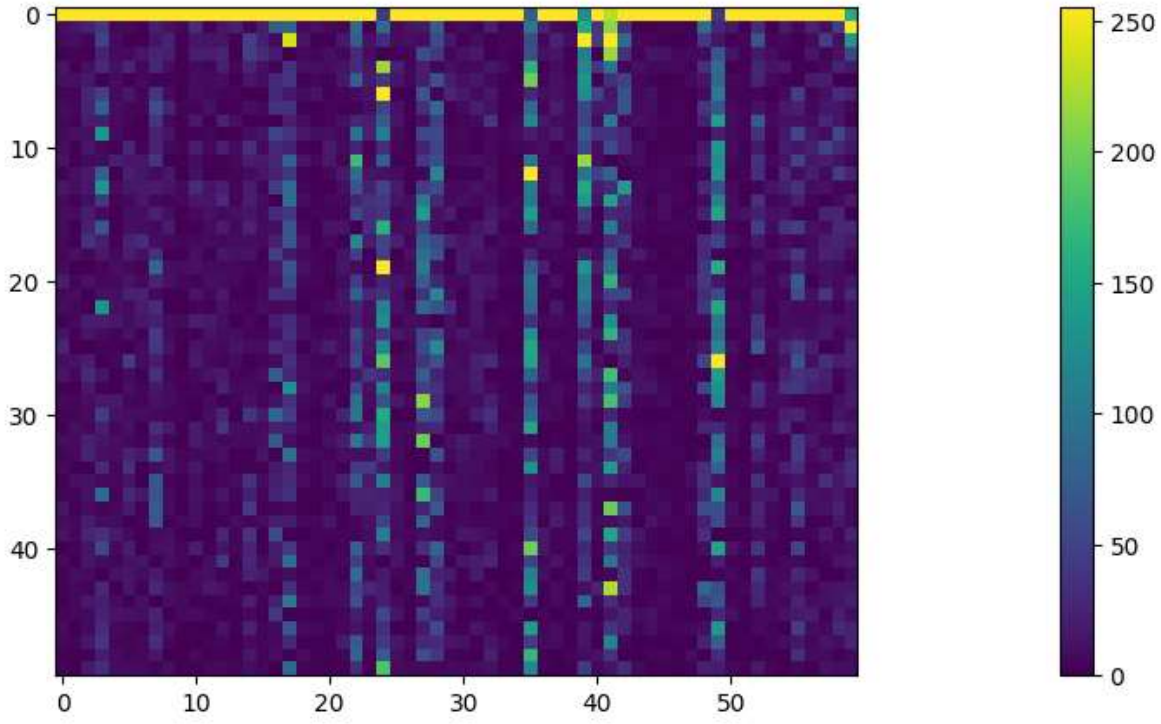
```python
def getDCT(temp):
    nSteps = int(s.shape[0]/pp)
    c_mat = np.zeros((pp, nSteps, s.shape[1]))

    for j in range(s.shape[1]):
        for i in range(nSteps):
            c_mat[:,i,j] = D.T@temp[pp*i:pp*i + pp, j]
```

```
    return c_mat

def getiDCT(temp):
    nSteps = int(s.shape[0]/pp)
    d_hat = np.zeros_like(s)

    for j in range(s.shape[1]):
        for i in range(nSteps):
            d_hat[pp*i:pp*i + pp, j] = D@temp[:,i,j]

    return d_hat
```

perform

In [ ]:
```
temp = s.astype(int)

# function to normalize each column of the dct coefficients
def visualize_dct(c):
    for i in range(c.shape[0]):
        c[i,:] = 255* (c[i,:] - c[i,:].min()) / (c[i,:].max() - c[i,:].min())
    return c

# perform DCT
c_mat = getDCT(temp).astype(int)

# flatten coefficient matrix
c = c_mat.reshape(c_mat.shape[0]*c_mat.shape[1], c_mat.shape[2]).T

# reduce the number of columns and rows for better visualization
M = 60
N = 50
c_new = np.zeros((N,M))
step_col = c_mat.shape[1] // M
step_row = c_mat.shape[0] // (N)
for j in range(N):
    for i in range(M):
        c_new[j,i] = np.mean(c_mat[j*step_row :j*step_row + step_row, i*step_col


# visulaize the DCT coefficients
plt.figure(figsize=[30,5])
plt.imshow(visualize_dct(abs(c_new).T).T)
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ea68782750>
```



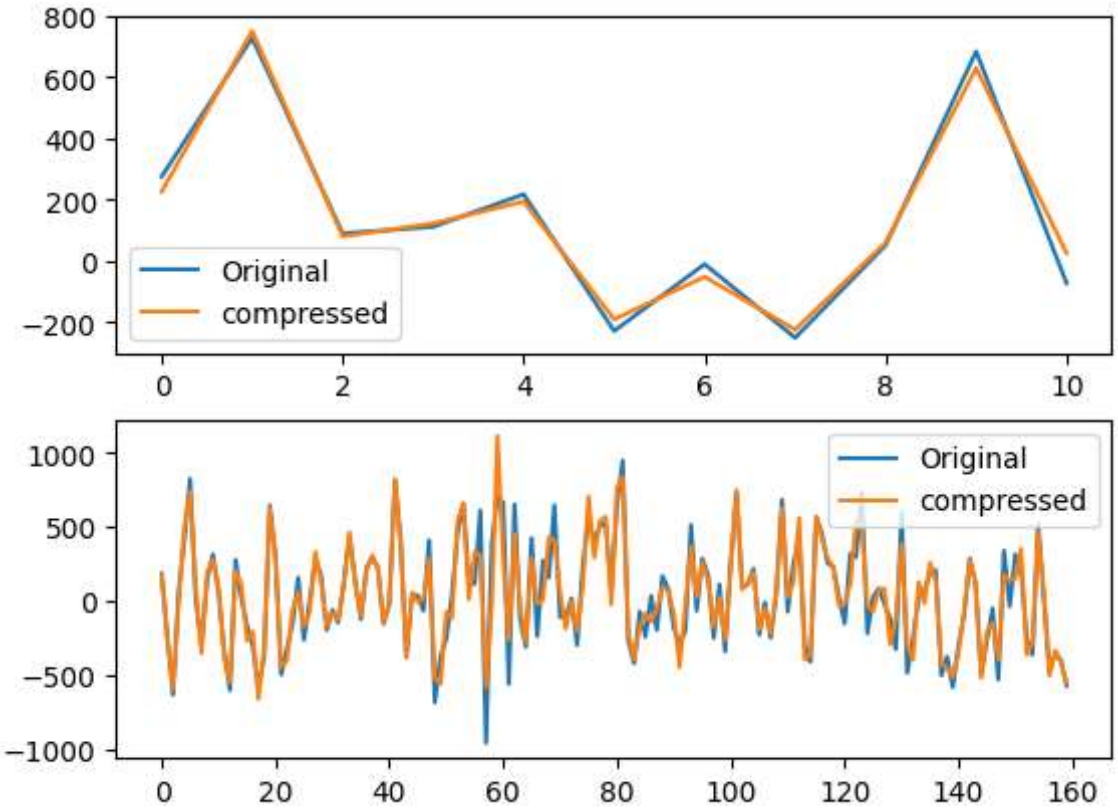
this shows the coefficients of the first channel, we see that the DC offset has the biggest contribution.

Threshold coefficients

Threshold to see how much loss we get from removing the highest frequency coefficients

In [ ]:
```
# threshold coefficients
n_remove = 100
c_mat = getDCT(temp).astype(int)

c_mat_th = c_mat[:-n_remove,:,:] # remove n_remove highest freqs


temp_hat = getiDCT(np.pad(c_mat_th, ((0,n_remove), (0,0), (0,0)), 'constant', cc

# plot
fig2, axs2 = plt.subplots(2,1)
axs2[0].plot(temp[140:151,5])
axs2[0].plot(temp_hat[140:151,5])
axs2[0].legend(['Original', 'compressed'])
axs2[1].plot(temp[40:200,5])
axs2[1].plot(temp_hat[40:200,5])
axs2[1].legend(['Original', 'compressed'])
plt.show()

# compute SNDR
SNDR = 0
for i in range(temp.shape[1]):
    P_d = np.linalg.norm(temp[:,i])
    P_error = np.linalg.norm(temp[:,i] - temp_hat[:,i])
```

```
    SNDR += 1/(temp.shape[1]) * 20*np.log10(P_d/P_error)
print(f'Mean SNDR across channels: {SNDR:.2f} dB')
```




```
Mean SNDR across channels: 17.94 dB
```

APPENDIX

H

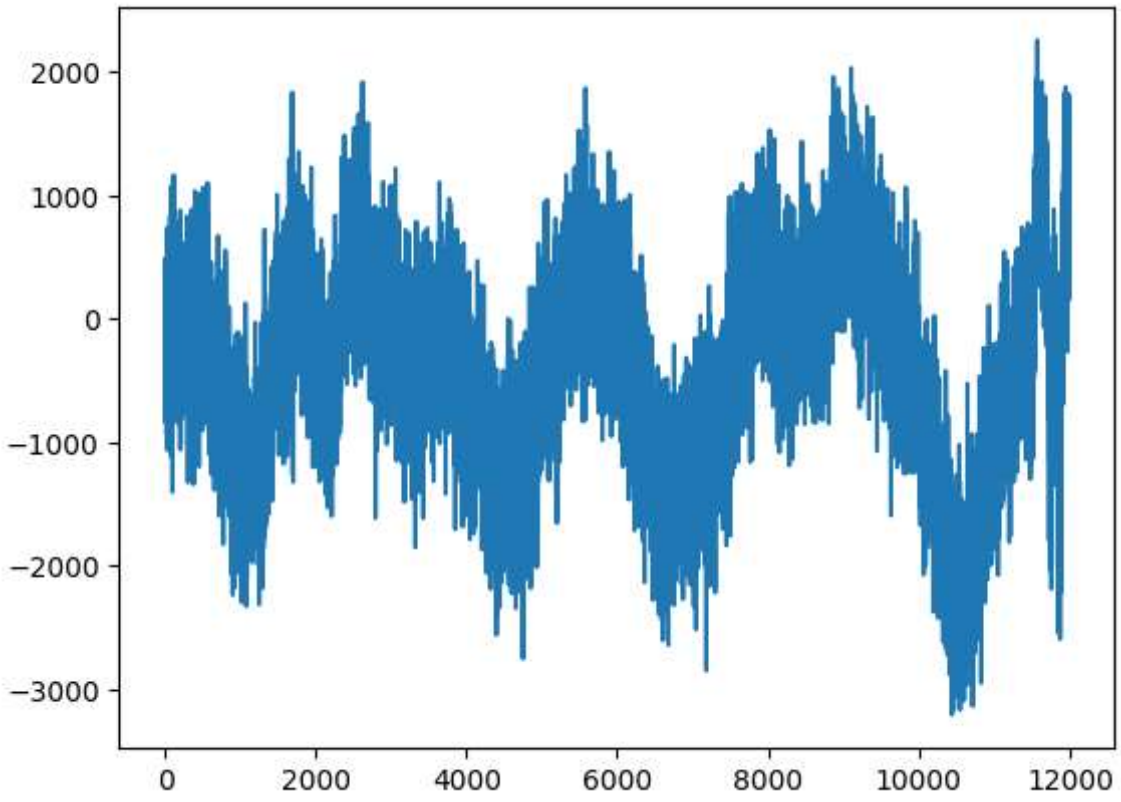
EFFECTS OF DPCM, MS STEREO, MS QUADRO, CHANNEL MEAN REMOVAL, RICE CODING, AND HUFFMAN CODING

Here the methods: DPCM, MS Stereo, MS Quadro, Channel Mean removal, Rice Coding, and Huffman coding will be illustrated

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import heapq
from collections import defaultdict

s = np.load('../test_data_sintef.npy')

# calculate min num bits we need to allocate to the original
bO = len(bin(np.max(np.abs(s)).astype(int))[2:])

# Define signal
x = s[:12000,:].astype(int)

# Rice coding parameter
K = 9
```

In [ ]:
```
plt.plot(x[:,0])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x20f7c49f150>]
```



DPCM

In [ ]:
```
def dpcm(x):
    previousSample = 0 # initial predicted value

    residual = np.zeros_like(x) # residual signal

    for i in range(len(x)):
        residual[i] = x[i] - previousSample # residual is true - predict

        previousSample = x[i] # reconstruct by adding residual to past sample

    return  residual

# inverse DPCM
def dpcm_i(x):
    previousSample = 0 # initial predicted value

    predict = np.zeros_like(x) # reconstructed signal

    for i in range(len(x)):

        previousSample = previousSample + x[i] # reconstruct by adding residual

        predict[i] = previousSample # predicted value is set to be the past samp
    return predict
```

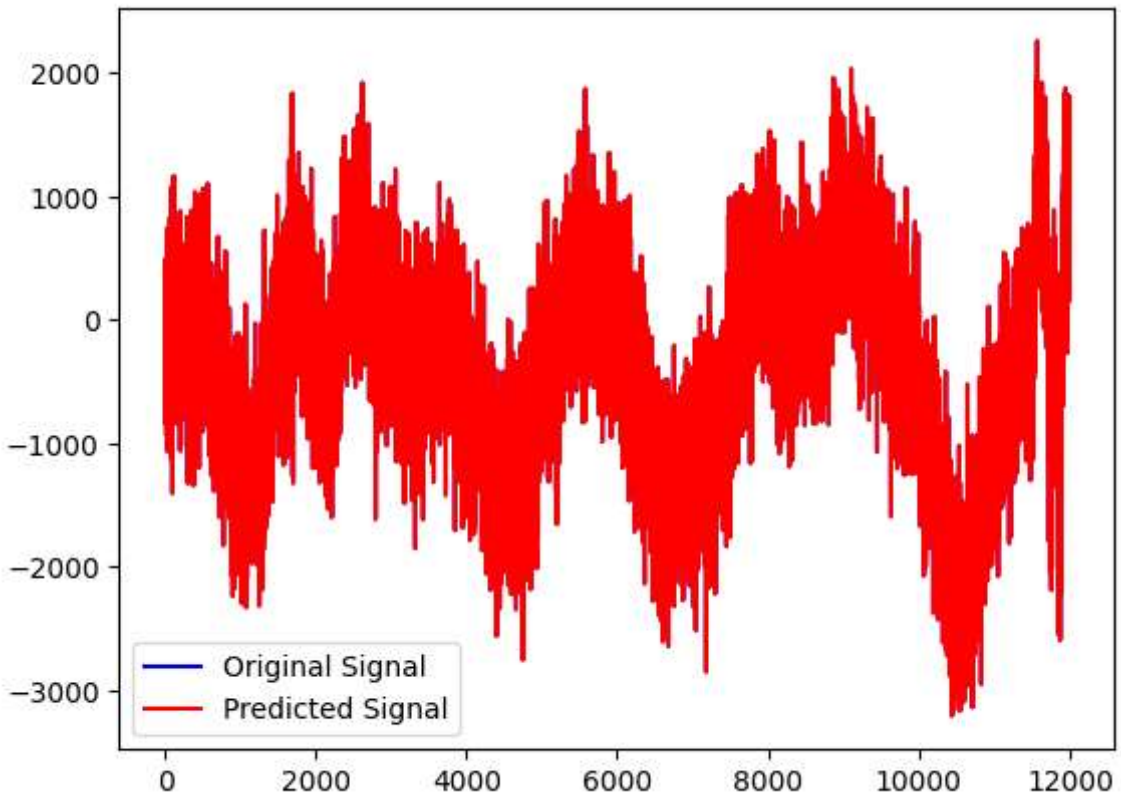
Perform DPCM

In [ ]:
```
#perform DPCM
predictedSignal = []
residualSignal = []
for i in range(x.shape[1]): # loop through all channels
    res = dpcm(x[:,i])
    pred = dpcm_i(res)
    predictedSignal.append(pred)
    residualSignal.append(res)
```

Test the results

In [ ]:
```
#Test one channel:
# Plot the original and predicted signals
k = 0
plt.plot(x[:,k], 'b')
plt.plot(predictedSignal[k], 'r')
plt.legend(['Original Signal', 'Predicted Signal'])

i = 6 #we check signal num 6 just for fun
print('quantized value that is transmitted:')
print(bin(int(residualSignal[k][i])))

predict = residualSignal[k][i] + predictedSignal[k][i-1]
print('value obtained from adding this to last signal:')
print(bin(int(predict)))

print('original signal:')
print(bin(int(x[i,k])))
```

```
quantized value that is transmitted:
0b10110
value obtained from adding this to last signal:
-0b101110100
original signal:
-0b101110100
```



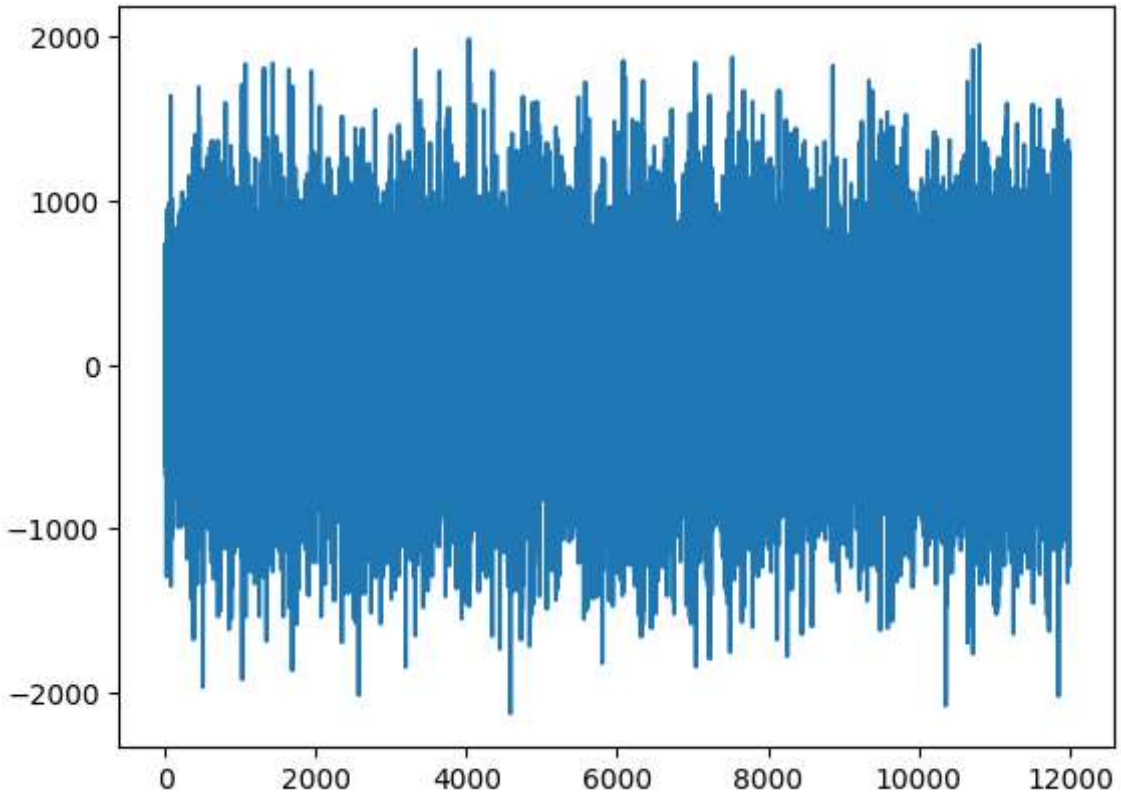
In [ ]:
```python
# calculate min num bits we need to allocate
bR = len(bin(np.max(np.abs(np.array(residualSignal))).astype(int))[2:])
error = np.sum(np.abs(x - np.array(predictedSignal).T))
print(f'Bits used on compressed signal: {bR}')
print(f'Error: {error:.2f}')
```

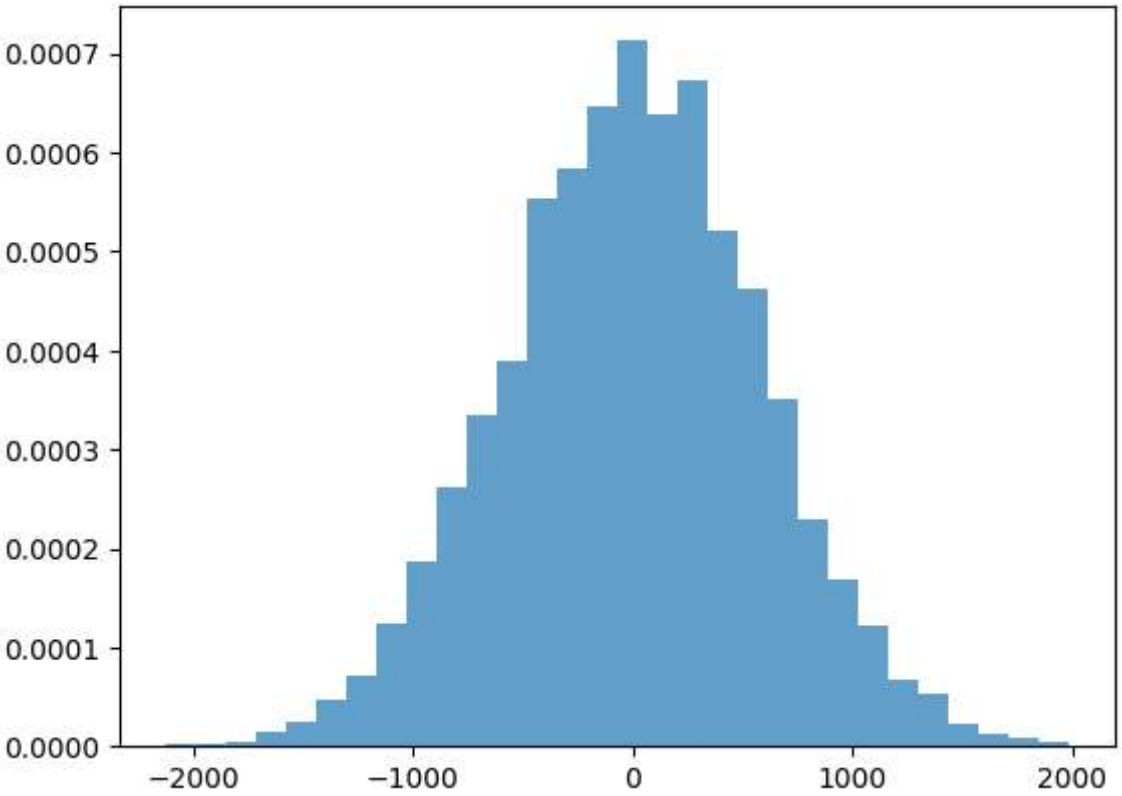
```
Bits used on compressed signal: 13
Error: 0.00
```

Plot the residual signal and its distribution to make sure that it looks like a Laplacian dist, so that it is resonable to use Rice coding

In [ ]:
```python
# check for channel 1
plt.figure(1)
plt.plot(residualSignal[0])
plt.figure(2)
plt.hist(residualSignal[0], bins=30, density=True, alpha=0.7)
```

Out[ ]:
```
(array([2.43072436e-06, 1.82304327e-06, 4.86144871e-06, 1.45843461e-05,
        2.55226057e-05, 4.61837628e-05, 7.10986874e-05, 1.23359261e-04,
        1.86558094e-04, 2.61910549e-04, 3.35439961e-04, 3.89523578e-04,
        5.52989791e-04, 5.82766164e-04, 6.46572679e-04, 7.12809917e-04,
        6.39280506e-04, 6.72095284e-04, 5.20175012e-04, 4.61837628e-04,
        3.51847351e-04, 2.29703452e-04, 1.67719981e-04, 1.22751580e-04,
        6.74526009e-05, 5.22605737e-05, 2.24842003e-05, 1.33689840e-05,
        9.11521633e-06, 3.64608653e-06]),
 array([-2128.        , -1990.86666667, -1853.73333333, -1716.6       ,
        -1579.46666667, -1442.33333333, -1305.2       , -1168.06666667,
        -1030.93333333,  -893.8       ,  -756.66666667,  -619.53333333,
         -482.4       ,  -345.26666667,  -208.13333333,   -71.        ,
           66.13333333,   203.26666667,   340.4       ,   477.53333333,
          614.66666667,   751.8       ,   888.93333333,  1026.06666667,
         1163.2       ,  1300.33333333,  1437.46666667,  1574.6       ,
         1711.73333333,  1848.86666667,  1986.        ]),
 <BarContainer object of 30 artists>)
```





MS stereo coding (Sum/Difference coding):

In [ ]:

```
def stereo(temp):
    channelPairs = [[0,1],[2,3],[4,5],[6,7],[8,9], [10,11], [12,13], [14,15]] #

    S = np.zeros([temp.shape[0],int(temp.shape[1]/2)])
    D = np.zeros([temp.shape[0],int(temp.shape[1]/2)])

    k = 0
    for i in channelPairs:
        # encode
        S[:,k] = 0.5*(temp[:,i[0]] + temp[:,i[1]]) # take avg. of all samples of
        D[:,k] = 0.5*(temp[:,i[0]] - temp[:,i[1]]) # take diff. of all samples o

        k += 1
    return D, S

def stereo_i(temp):
    channelPairs = [[0,1],[2,3],[4,5],[6,7],[8,9], [10,11], [12,13], [14,15]] #

    S = np.zeros([temp.shape[0],int(temp.shape[1]/2)])
    D = np.zeros([temp.shape[0],int(temp.shape[1]/2)])
    predict = np.zeros_like(temp)

    D, S = np.split(temp, 2, axis=1)

    k = 0
    for i in channelPairs:

        # decode
        predict[:,i[0]] = S[:,k] + D[:,k] # reconstruct original channels
```

```
        predict[:,i[1]] = S[:,k] - D[:,k]

        k += 1
    return predict
```

In [ ]:
```
# conpute the SumDiff with the given pairs
D, S = stereo(x)
DS = np.concatenate((D,S),1)

predict = stereo_i(DS)
```

Testing the results:

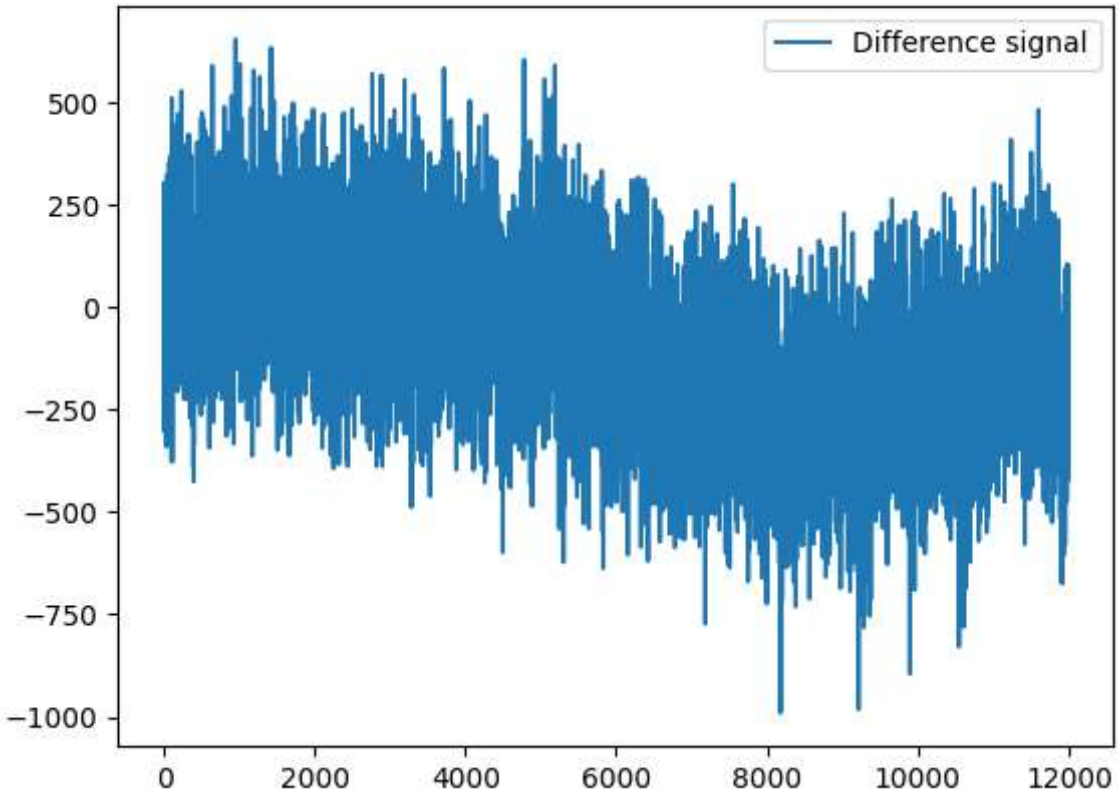
In [ ]:
```
# calculate min num bits we need to allocate
bDS = len(bin(np.max(np.abs(DS)).astype(int))[2:])
error = np.sum(np.abs(x - predict))
print(f'Bits used on compressed signal: {bDS}')
print(f'Error: {error:.2f}')
```

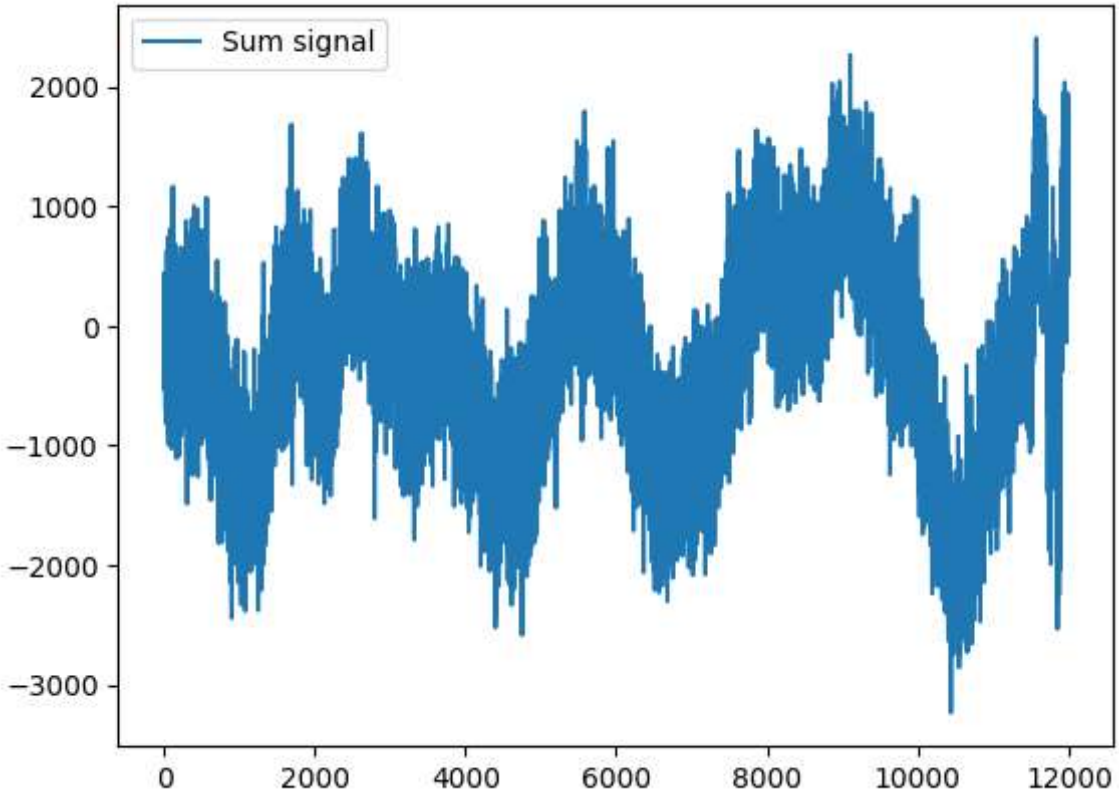
```
Bits used on compressed signal: 13
Error: 0.00
```

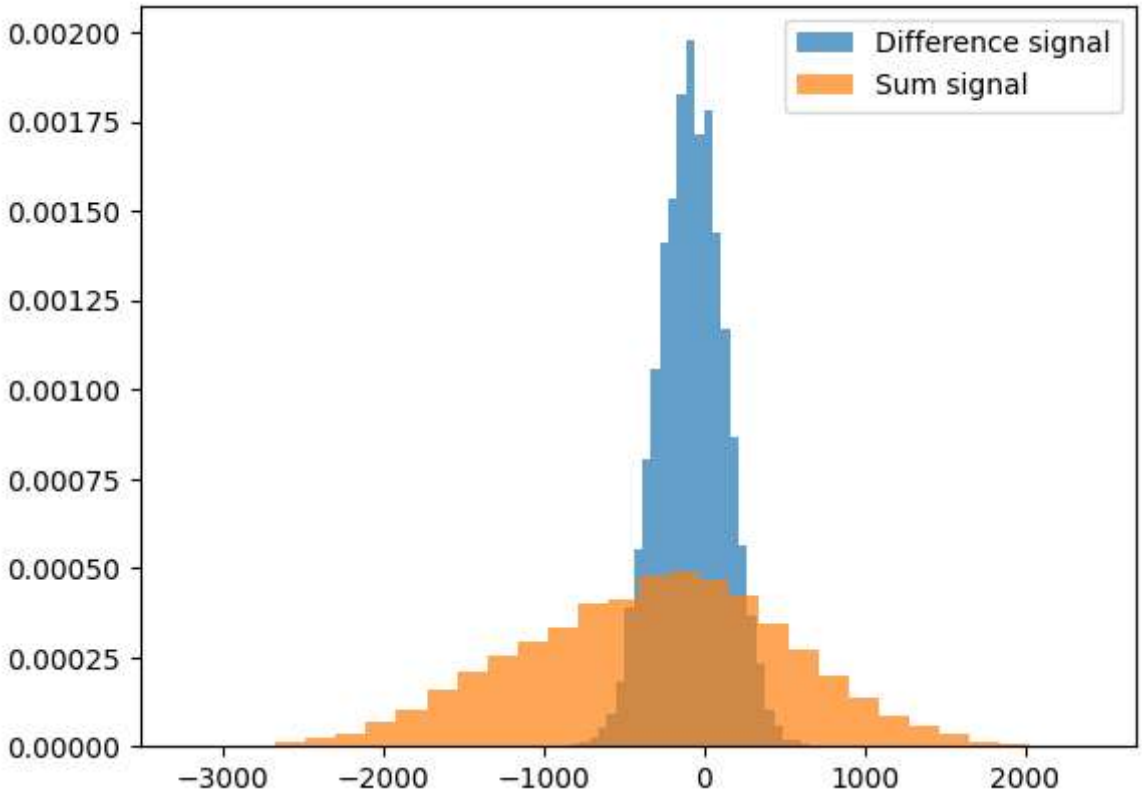
Check that the result has the right distribution to be used in Rice coding

In [ ]:
```
plt.figure(1)
plt.plot(D[:,0])
plt.legend(['Difference signal'])

plt.figure(2)
plt.plot(S[:,0])
plt.legend(['Sum signal'])

plt.figure(3)
plt.hist(D[:,0], bins=30, density=True, alpha=0.7)
plt.hist(S[:,0], bins=30, density=True, alpha=0.7)
plt.legend(['Difference signal', 'Sum signal'])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x20f00085690>
```


Difference signal
500
250
0
−250
−500
−750
−1000
0
2000
4000
6000
8000
10000
12000


Sum signal
2000
1000
0
−1000
−2000
−3000
0
2000
4000
6000
8000
10000
12000



MS Quadro

In [ ]:

```
def quadro(temp):
    channelQuartets = [[0,1,2,3],[4,5,6,7],[8,9,10,11]] # compare the channels c
    channelPairs = [[12,13], [14,15]] # compare the channels closes to each othe

    # initialize
    Q1 = np.zeros([temp.shape[0],3])
    Q2 = np.zeros([temp.shape[0],3])
    Q3 = np.zeros([temp.shape[0],3])
    Q4 = np.zeros([temp.shape[0],3])

    S = np.zeros([temp.shape[0],2])
    D = np.zeros([temp.shape[0],2])

    k = 0 # counting channels in one quartet
    for i in channelQuartets:
        # encode
        Q1[:,k] = 0.25*(temp[:,i[0]] + temp[:,i[1]] + temp[:,i[2]] + temp[:,i[3]
        Q2[:,k] = 0.25*(temp[:,i[0]] - temp[:,i[1]] - temp[:,i[2]] + temp[:,i[3]
        Q3[:,k] = 0.25*(temp[:,i[0]] - temp[:,i[1]] + temp[:,i[2]] - temp[:,i[3]
        Q4[:,k] = 0.25*(temp[:,i[0]] + temp[:,i[1]] - temp[:,i[2]] - temp[:,i[3]

        k += 1

    k = 0 # counting channles in one duo
    for i in channelPairs:
        # encode
        S[:,k] = 0.5*(temp[:,i[0]] + temp[:,i[1]])
        D[:,k] = 0.5*(temp[:,i[0]] - temp[:,i[1]])

        k += 1
```

```
    Q = np.concatenate((Q1,Q2,Q3,Q4,D,S),1)

    return Q

def quadro_i(temp):
    channelQuartets = [[0,1,2,3],[4,5,6,7],[8,9,10,11]] # compare the channels c
    channelPairs = [[12,13], [14,15]] # compare the channels closes to each othe

    predict = np.zeros_like(temp)

    split1, split2, split3, split4 = np.split(temp, 4, axis=1)
    split5 = np.concatenate((split1, split2, split3), axis=1)
    Q1,Q2,Q3,Q4 = np.split(split5, 4, axis=1)
    D,S = np.split(split4, 2, axis=1)

    k = 0 # counting channels in one quartet
    for i in channelQuartets:

        # decode
        predict[:,i[0]] = (Q1[:,k] + Q2[:,k] + Q3[:,k] + Q4[:,k])
        predict[:,i[1]] = (Q1[:,k] - Q2[:,k] - Q3[:,k] + Q4[:,k])
        predict[:,i[2]] = (Q1[:,k] - Q2[:,k] + Q3[:,k] - Q4[:,k])
        predict[:,i[3]] = (Q1[:,k] + Q2[:,k] - Q3[:,k] - Q4[:,k])

        k += 1

    k = 0 # counting channles in one duo
    for i in channelPairs:

        # decode
        predict[:,i[0]] = (S[:,k] + D[:,k])
        predict[:,i[1]] = (S[:,k] - D[:,k])

        k += 1

    return predict
```

In [ ]:
```
# perform compression
Q = quadro(x)

predict = quadro_i(Q)
```

Testing the results:

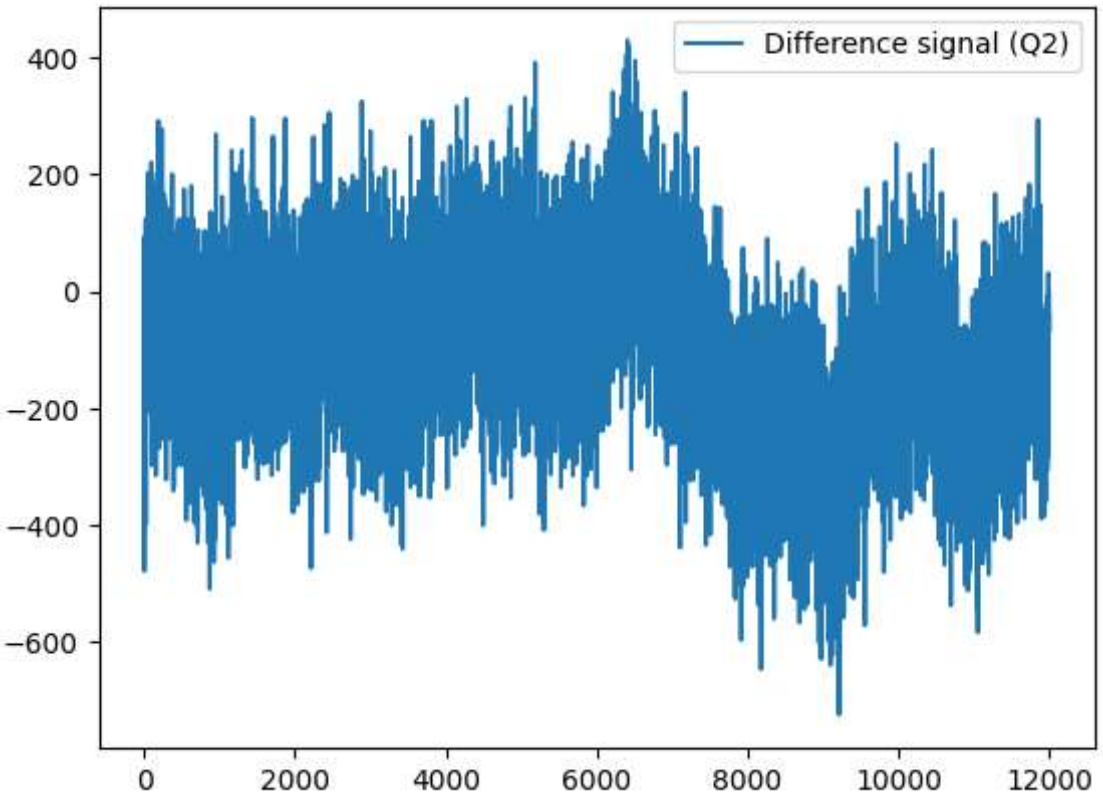
In [ ]:
```
# calculate min num bits we need to allocate
bQ = len(bin(np.max(np.abs(Q)).astype(int))[2:])
error = np.sum(np.abs(x - predict))
print(f'Bits used on compressed signal: {bQ}')
print(f'Error: {error:.2f}')
```

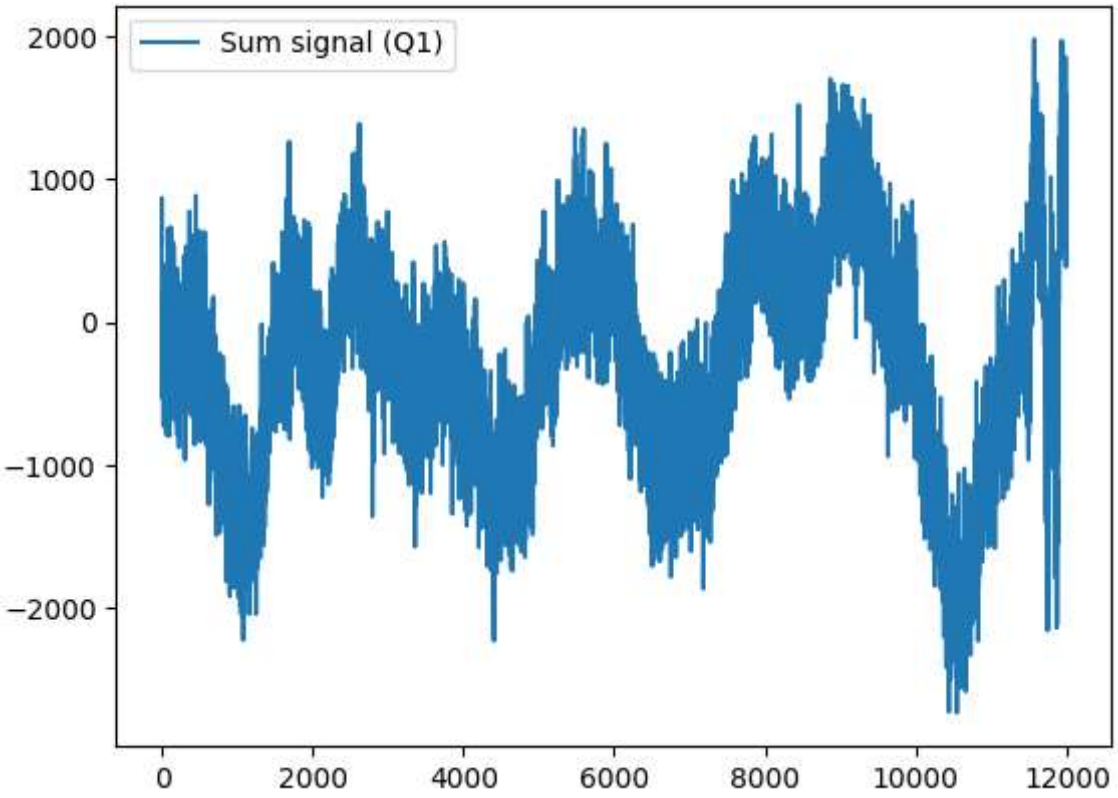
```
Bits used on compressed signal: 13
Error: 0.00
```

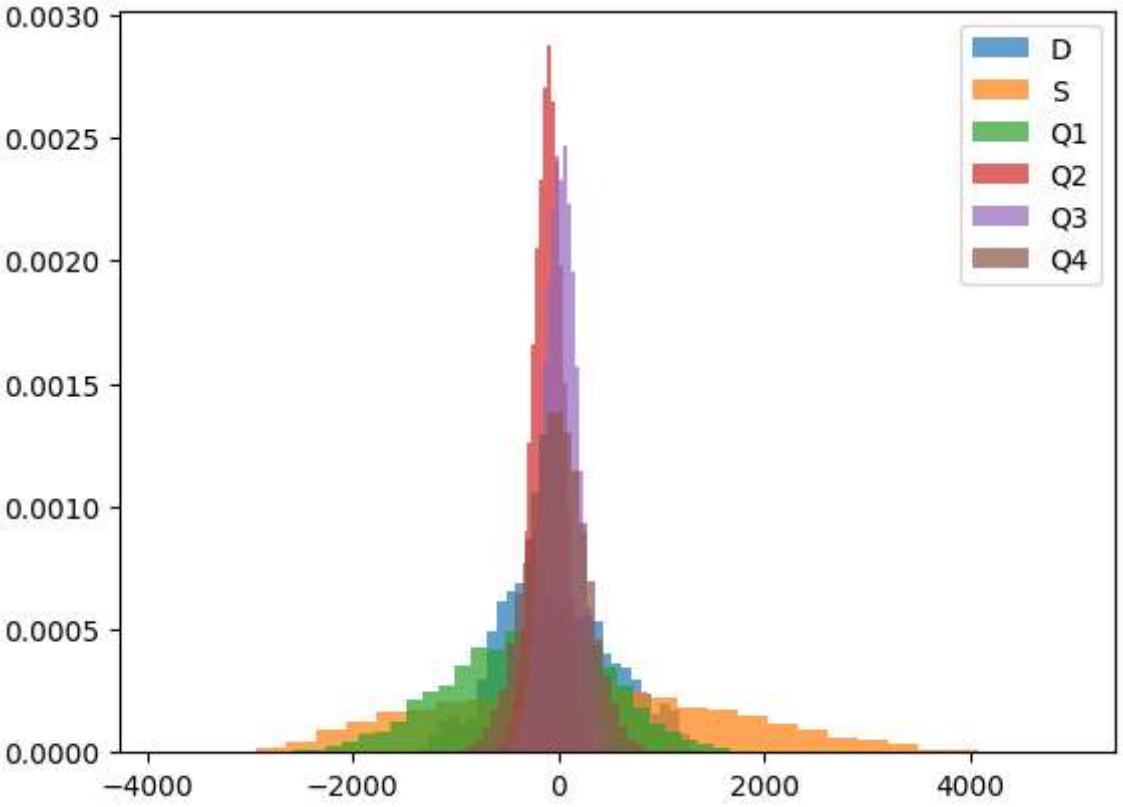
Check that the result has the right distribution to be used in Rice coding

In [ ]:
```
split1, split2, split3, split4 = np.split(Q, 4, axis=1)
split5 = np.concatenate((split1, split2, split3), axis=1)
Q1,Q2,Q3,Q4 = np.split(split5, 4, axis=1)
D,S = np.split(split4, 2, axis=1)

plt.figure(1)
plt.plot(Q2[:,0])
plt.legend(['Difference signal (Q2)'])

plt.figure(2)
plt.plot(Q1[:,0])
plt.legend(['Sum signal (Q1)'])

plt.figure(3)
plt.hist(D[:,0], bins=30, density=True, alpha=0.7)
plt.hist(S[:,0], bins=30, density=True, alpha=0.7)
plt.hist(Q1[:,0], bins=30, density=True, alpha=0.7)
plt.hist(Q2[:,0], bins=30, density=True, alpha=0.7)
plt.hist(Q3[:,0], bins=30, density=True, alpha=0.7)
plt.hist(Q4[:,0], bins=30, density=True, alpha=0.7)
plt.legend(['D', 'S', 'Q1', 'Q2', 'Q3', 'Q4'])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x20f066bce10>
```







Channel mean removal

In [ ]:
```
def chmean(temp):
    output = np.zeros((temp.shape[0], temp.shape[1] + 1))
    for i in range(temp.shape[1]):
        output[:,i] = temp[:,i] - np.mean(temp, axis=1)
    output[:,-1] = np.mean(temp, axis=1)
```

```
    return output

def chmean_i(temp):
    output = np.zeros((temp.shape[0], temp.shape[1]-1))
    for i in range(temp.shape[1]-1):
        output[:,i] = temp[:,i] + temp[:,-1]

    return output
```

In [ ]:
```
CH = chmean(x)

predict = chmean_i(CH)
```

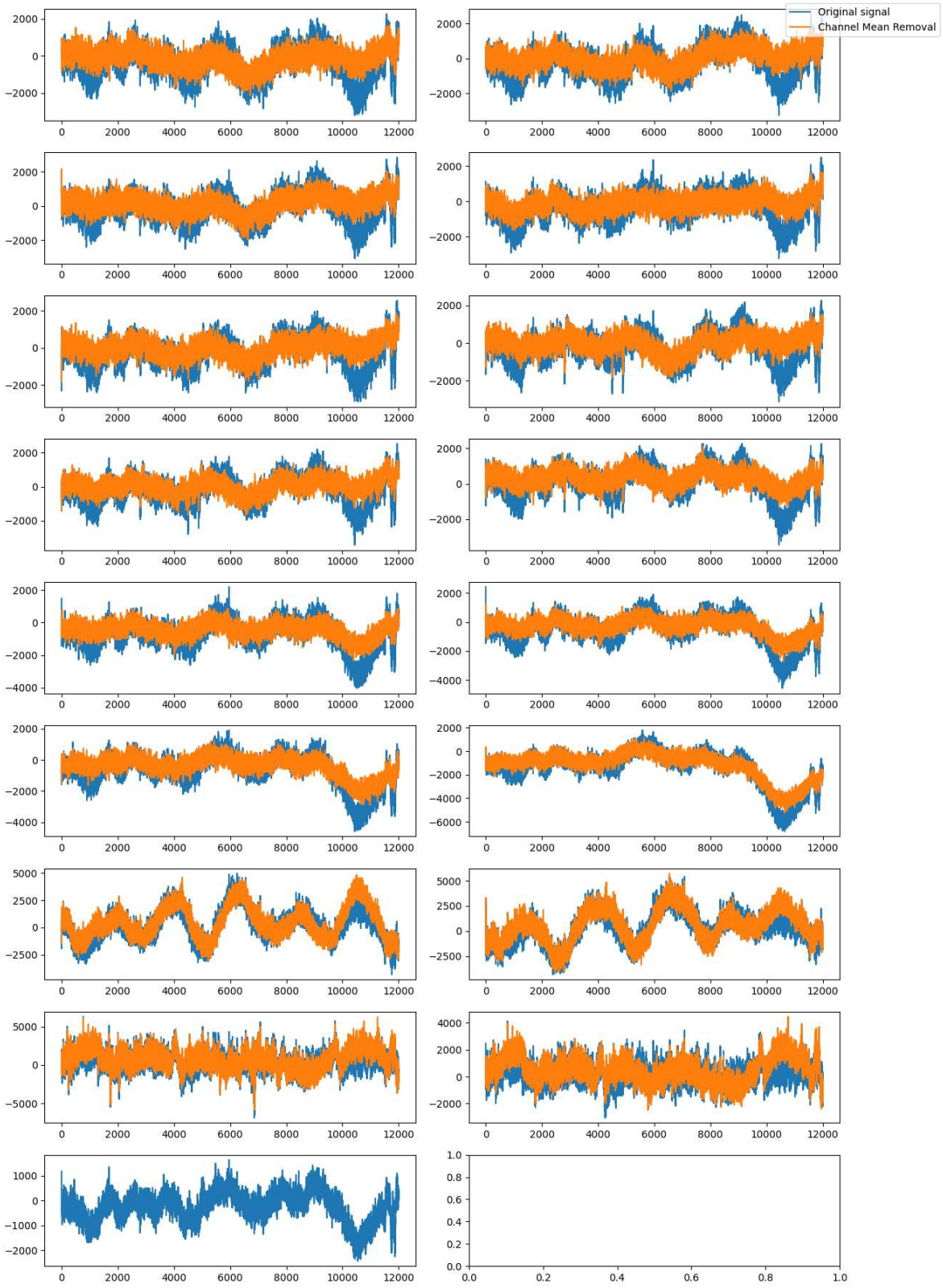
Testing the results:

In [ ]:
```
# calculate min num bits we need to allocate
bCH = len(bin(np.max(np.abs(CH)).astype(int))[2:])
error = np.sum(np.abs(x - predict))
print(f'Bits used on compressed signal: {bCH}')
print(f'Error: {error:.2f}')
```

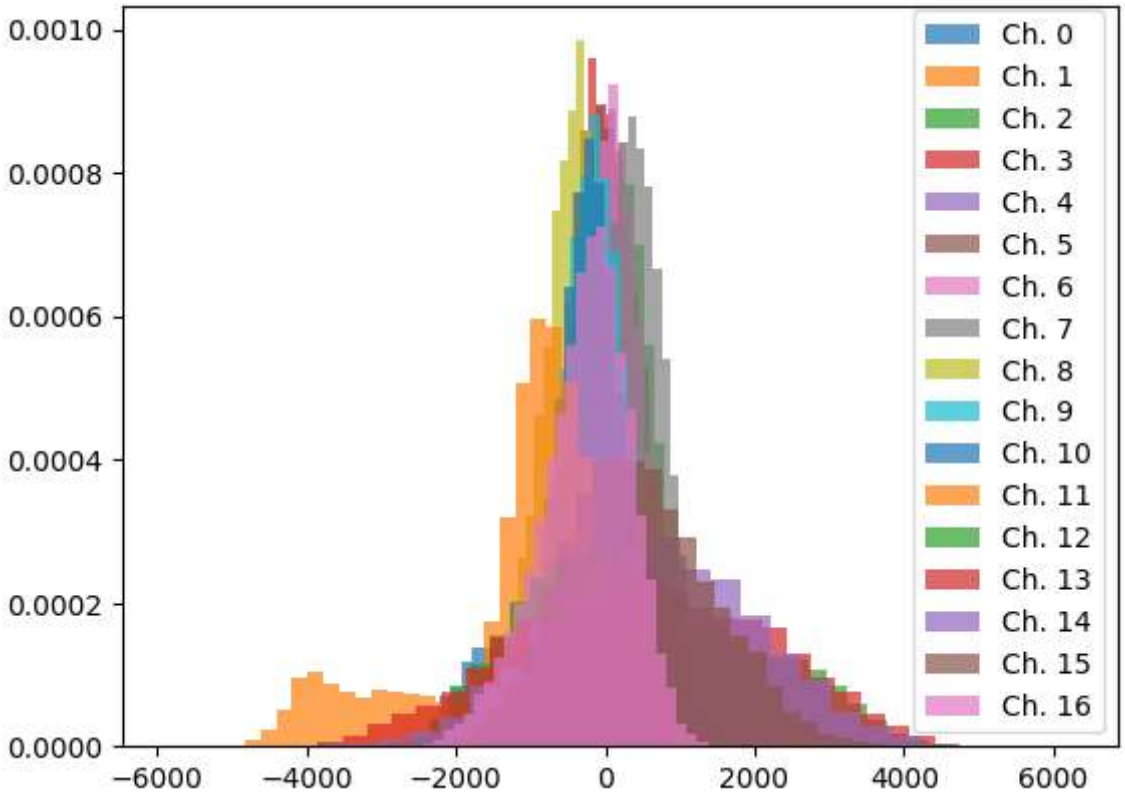
```
Bits used on compressed signal: 13
Error: 0.00
```

Check that the result has the right distribution to be used in Rice coding

In [ ]:
```
# Plot the channels with mean removed, ch. 17 is the mean channel
fig, axs = plt.subplots(9, 2, figsize=(12, 18))
for i in range(CH.shape[1]):
    if i != 16:
        axs.flatten()[i].plot(x[:,i])
    axs.flatten()[i].plot(CH[:,i])
fig.legend(['Original signal', 'Channel Mean Removal'], loc='upper right', bbox_
plt.tight_layout()
plt.show()

leg = []
plt.figure()
for i in range(CH.shape[1]):
    plt.hist(CH[:,i], bins=30, density=True, alpha=0.7)
    leg.append('Ch. ' + str(i))
plt.legend(leg)
```



Out[ ]: <matplotlib.legend.Legend at 0x20f07db5650>



Rice Coding

Define the functions and compute Rice

In [ ]:

```
def rice_encode_array_sign(values, K, alpha = 0.5):
    encoded_values = []
    window = 40

    # look at window values at a time, update the divisor, and use this to perfo
    for i in range(int(len(values)/window)):
        if (i>0):
            if (np.log(2)*np.mean(np.abs(values[(i-1)*window:(i-1)*window + wind
                K = 0 # since log2(0) is not possible
            else:
                K = np.round( alpha*np.log2(np.log(2)*np.mean(np.abs(values[(i-1
                if(K<0):
                    K = 0
        K = int(K)

        divisor = int(2**K) # can be impelented by right shifting

        encoded_values.append(bin(K)[2:].zfill(4)) # send the K-value that is us

        for value in values[i*window:i*window + window]:
            sign_bit = "1" if value < 0 else "0"
            abs_value = abs(value)

            quotient = abs_value // divisor
            remainder = abs_value % divisor

            unary_code = "1" * quotient + "0"
```

```
            binary_code = bin(remainder)[2:].zfill(divisor.bit_length() - 1)

            encoded_values.append(sign_bit + unary_code + binary_code) # since t

    return encoded_values

def rice_decode_array_sign(encoded_values):
    decoded_values = []
    window = 40
    counter = 0

    for encoded_value in encoded_values:
        if counter == 0:
            K = int(encoded_value, 2)
            divisor = int(2**K)
            counter = window
        else:
            counter -= 1

            sign_bit = encoded_value[0]
            quotient = encoded_value[1:].index("0") #+ 1
            remainder = int(encoded_value[quotient + 1:], 2)

            value = quotient * divisor + remainder
            if sign_bit == "1":
                value = -value

            decoded_values.append(value)

    return decoded_values
```

In [ ]:

```
# here the sign bit is implemented as described here: https://en.wikipedia.org/w
# from testing the two ways of implementing the sign bit give the same results
def rice_encode(values, K, alpha = 0.5):
    encoded_values = []
    window = 40

    # look at window values at a time, update the divisor, and use this to perfo
    for i in range(int(len(values)/window)):
        if (i>0):
            if (np.log(2)*np.mean(np.abs(values[(i-1)*window:(i-1)*window + wind
                K = 0 # since log2(0) is not possible
            else:
                K = np.round( alpha*np.log2(np.log(2)*np.mean(np.abs(values[(i-1
                if(K<0):
                    K = 0
        K = int(K)

        divisor = int(2**K) # can be impelented by right shifting

        encoded_values.append(bin(K)[2:].zfill(4)) # send the K-value that is us

        for value in values[i*window:i*window + window]:

            # make sure that positive and negative numbers get unique values
            if value >= 0:
                abs_value = abs(value)*2
            else:
                abs_value = abs(value)*2 - 1
```

```
            quotient = abs_value // divisor
            remainder = abs_value % divisor

            unary_code = "1" * quotient + "0"
            binary_code = bin(remainder)[2:].zfill(divisor.bit_length() - 1)

            encoded_values.append(unary_code + binary_code)

    return encoded_values

def rice_decode(encoded_values):
    decoded_values = []
    window = 40
    counter = 0

    for encoded_value in encoded_values:
        if counter == 0:
            K = int(encoded_value, 2)
            divisor = int(2**K)
            counter = window
        else:
            counter -= 1

            quotient = encoded_value[:].index("0")
            remainder = int(encoded_value[quotient:], 2)

            value = quotient * divisor + remainder

            # determine if the value was positive or negative
            if value % 2 == 0:
                value = value // 2
            else:
                value = - (value+1) // 2

            decoded_values.append(value)

    return decoded_values
```

In [ ]:
```
# calculate best k (to get the best divisor m):
np.log2(np.log(2)*np.mean(np.abs(x)))
```

Out[ ]: 9.226195137102788

In [ ]:
```
encoded = []
encoded_alt_sign = []
decoded = np.zeros_like(x)

for i in range(x.shape[1]):
    print('Channel: ' + str(i)) # print channel to keep track of progress
    encoded_alt_sign.append(           rice_encode(x[:,i], K))
    encoded.append(         rice_encode_array_sign(x[:,i], K))
    decoded[:,i] = np.array(rice_decode_array_sign(encoded[i]))
```

```
Channel: 0
Channel: 1
Channel: 2
Channel: 3
Channel: 4
Channel: 5
Channel: 6
Channel: 7
Channel: 8
Channel: 9
Channel: 10
Channel: 11
Channel: 12
Channel: 13
Channel: 14
Channel: 15
```

Results Rice:

In [ ]:

```
error = 0
lenOriginal = 0
lenRice = 0
lenRice_alt_sign = 0


for j in range(x.shape[1]):
    error += np.sum(np.abs(decoded[:,j] - x[:,j]))
    for i in range(x.shape[0]):
        lenOriginal += bO #
        lenRice += len(encoded[j][i])
        lenRice_alt_sign += len(encoded_alt_sign[j][i])




print(f'Total compression error: {error:.2f}')
print(f'Length of original signal: {lenOriginal}')
print(f'Length of encoded signal: {lenRice}')
print(f'Length of encoded signal using alternative sign method: {lenRice_alt_sig


RCR2 = (1 - lenRice_alt_sign/lenOriginal) * 100
RCR = (1 - lenRice/lenOriginal) * 100
CR = lenOriginal/lenRice
print(f'Compression ratio (RCR) of Rice: {RCR:.2f}%')
print(f'Compression ratio (RCR) of Rice (with alternative sign method): {RCR2:.2
print(f'CR: {CR:.2f}')


print('\n')
print('--------------------Looking at a single sample--------------------')

k = 0
i = 12 #we check signal num 6 just for fun

print('Original sample:')
x_bin = bin(int(x[i,k]))
if x_bin[0] == '-':
```

```
    x_bin = x_bin[3:].zfill(b0)
    x_bin = '1' + x_bin
else:
    x_bin = x_bin[2:].zfill(b0)
    x_bin = '0' + x_bin
print(x_bin)


print('Quantized value that is transmitted:')
print(encoded[k][i])


print('Decoded value:')
decoded_bin = bin(int(decoded[i,k]))
if decoded_bin[0] == '-':
    decoded_bin = decoded_bin[3:].zfill(b0)
    decoded_bin = '1' + decoded_bin
else:
    decoded_bin = decoded_bin[2:].zfill(b0)
    decoded_bin = '0' + decoded_bin
print(decoded_bin)
```

```
Total compression error: 0.00
Length of original signal: 3072000
Length of encoded signal: 2263829
Length of encoded signal using alternative sign method: 2263538
Compression ratio (RCR) of Rice: 26.31%
Compression ratio (RCR) of Rice (with alternative sign method): 26.32%
CR: 1.36


--------------------Looking at a single sample--------------------
Original sample:
0000000100011110
Quantized value that is transmitted:
10111001000
Decoded value:
0000000100011110
```

We see that the difference between the two sign methods is negligable

Huffman

Here the same dataset is used to bulid the huffman tree as the one that is being processed, which is an unrealist case

In [ ]:
```
# written with the help of GPT4
class HuffmanNode:
    def __init__(self, value=None, frequency=0):
        self.value = value
        self.frequency = frequency
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.frequency < other.frequency
```

```
def build_frequency_table(data):
    frequency_table = defaultdict(int)
    for num in data:
        frequency_table[num] += 1
    return frequency_table

def build_huffman_tree(frequency_table):
    priority_queue = [HuffmanNode(value, freq) for value, freq in frequency_tabl
    heapq.heapify(priority_queue)

    while len(priority_queue) > 1:
        left_node = heapq.heappop(priority_queue)
        right_node = heapq.heappop(priority_queue)

        parent_node = HuffmanNode(frequency=left_node.frequency + right_node.fre
        parent_node.left = left_node
        parent_node.right = right_node

        heapq.heappush(priority_queue, parent_node)

    return heapq.heappop(priority_queue)

def build_encoding_table(huffman_tree):
    encoding_table = {}
    def traverse(node, code):
        if node.value is not None:
            encoding_table[node.value] = code
        else:
            traverse(node.left, code + "0")
            traverse(node.right, code + "1")
    traverse(huffman_tree, "")
    return encoding_table

def huffman_encode(data):
    frequency_table = build_frequency_table(data)
    huffman_tree = build_huffman_tree(frequency_table)
    encoding_table = build_encoding_table(huffman_tree)

    encoded_data = ""
    for num in data:
        encoded_data += encoding_table[num]

    return encoded_data, huffman_tree

def huffman_decode(encoded_data, huffman_tree):
    decoded_data = []
    current_node = huffman_tree

    for bit in encoded_data:
        if bit == "0":
            current_node = current_node.left
        else:
            current_node = current_node.right

        if current_node.value is not None:
            decoded_data.append(current_node.value)
            current_node = huffman_tree

    return decoded_data
```

In [ ]:
```
if x.ndim == 1:
    encoded, huffman_tree = huffman_encode(x)
    decoded = np.array(huffman_decode(encoded, huffman_tree))
else:
    encoded = []
    decoded = np.zeros_like(x)

    for i in range(x.shape[1]):
        print('Channel: ' + str(i))
        encoded_data, huffman_tree = huffman_encode(x[:,i])
        encoded.append(encoded_data)
        decoded[:,i] = np.array(huffman_decode(encoded_data, huffman_tree))
```

```
Channel: 0
Channel: 1
Channel: 2
Channel: 3
Channel: 4
Channel: 5
Channel: 6
Channel: 7
Channel: 8
Channel: 9
Channel: 10
Channel: 11
Channel: 12
Channel: 13
Channel: 14
Channel: 15
```

Huff Result:

In [ ]:
```
error = 0
lenOriginal = 0
lenHuff = 0

# check if we use one or several channels
if x.ndim == 1:
    error = np.sum(np.abs(decoded - x))
    lenHuff = len(encoded)
    for i in range(x.shape[0]):
        lenOriginal += bO
else:
    for j in range(x.shape[1]):
        error += np.sum(np.abs(decoded[:,j] - x[:,j]))
        lenHuff += len(encoded[j])
        for i in range(x.shape[0]):
            lenOriginal += bO  # 16



#lenHuff = len(encoded)
print(f'Total compression error: {error:.2f}')
print(f'Length of original signal: {lenOriginal}')
print(f'Length of encoded signal: {lenHuff}')
```

```
RCR = (1 - lenHuff/lenOriginal) * 100
print(f'Compression ratio of Huffman: {RCR:.2f}%')
```

```
Total compression error: 0.00
Length of original signal: 3072000
Length of encoded signal: 2067482
Compression ratio of Huffman: 32.70%
```

APPENDIX

I

FREQUENCY PLOTS OF DATASETS

Dataset 1

Plots of the frequency power spectrum of dataset 1 with the y-axis in logarithmic scale are shown in figure I.0.1 and I.0.2.

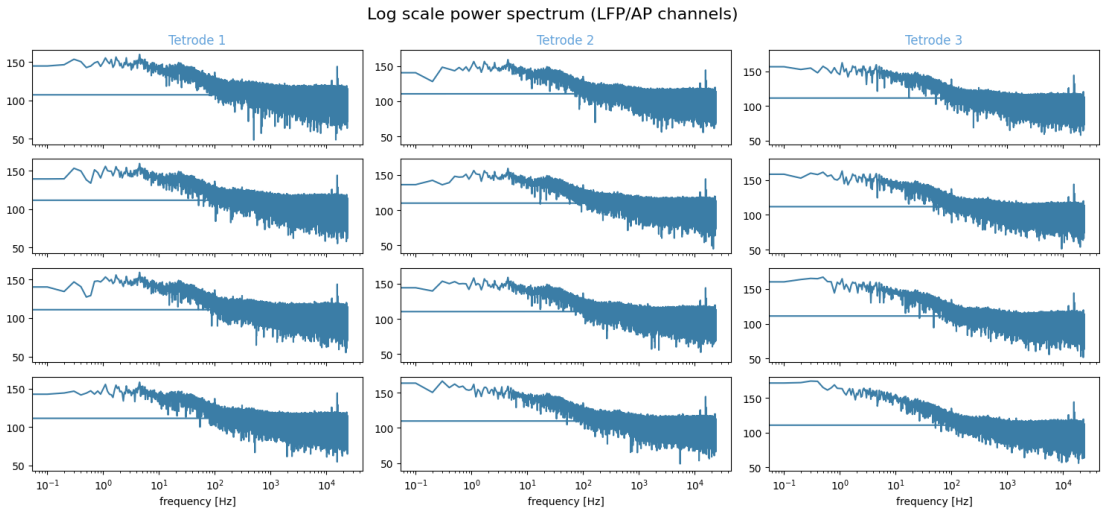


Figure I.0.1: Power spectrum of LFP/EAP channels of dataset 1.

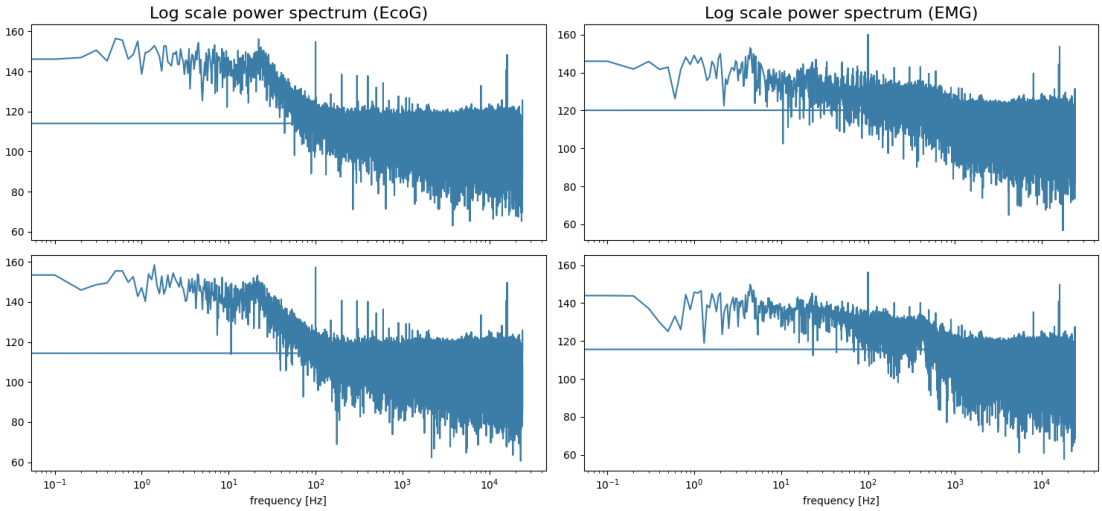


Figure I.0.2: Power spectrum of EcoG and EMG channels of dataset 1.

Dataset 2

Figure I.0.3 and I.0.4 show the frequency power spectrum of dataset 2. Here the y-axis indicates the magnitude of the power in the different frequencies and is not in logarithmic scale. Plots of the frequency power spectrum with the y-axis in logarithmic scale are shown in figure I.0.5 and I.0.6.

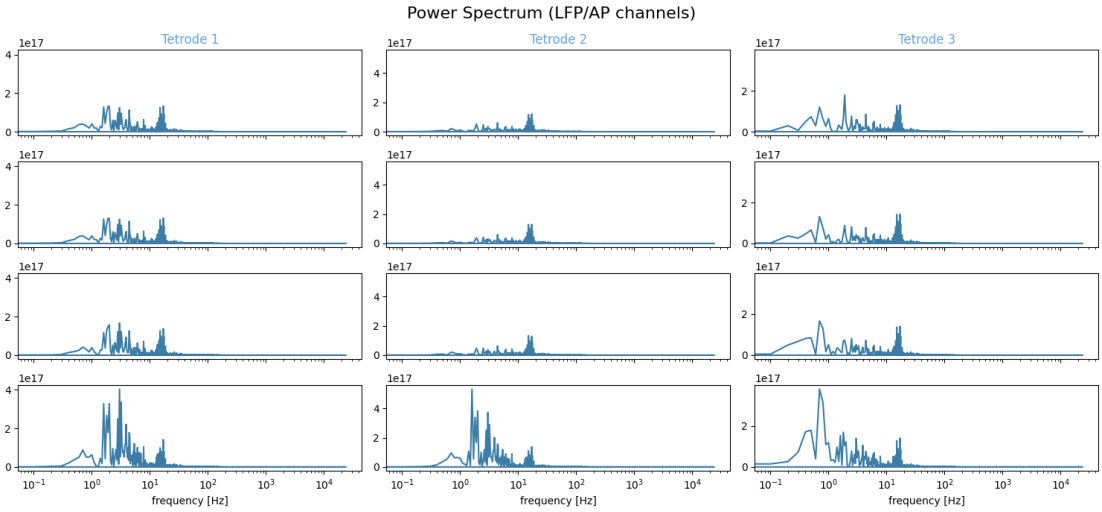


Figure I.0.3: Power spectrum of LFP/EAP channels of dataset 2.

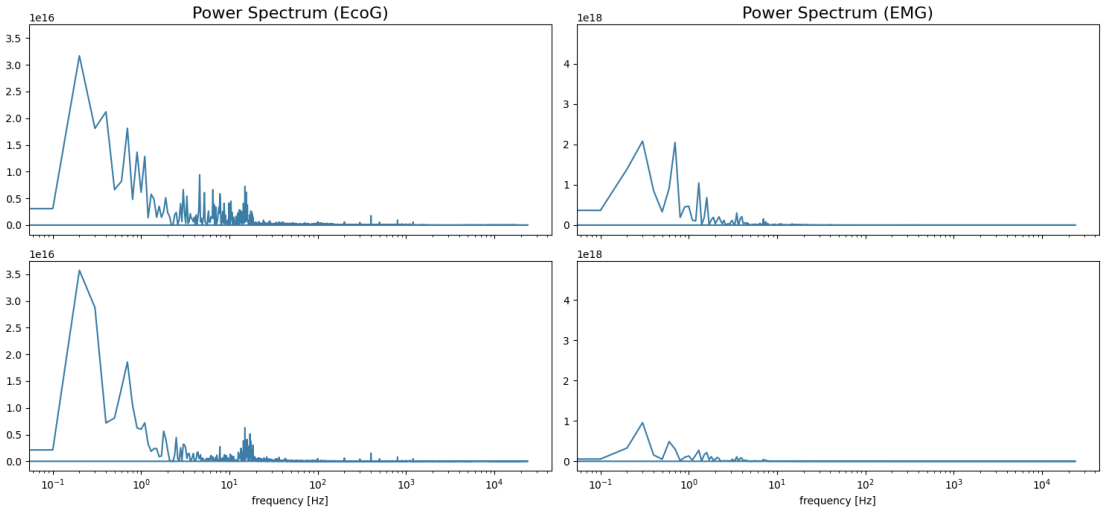


Figure I.0.4: Power spectrum of EcoG and EMG channels of dataset 2.

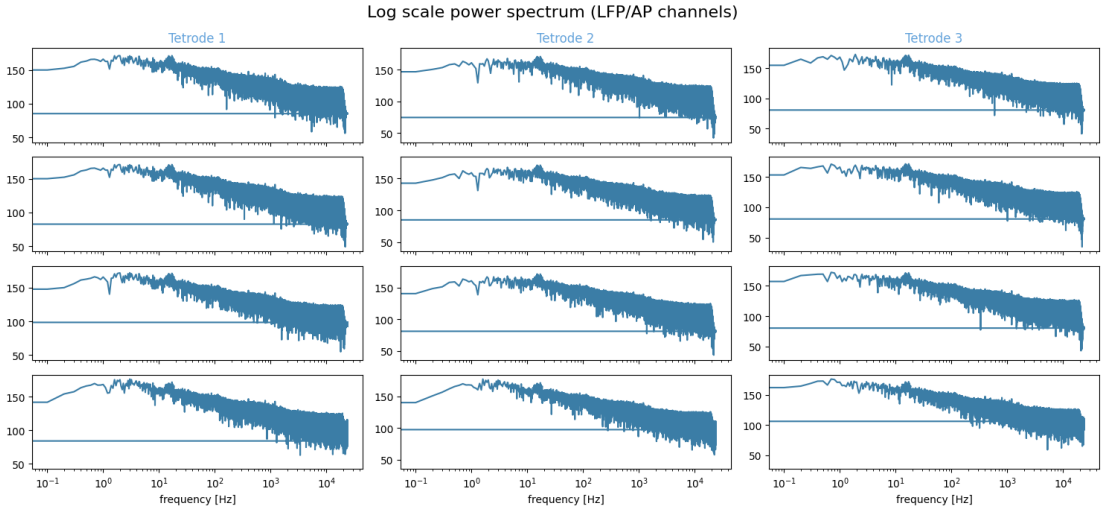


Figure I.0.5: Power spectrum of LFP/EAP channels of dataset 2.

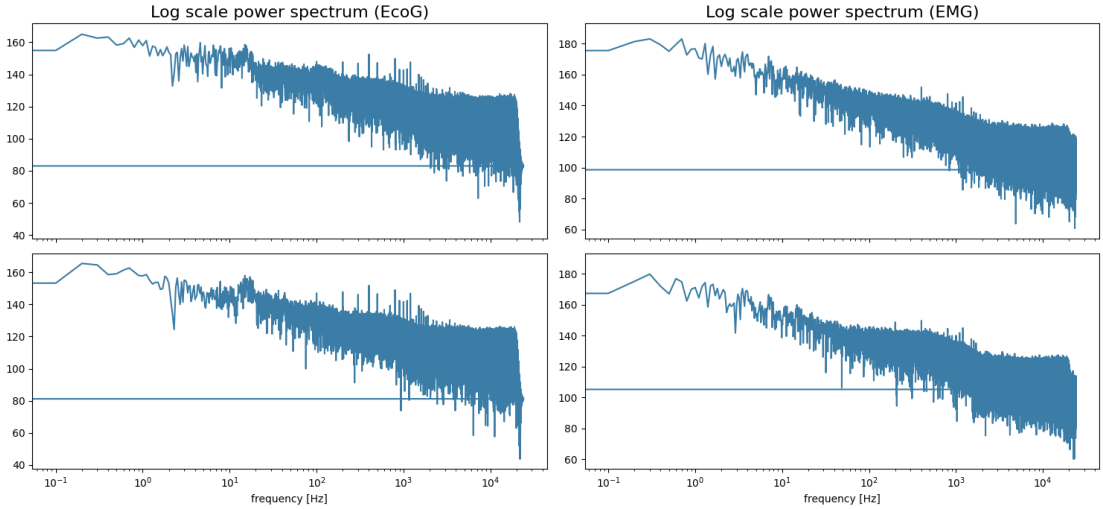


Figure I.0.6: Power spectrum of EcoG and EMG channels of dataset 2.

Dataset 3

Figure I.0.7 and I.0.8 show the frequency power spectrum of dataset 3. Here the y-axis indicates the magnitude of the power in the different frequencies and is not in logarithmic scale. A plot of the frequency power spectrum of the tetrode channels in logarithmic scale is shown in figure I.0.9.

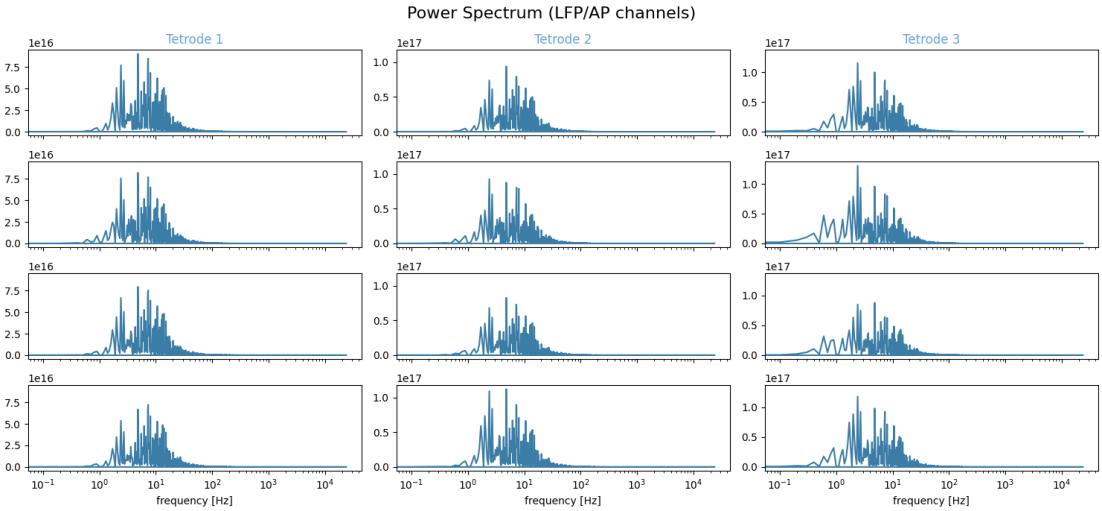


Figure I.0.7: Power spectrum of LFP/EAP channels of dataset 3.

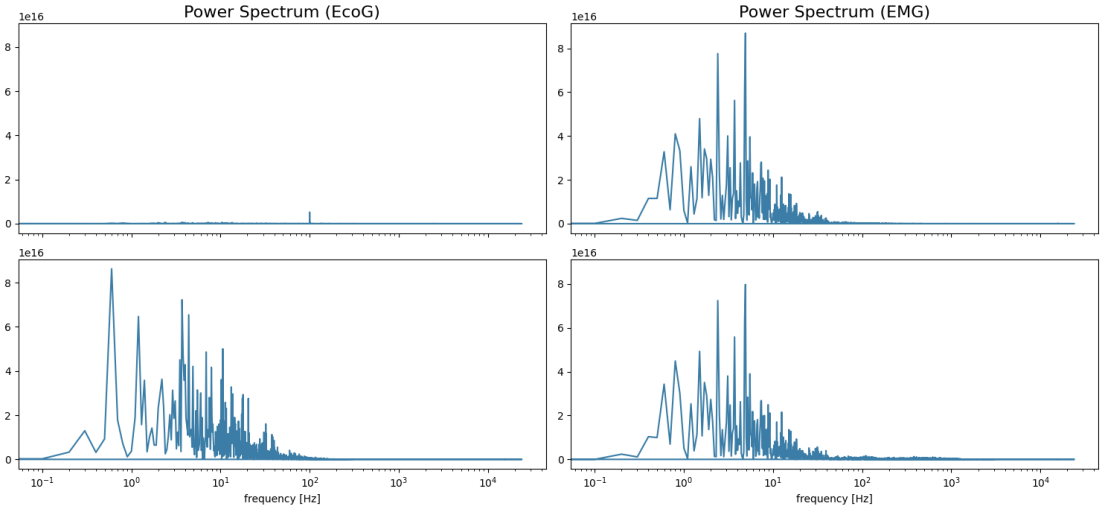


Figure I.0.8: Power spectrum of EcoG and EMG channels of dataset 3.

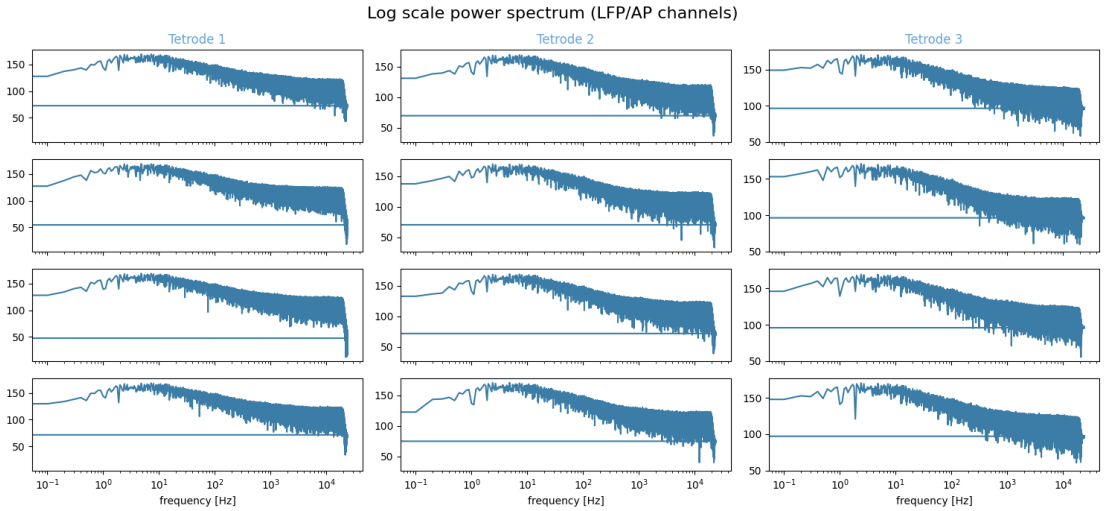


Figure I.0.9: Power spectrum of LFP/EAP channels of dataset 3.

Dataset 4

Plots of the frequency power spectrum of dataset 4 with the y-axis in logarithmic scale are shown in figure I.0.10 and I.0.11.

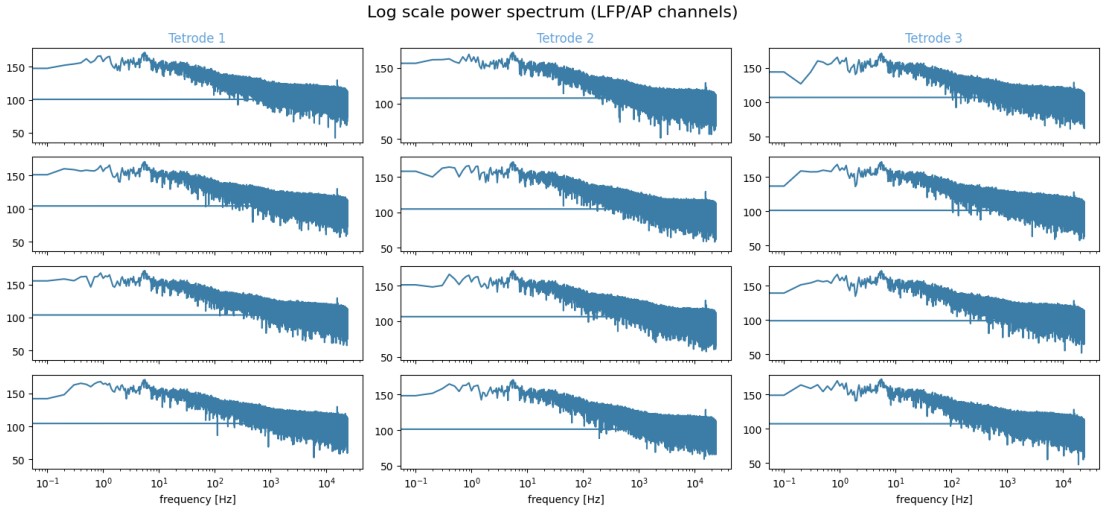


Figure I.0.10: Power spectrum of LFP/EAP channels of dataset 4.

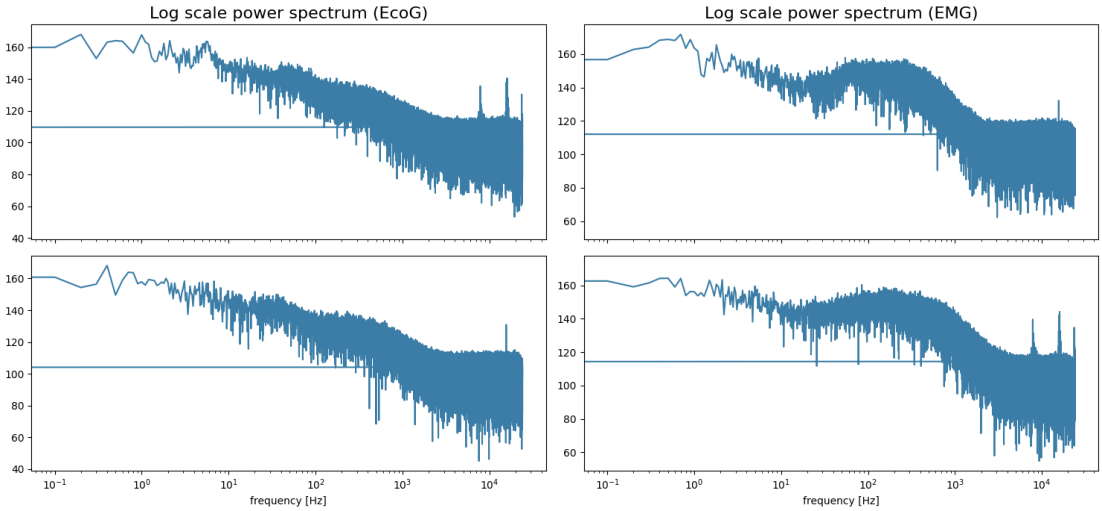


Figure I.0.11: Power spectrum of EcoG and EMG channels of dataset 4.