Computational Biology master's degree at Universidad Politécnica de Madrid (UPM)

# Development of an algorithm to compare electrophysiological signals to for the creation of a brain-to-brain inferface

*Author:* Jorge Vindel Alfageme

*Form teacher:* Fivos Panetsos Petrova

*Form UPM teacher:* Stephan Pollmann

Universidad Complutense de Madrid

## Abstract

Potential differences are generated in neurons and these electrical impulses can be registered as signals. These signals are capable of being registered using microelectrodes and processed with the intention of operating processors, forming a system known as the brain-machine interface. These concepts have served to create the first brain-to-brain interface, in which 2 animals participate. However, in many articles that mention signal processing algorithms that are used in brain-machine interfaces and, specifically, in the one that talks about the first brain-to-brain interface, the process by which the signals are analyzed is not specified. In this work, a new algorithm has been created and shown, with the intention of processing and comparing signals with each other to be able to be used in a brain-to-brain interface construction.

Based on some of the principles mentioned in the first BTBI interface article, a neurophysiological signal processing and comparison algorithm has been created from scratch to be used in the creation of a new interface. The algorithm is based on using signals that come from the average of a sample of electrophysiological signals, to later be compared with new signals. As a result, a ratio is generated that indicates the degree of similarity that exists between the compared signals. This proportion will serve to encode several electrical impulses that will allow the 2 animals that are part of the BTBI interface to interact.

This algorithm represents a simple way to compare electrophysiological signals with each other. The fact that it can be used to process any type of neurophysiological signal allows it to be adapted to the brain-to-brain interface, but also to other interfaces. Despite being a computationally easy algorithm to create and implement, it can be applied to systems involving multiple signals to generate very complex results, making it a very versatile algorithm.

## Introduction

### *Short introduction to brain-machine interfaces, and the first BTBI interface*

Brain-machine interfaces (BMI) are systems that allow artificial devices to be controlled by processing nervous system signals [1]. These interfaces serve to interact with a machine only through biological electrophysiological signals, without the need to physically intervene with the device. During the last few years, these systems have been studied, and very different BMIs have been developed [2–5].

The registration and processing of neural signals has been applied, not only for the design of brain-machine interfaces, but for that of the first brain-to-brain interface (BTBI) [6], which has represented a new paradigm in neurobotics. This master's thesis is inspired by the project for the development of the first BTBI interface.

### *Description and results of the first BTBI interface*

This BTBI [6] used 2 rats. Neural signals were transmitted from the brain of one rat to that of the second rat. By constructing a reward system (based on the rats' deprivation of water), the rats were able to interact with the interface, so that the response given by a rat, which was called the "encoder rat", could be transmitted in the form of electrical pulses on the brain of the other, which was called "decoder rat", so that it could perform the same action as the first one. These rats were in different chambers, where the encoder rat had to respond correctly to a visual or sensory stimulus, so that its neuronal signal recorded directly using electrodes arranged on the motor or somatosensory cortex was processed and sent to the brain of the decoder rat as electrical pulses, so that this rat could act in the same way as the encoder rat only through the intracortical microstimulation received by microelectrodes. In this process, the neural signal was digitized, processed, and then converted into several electrical pulses that were sent to the decoder rat.

In the BTBI, the encoder rat had to discern between 2 different stimuli: either 2 motor stimuli or 2 sensory stimuli. In all the experiments, the rats were previously trained to participate in them. To study the activity of the motor cortex (M1 cortex region), the encoder rat had to choose between pressing either a lever to their right or to their left on the interface. To determine which lever to choose, a light was lit on either the right lever or the left one, so that the lever to choose would be the one on which the light was on. The neuronal signal generated in the motor cortex of the encoder rats that pressed a lever was recorded, processed, and sent as microstimuli to the decoder rat. Depending on whether the lever pressed was the right one or the left one, the neural signal was different and involved the generation of a different number of pulses on the cerebral cortex of the decoder rat. To create a different pulse signal in each case, depending on whether the neural signal was that of pressing the right lever or the left lever, a template created from a set of signals generated when the encoder rat pulsed the left lever was used. This template represents an average signal that can be compared to other neural signals. In this case, an incoming neural signal to press the left lever closely resembled the average signal represented by the template, whereas a signal to press the right lever was identified as a distinct signal. Based on the resemblance to the template, the newly recorded signals generated several pulses that were sent to the decoder rat's cortex, which the decoder rat interpreted to act as the decoder rat did. It should be noted that the rats had undergone several training processes, one of which meant that they learned to associate receiving a certain number of cortical microstimulations with an action they had to perform. That is, before carrying out the tests of the BTBI, the decoder rats were accustomed to receiving a specific number of microstimulations as a signal that they interpreted as pressing the right lever, or the left lever. In experiments involving the motor cortex, when the recorded neural signal implied that the encoder rat had pressed the left lever, a minimum number of only 1 microstimulation was sent, whereas if the signal was produced after pressing the right lever, a maximum number of up to 100 intracortical microstimulations were sent.

In the case of the experiments in which the neuronal activity of the somatosensory cortex (cortex region S1) was studied, the rats were introduced into a cavity where there were movable rods, so that these rods could be arranged to press the whiskers of rats, simulating the sensation that would produce being in a narrow space or, conversely, being separated from each other so as not to apply pressure on their whiskers, which simulates a wide space. When the encoder rats sensed the stimulus of a narrow space, they would press a button to their left in a cavity to receive water as a reward, whereas, if the signal they detected was that of a wide space, they would press a button to the right. The signal from the somatosensory cortex originated by the whiskers of the rat was sent to the cortex of the decoder rat after being processed in the form of microstimuli to perform the same action, without being subjected to the same sensory stimulus. Neural signal processing in this case is the same as in the previous experiment. For these experiments, the template was made from the detection signals of a narrow space, the minimum number of pulses sent would be 1, and the maximum would be up to 50 pulses.

In experiments using the BTBI interface, the decoder rats hit scores were above average. In the case of the lever pressure experiments, the decoder animals obtained a percentage of success around 64%, and considering the experiments of tactile detection of space, the percentage was around 62%. In the original article, it is argued that rats trained through cortical microstimulations have undergone learning through neuronal plasticity stimulated through the electrical impulses generated by the interface on the brain surface.

*Programs used in the design of the signal processing algorithm in the first BTBI ever created*

In the article, it is mentioned that there is a system to process signals and generate electrical pulses as part of the BTBI. It is indicated that a Plexon system was used to record the signals. This neural data was then analyzed using NeuroExplorer software, which in turn used various custom scripts written in MatLab. The signal processing system consists of an algorithm that compares an average signal represented in a template with the newly recorded signals of the encoder rats when they are performing one of the tasks of the experiments. Through the similarity between the 2 signals are computed several electrical pulses, which are finally sent to the cortex of the decoder rat. The result of comparing the newly recorded signal with the template, according to the signal comparison algorithm they use, is a Z-score, which is then translated into a number of microstimulations using a sigmoid function. The Z-score is a statistical term that is used to indicate the number of standard distributions a point is distant from the mean in a normal distribution, and this number can be both negative or positive, depending on the position of the point in the normal distribution, whether it is to the left or right of the mean in the function. However, in the article there is no more information about how they came to generate the templates, how exactly the Z-score is calculated or what characteristics the sigmoid function must transform the value of the Z-score variable into the number of microstimulations. Finally, an electrical microstimulator controlled by a MatLab script generated and sent the electrical pulses.

*Objectives of this master's thesis regarding the interface*

With this master's thesis, as part of a larger project that seeks to create a BTBI with which to repeat the tactile recognition experiments of the original article, in the absence of a detailed algorithm for the processing of signals in the interface, what has been done is to devise 2 algorithms: The first is used to generate a template that represents an average signal from a sample of signals, and the second, is used to compare any signal with the generated template. As a result, a value is generated that represents a similarity ratio, which indicates the similarity that exists between the signal to be compared with the template and the average signal of the template. These 2 algorithms are based on some of the principles mentioned in the original article. For example, templates that are generated that represent average signals are created from a set of signals that are averaged over each other. This means that, when averaged, statistical parameters such as the mean or standard deviation are calculated, which are used when comparing the average signal with the new registered signals to establish the degree of similarity between them. These algorithms will be housed in a Raspberry Pi Model 3B microprocessor. The similarity ratio is intended so that, once stored in a text file, it is sent to a second microprocessor in which the similarity ratio will be used to calculate the number of microstimulations to be sent to the decoder rat since it will control the pulse generator. It is also important to mention that, as in the original work, the neural signal processing system runs and generates its results simultaneously while the rats are being subjected to the experiments. Therefore, the algorithm for calculating the similarity between the signals is executed while the recorded neuronal signals from the encoder rat are received in the microprocessor.

## Materials and methods

In this section, it will be explained which programming language and what additional modules have been used to create the scripts.

The scripts that contain the signal processing algorithms have been written from scratch in Python. Python is a multipurpose programming language in which a variety of libraries or modules have been developed with which, in recent years, research has addressed studies such as in mathematics and physics [7], and biology [8]. Different Python modules have been used to develop the algorithms for this work. Some of these modules are specific to fields such as mathematics or biology, but others are libraries used in many scripts, regardless of their purpose.

In the first script, in charge of creating the template, the “Sys” modules have been used, with which to create arguments from the interpreter from where the program is executed to indicate in the processor where the necessary signal files to create the templates are to be found, “Os”, used to indicate the addresses within the processor where the signals are located and that they can be processed, “NumPy” [9], the most popular and versatile Python module that contains a plethora of functions with which to perform mathematical calculation, among which we even find a set of operations for signal processing, "Pandas" [10], a module used in data analysis that allows the manipulation of large volumes of data at high speed, and "Neo" [11], a library for working with electrophysiological data, which allows reading a variety of neurophysiological file formats.

The second script, intended for the comparison of signals, uses the modules Sys, Os, NumPy, Neo and "SciPy" [12], an ecosystem that contains functions to perform mathematical, physics and engineering calculations, but in this work, it will be used for the functions that allow statistical computations.

## Results

### *Introduction to the biological mechanisms on which the experiments are based*

For the elaboration of any signal processing program in a BMI or BTBI, it all starts from the data provided by the nerve signals that are recorded. A signal consists of a series of amplitude values over time. Neural signals share the same nature as other signals and are susceptible to processing, as is the case with radio signals or sound. In the nervous system, neurons transmit nerve impulses from one to another. These nerve impulses are potential differences that are generated from the change in ionic concentrations that occurs in neurons through the opening of channel proteins on their surface [13–16].

The signals can come from a single neuron, from a group of neurons, or from the potential difference that is transmitted through the tissues and reaches the surface of the skin, as in the case of electroencephalography’s [17]; and in turn, depending on which neuron, sets of them or tissue the electrodes have been arranged, the detected signal will be completely different from any other recorded neuronal signal [18]. That is, there are different factors that cause the registered neuronal signals to be different from each other. This makes neural signals susceptible to registration so that they can later be compared with other signals and establish a degree of similarity between them. The fact of being able to generate a result from the similarity between 2 signals, allows an interface to use this result to carry out different actions, depending on whether the received signals are similar, or not, to other reference signals.

### *Definition of Z-score to understand the algorithm of the original work*

To create the BTBI, the authors of the original paper used a program by which received neural signals were compared with an average signal. This average signal had been made from a set of signals similar to each other and was represented by a template. The template gives rise to a sigmoid function, where the dependent variable was the so-called Z-score. Through the Z-score and using this function, the number of pulses that were to be sent to the brain of the decoder rat was obtained. The Z-score is a statistical concept that could be derived from generating the average signal from a sample of signals. Unfortunately, the original article does not detail which computations the Z-score comes from, nor how the function calculates the number of pulses originated. However, knowing that the function used has the appearance of a sigmoid function, a hypothesis can be made about what the function was like and how the Z-score intervenes in it. Next, it will be explained in more depth what could be the origin of the Z-score that is observed in the article and what kind of function we could be facing.

In statistics, the Z-score is a value that represents the number of standard deviations that a value is far from the mean in a normal distribution [19]. The Z-score value is calculated using the following formula:

$$Z = \frac{x-\mu}{\sigma} \tag{1}$$

where $x$ represents a value along the abscissa axis in the normal distribution, $\mu$ represents the mean of the normal distribution, and $\sigma$ represents the standard deviation of the distribution. In this way, the value of the Z-score is equal to zero when the value of $x$ is equal to that of the mean ($x = \mu$), and it becomes more negative or positive as the value of $x$ distances itself from that of the half. If you want to make a comparison between 2 values in a normal distribution, you can use the Z-score to calculate how much these 2 values are similar or different from each other.

*Discussion about the function that could have been used in the original article*

Now, we are going to consider what could be the origin of the sigmoid function used to calculate the number of pulses sent to the brain of the decoder rat. To do this, we will consider the formula of the sigmoid function, some of its characteristics regarding its range, and how it can be modified based on a series of addends and factors that can be included in the starting formula.

The logistic function is determined by the equation:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The function range is defined by the 2 horizontal asymptotes:

$$\lim_{x\to\infty} \frac{1}{1+e^{-x}} = 1 \tag{3}$$

$$\lim_{x\to-\infty} \frac{1}{1+e^{-x}} = 0 \tag{4}$$

The horizontal asymptote when $x$ approaches minus infinity or the lower horizontal asymptote is determined by an addend that can be incorporated into the formula as follows:

$$f(x) = \frac{1}{1+e^{-x}} + b \tag{5}$$

In this way, this horizontal asymptote of the formula for the sigmoid function becomes the value of the addend $b$:

$$\lim_{x\to-\infty} \frac{1}{1+e^{-x}} + b = b \tag{6}$$

The horizontal asymptote when the variable $x$ approches infinity or the upper horizontal asymptote is determined both by the value of the numerator of the quotient of the formula, which we can define as $a$, and by the addend $a$ that represents the horizontal asymptote when the variable $x$ approaches minus infinity. Thus, the equation of the sigmoid function can be written as follows:

$$f(x) = \frac{a}{1+e^{-x}} + b \qquad (7)$$

where $a + b$ is the value of the horizontal asymptote that is calculated as $x$ approaches infinity:

$$\lim_{x\to\infty} \frac{a}{1+e^{-x}} + b = a + b \qquad (8)$$

On the other hand, let us remember that the function of the original article was elaborated from an average signal of a sample of neural signals to generate a template. In the article, it is mentioned that the sigmoid function that derives from the average signal is determined by a standard deviation value that has been obtained by averaging the neural signals of the sample, and that this standard deviation value affects its position along the abscissa axis and how steep the function is.

Now, if we look again at formula (7), the sigmoid function is more or less steep, depending on the value of a factor that can accompany the exponent $x$ of the function, which can be represented as a denominator for the exponent as follows:

$$f(x) = \frac{a}{1+e^{\frac{-x}{\sigma}}} + b \qquad (9)$$

where $\sigma$ is the denominator of the factor that accompanies the exponent $x$ of the function. The smaller the value of $\sigma$, the greater the factor by which the exponent will be multiplied and the more steep the function will be.

Also, if we add an addend to the numerator of the exponent that represents $x$ in the function, the sigmoid curve will move along the abscissa axis. If this addend is equal to $\sigma$, we obtain the following formula:

$$f(x) = \frac{a}{1+e^{\frac{-x+\sigma}{\sigma}}} + b \qquad (10)$$

When calculating the mean and standard deviation of a normal distribution, the smaller the variance of the data that was used to perform this computation, the smaller the value of the standard deviation. If $b$ is chosen as the minimum value of the function, $a + b$ as the maximum value, the standard deviation $\sigma$ of the normal distribution is used as the denominator of the exponent of the sigmoid function, and the value of $x$ is mapped to the value of the Z-score, then the following equation is obtained:

$$f(Z) = \frac{a}{1+e^{\frac{-Z+\sigma}{\sigma}}} + b \qquad (11)$$

where the value of $f(Z)$ is determined by the value of the Z-score, and where the smaller the value of the standard deviation $\sigma$, the more likely it is that the values of the function correspond to one of the values determined by the asymptotes ($b$, or $a + b$) given a Z-score.

However, despite having obtained a function that could be adapted to the one used in the article, the Z-score values cause the number of pulses to vary abruptly when the value of $\sigma$ is very small. In fact, the article mentions that, by averaging the sample signals, a standard deviation of 0.3 was obtained. On the other hand, the number of pulses sent to the cortex of the decoder rat in the motor cortex experiment ranged from 1 to 100 pulses, and in the somatosensory cortex it ranged from 1 to 50. Using the value of 0.3 for the standard deviation, together with the values of the asymptotes $b$ and $a+b$ of 1 and 50, respectively, in formula (11), a function is obtained with which the values of the pulses vary drastically for very small changes in the value of the Z-score. Faced with a formula whose result varies greatly with small changes in the independent variable, it is impractical to use it in a system in which the data of the neural signals to be used can be affected by many factors and may not be well suited to the formula for conversion to a number of pulses. Given the possibility of using a formula that did not guarantee the correct transformation of the signals into pulses, it was decided to try a new algorithm to build the interface.

*Mathematical foundations on which the new processing algorithm is established*

Due to the absence of a clear explanation of the algorithm for the comparison of signals, a program has been created from scratch with which to calculate the similarity between neuronal signals. Even though in the original article there is little information about the calculation to create the templates with the average signals and the comparison of signals, terms such as the Z-score are mentioned, suggesting that statistical values such as the arithmetic mean and the standard deviation, on which this score depends. The arithmetic mean and standard deviation are values that are associated, like the Z-score, with the normal distribution formula. Therefore, it is possible that the operations of the original algorithm are involved in the normal distribution, which can be considered a starting point for developing a signal processing method.

A normal distribution is a density function that is determined by a mean value $\mu$, and a standard deviation $\sigma$, which is defined by the function:

$$\phi_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{12}$$

It is a symmetric function with respect to the mean, and whose maximum is at point $\phi_{\mu,\sigma^2}(\mu)$.

Considering that values can be entered in the function as points where each of them has a density value associated with it, a proportion of densities can be established with the point $x$ to be compared, with respect to the point $\mu$ of the function associated with the maximum density of the normal distribution, to establish a comparison between the 2 points:

$$\frac{\phi_{\mu,\sigma^2}(x)}{\phi_{\mu,\sigma^2}(\mu)} \in [0,1] \tag{13}$$

If the point $x$ is equal to $\mu$, then the ratio of densities will be equal to 1, while its value will decrease as the point $x$ moves a number of standard deviations away from the mean $\mu$.

Among the most common computational algorithms used for signal processing we found the Fourier transform. A signal constitutes a series of values over time. The Fourier transform makes it possible to trace the frequency domain of a signal from its time domain, so that, for each of the possible frequencies, there is an associated amplitude value. For each signal, there is a frequency domain that represents all the information in the signal, and that constitutes a template that characterizes it. This template is formed by each of the amplitude values of the frequencies represented in the frequency domain. Therefore, 2 signals are capable of being compared with each other based on the frequency domain of each of them. On the other hand, these frequency spectra can be averaged among themselves to calculate a standardized frequency domain, which represents an average signal of all those that have been used in the computation.

By averaging the signals, an arithmetic mean and a standard deviation of each point in the frequency domain can be obtained and, therefore, a normal distribution, with which to estimate the similarity between the frequencies that make up 2 signals different, point-wisely. When the amplitude value of an average signal frequency is established as the mean of a normal distribution with an associated standard deviation, the density ratio represented in equation (13) can be calculated where the amplitude value $x$ of the frequency of a signal is compared with that of the mean $\mu$ of the normal distribution. After this, the arithmetic mean of all the proportions of all the compared frequency values is calculated, and a similarity ratio of the 2 signals, that of the template and the incoming signal, is obtained.

In total, there are 2 different calculations: The first involves the calculation of a template that represents an average signal. To obtain this template, a signal sample is needed. Then, the frequency domain of each of the signals is calculated by the Fourier transform. Once the frequency domain has been calculated, the arithmetic mean and standard deviation of the amplitude values of the frequencies can be calculated point-wisely. The result is a value of arithmetic mean and standard deviation for each of the frequencies, where each value has been calculated from the number of signals that have intervened in the template generation. Thus, with the arithmetic mean and standard deviation associated with a specific frequency, a normal distribution can be drawn. In this normal distribution, other values can be placed along the function, which will have an associated probability density value; and the second algorithm, which will allow you to calculate the similarity that exists between the average signal and a signal that you want to compare, generating as a result a proportion (a value that will oscillate between 0 and 1) that will indicate how similar the 2 signals are with each other according to which, the higher the value, the more similar. The calculation with this second algorithm will be carried out by obtaining the frequency domain of the signal that you want to compare in the first place. By doing this, an amplitude value will have been obtained for each of the frequencies that make up the signal. Now, knowing that for each of the frequencies there is an associated normal distribution in the template, the value of each frequency of the signal to be compared is entered in its corresponding normal distribution of the template, and a density ratio is obtained by applying the equation (13), a ratio between the density of the amplitude value of the frequency and that of the density of the mean. Finally, an average of all the proportions of all the frequencies is calculated, which corresponds to the proportion of similarity between signals.

*Script content*

First, a template is made that represents an average signal from a sample of signals, and then the average signal is compared with new ones. As a result, a file is generated with a ratio that represents the similarity between the average signal and the signal compared to it. As indicated before, 2 different scripts have been written, and each one of them deals with a part of the processing: one script generates the templates, and the second script carries out the comparison between signals and generates the comparison result.

The neurophysiological signal files that are going to be analyzed with the scripts are characterized by having their own format that is used to store data of this class of signals. These signals are recorded in a specialized file format using specialized instrumentation. Among the tools that are part of this instrumentation, we find the microelectrodes, preamplifier, amplifier and filter, and acquisition device. Electrophysiological signals are detected by microelectrodes, which are then passed to the preamplifier, amplifier and filter to increase the size of the signal and reduce noise and, finally, thanks to the acquisition device, they acquire a suitable format so that they can be manipulated in a processor. When recording the electrophysiological signals of an animal using microelectrodes, there may be a variable number of microelectrodes used, and each of them will collect a different signal, depending on the area where they have been placed. Each of the detected signals will be recorded in the file that is generated. Thus, a single file of electrophysiological signals can accumulate several neural signals at the same time.

There are specific file formats that are dedicated to storing electrophysiological signal information. Neurophysiological signal log file formats include, for example, the Spike2 (SMR), NeuroExplorer (NEX) or Plexon (PLX) format. These files are characterized because, in addition to containing the amplitude values of the recorded signals ordered by channels, they contain metadata about the recording conditions of the signals, such as the sampling frequency or their duration. In these files, each channel generally holds a signal, although sometimes the same channel can contain several signals, that is, a channel can contain the information registered by several electrodes, where each of the electrodes has registered a signal. Written scripts contain code to read SMR and PLX format files, such as specialized formats to contain signals, and can also read text files (TXT) where signal points are ordered by columns (in these files, the first column is the time vector, and the successive columns to its right contain, each one, the data of a signal, where each of the points of the signal corresponds to one of the instants in time of the first column).

To extract the information from the signals of the specialized files, the Neo library has been used. Neo is a Python library designed to work with file formats intended to hold electrophysiological information [11], including SMR, NEX, or PLX. To constitute a tool that demands the least possible number of computing requirements, Neo is deliberately limited to reading data, without having functions for data analysis or visualization. Neo implements a data model well adapted to intracellular and extracellular electrophysiological data and electroencephalograms (EEG) that have been recorded with multiple electrodes at the same time. Neo complements NumPy, such that Neo objects are NumPy arrays along with additional metadata.

NumPy is a Python module that brings with it a new variable: Arrays. Arrays are multidimensional variables that allow you to store data to operate with them very efficiently [20]. To work with matrices, it must be considered that the form they must take is a hypercube and, therefore, the length along each dimension must be maintained. The elements of the array are ordered by indexes, and in turn, new elements can be inserted into an array by indexing. In the NumPy library there are functions that are designed to be used with matrices and to perform matrix operations, and in these functions you can also indicate along which dimension you want the operation executed by the functions to be applied. Considering the indexing and functions designed for arrays, it is also possible to apply certain operations to only one set of numbers in the array.

To understand algorithms in scripts, you need to understand how signal data is organized within the script. To efficiently compute the signal data, it has been thought to introduce the values of the signals in matrices. Once organized within the matrices, the arithmetic means and standard deviations of the amplitudes of the frequencies will be calculated to create the templates. Arrays are variables that can have unlimited dimensions. In this case, a matrix has been created that accumulates the data of all the signals and that has 3 dimensions, attending to 3 different values: the number of points that make up the signals, the number of signals per file, and the number of files. The meaning of these 3 values is explained in more detail below. The most important thing to consider is that the three-dimensional matrix that holds the information of these signals will be created from these values.

First, we need to understand what the number of points per signal is. By applying the Fourier transform on a signal, several Fourier coefficients are obtained equal to the number of points that make up the signal. The number of points in a signal is determined by the duration of the recording and the sampling rate. The sampling rate indicates the number of points that are recorded per unit of time and is measured in Hertz (Hz). As the duration and sampling rate of a signal increase, the number of points will increase. To obtain the templates with the mean and standard deviation values after averaging these values between several signals, the number of points between signals needs to be constant. Therefore, of all the signals that the template creation algorithm analyzes, first the signal formed by the largest number of points is traced, and then this number is used to standardize the number of Fourier coefficients that are used. In this way, all signals, when applying the Fourier transform, will assume the same number of Fourier coefficients.

On the other hand, it must be considered that the signal files will contain one or more signals. This number of signals is equal to the number of microelectrodes used when recording them from an animal. Each signal will be recorded in a different region, and will have characteristics, in such a way that a template must be created for each of the microelectrodes used. The number of microelectrodes used is equal to the number of signals found in each file.

Finally, the number of files to create the templates must be considered. Each of the files will contain a constant number of signals. Each signal belongs to a register of a different microelectrode. For example, if 4 different microelectrodes arranged on the brain surface of a rat are used to record the signals, there will be 4 different signals in each generated file. The signals from each microelectrode must be ordered taking into account that, in each file, there will be a signal recorded with each of the microelectrodes, with different characteristics regarding the area of the cerebral cortex in which the microelectrodes have been implanted. Therefore, when averaging the signals, they must first be ordered by the microelectrode from which they come, and then averaged and made the templates. Thus, when ordering the signals per microelectrode for averaging, as each file will have only one signal per microelectrode, for each template a number of signals equal to the number of files that make up the sample will be averaged.

In the first script, the signal files will be read through a function written using the Neo module (Fig. 1). This function allows to read files in SMR or PLX format. In this script, you indicate in which directory the signal files to be used to create the templates are located. First, the files are to be read and the maximum number of points per signal, the number of signals per file and the total number of files used to create the templates will be saved (Fig. 2). In this way, the necessary dimensions will be obtained to create the three-dimensional matrix. The function of this three-dimensional matrix is to hold the amplitudes of the frequency domain of the signals in order, depending on the microelectrode that has registered them (Fig. 3). These amplitudes are the result of applying the Fourier transform on the signals. Once the signals are ordered, the arithmetic mean and standard deviation are calculated in each set of signals depending on the microelectrode that has registered them, and the values are obtained to create the templates.

```
def neo_reader(file_format, file_name): # require Neo module.
    file_format = file_format.upper() # turns the file format name into uppercases.
    if file_format not in ["SMR","PLX"]: # it is executed if the format is not the adequate one.
        print("WARNING (from neo_reader() function): " +
              "File formats that can be processed by " +
              "neo_reader() function are: SMR, PLX. " +
              "Please, use just one of those file " +
              "formats.\n") # error message.
        quit() # quits.
    elif file_format == "SMR": # it is executed if the format is SMR.
        reader = neo.io.Spike2IO(filename = file_name) # the file is read.
        data = reader.read(lazy = False)[0] # the data are stored in a variable.
    elif file_format == "PLX": # it is executed if the format is PLX.
        reader = neo.io.PlexonIO(filename = file_name) # the file is read.
        data = reader.read(lazy = False) # the data are stored in a variable.
    return data # returns data.
```

**Figure 1. Function for reading the files with specific format of electrophysiological signals.** Code written in Python that shows the function that has been created using the Neo module to read files with a typical format of electrophysiological signals. The written function can read 2 kinds of typical formats for signal recording: the Spike2 (SMR) and the Plexon (PLX) format. This function only allows you to read the information from the files to accumulate it in a variable on which other functions of the Neo library can then be used to finally extract the numerical data from the signals.

```
max_time_vector = np.array([]) # empty vector that will store the time vector with the greatest number of points among all signals.
max_pnts = 0 # variable which will store the greatest number of points as an integer.
no_channels = 0 # variable which contains the number of channels.
first_file = True # boolean variable which is used to execute some part of the code only once.

if file_format == "txt": # it is executed if the file format is TXT.
    for File in sample_list: # loop that iterates over each file.
        data = np.loadtxt(sample_signals_path +
                          "/" + File,
                          delimiter = "\t") # file content is read using a function from NumPy.
        data_length = np.shape(data)[0] # signal length.
        if data_length > max_pnts: # it is executed if the variable max_pnts is lower than the signal length.
            max_pnts = data_length # max_pnts variable is overwriten.
            if Y_or_n in ["Y","YES"]: # it is executed if the first 'n' frequencies are going to be used to create the template.
                max_time_vector = data[:,0] # time vector.
                sampling_rate = 1/(max_time_vector[1]-max_time_vector[0]) # sampling rate.
    no_channels = np.shape(data)[1]-1 # number of signals or channels in file.

elif file_format == "smr": # it is executed if the file format is SMR.
    for File in sample_list: # loop that iterates over each file.
        data = neo_reader(file_name = sample_signals_path + File,
                          file_format = file_format) # file is read.
        for seg in data.segments: # iterates over each of the parts of the file.
            for an_sig in seg.analogsignals: # iterates over each of the signals of the file.
                array = np.transpose(np.array(an_sig)) # transposes the array that stores the file signals.
                time = an_sig.times.rescale("s").magnitude # signal time vector is created.
                time_len = len(time) # signal length is measured.
                if time_len > max_pnts: # it is executed if the signal length is greater than the value contained in max_pnts.
                    max_pnts = time_len # max_pnts se sobreescribe con esa longitud.
                    if Y_or_n in ["YES","Y"]: # it is executed if the first 'n' frequencies are going to be used to create the templates.
                        max_time_vector = time # time vector is stored.
                        sampling_rate = an_sig.sampling_rate # its sampling rate is registered.
                if first_file == True: # it is executed if the loop is iterating over the first file.
                    for ind_sig in array: # iterates over each one of the signals in each channel.
                        no_channels = no_channels + 1 # increases the channel number by one unit.
        first_file = False # the boolean variable equals False.

elif file_format == "plx": # it is executed if the file format is PLX.
    for File in sample_list: # loop that iterates over each file.
        data = neo_reader(file_name = sample_signals_path + File,
                          file_format = file_format) # file is read.
        for block in data: # iterates over each one of the blocks from the data.
            for seg in block.segments: # iterates over each of the parts of the file.
                for an_sig in seg.analogsignals: # iterates over each of the signals of the file.
                    array = np.transpose(np.array(an_sig)) # transposes the array that stores the file signals.
                    time = an_sig.times.rescale("s").magnitude # signal time vector is created.
                    time_len = len(time) # signal length is measured.
                    if time_len > max_pnts: # it is executed if the signal length is greater than the value contained in max_pnts.
                        max_pnts = time_len # max_pnts is overwritten.
                        if Y_or_n in ["YES","Y"]: # it is executed if the first 'n' frequencies are going to be used to create the templates.
                            max_time_vector = time # time vector is stored.
                            sampling_rate = an_sig.sampling_rate # its sampling rate is recorded.
                    if first_file == True: # it is executed if the loop is iterating over the first file.
                        for ind_sig in array: # iterates over each one of the signals in each channel.
                            no_channels = no_channels + 1 # increases the channel number by one unit.
        first_file = False # the boolean variable equals False.
```

**Figure 2. Python code fragment to save the maximum number of points per signal, number of signals per file, and number of files.** This code chunk shows how the sample signal files are read depending on their format, using the function described in Figure 1, and how a series of loops iterate over the files and their different segments to extract the necessary information with which to form the three-dimensional matrix.

```
channel_list = np.linspace(1,no_channels,no_channels) # vector that stores the number of channels there are in one file.
no_arr = no_channels # the number of subarrays equals the number of channels.
no_cols = max_pnts # the number of columns in the array equals the maximum number of points in the signals.

arr = np.zeros([no_arr,no_rows,no_cols]) # an empty tridimensional array is created which will contain the template data.

if file_format == "txt": # it is executed if the file format is TXT.
    for File in os.listdir(sample_signals_path): # loop that iterates over each signal file.
        data = np.loadtxt(sample_signals_path + "/" + File,
                          delimiter = "\t") # the file is read.
        file_idx = os.listdir(sample_signals_path).index(File) # the file index is recorded in the file list.
        no_row = file_idx # assigns the file index to a row index from the array.
        for channel in channel_list: # loop that iterates over each channel index from the channel vector.
            no_arr = int(channel-1) # a subarray index is created from each channel index.
            fCoefs = (np.fft.fft(data[:,int(channel)],
                                 n=no_cols)/
                      len(data[:,int(channel)])) # Fourier coefficients are calculated using the signal.
            ampls = np.absolute(fCoefs) # frequency amplitude is calculated using the coefficients.
            ampls[1:len(ampls)] *= 2 # amplitude values are standardized.
            arr[no_arr,no_row,:] = ampls # amplitude values are introduced ordered in the array.

elif file_format == "smr": # it is executed if the file format is SMR.
    for File in sample_list: # loop that iterates over the signal files.
        data = neo_reader(file_name = sample_signals_path + File,
                          file_format = file_format) # the file is read.
        file_idx = sample_list.index(File) # records the file index in the file list.
        no_row = file_idx # assigns the file index to a row index in the array.
        channel_idx = 0 # initial channel index.
        for seg in data.segments: # loop that iterates over the file parts.
            for an_sig in seg.analogsignals: # loop that iterates over each channel.
                array = np.transpose(np.array(an_sig)) # transposes the array that contains the signals.
                for ind_sig in array: # loop that iterates over each one of the rows of the array or signal.
                    fCoefs = (np.fft.fft(ind_sig,
                              n = no_cols)/
                              len(ind_sig)) # signal Fourier coefficients are calculated.
                    ampls = np.absolute(fCoefs) # frequency amplitudes are calculated.
                    ampls[1:len(ampls)] *= 2 # amplitudes are standardized.
                    arr[channel_idx,no_row,:] = ampls # amplitude values are included ordered in the array.
                    channel_idx = channel_idx + 1 # increases by one the channel index number.


elif file_format == "plx": # it is executed if the file format is PLX.
    for File in sample_list: # loop that iterates iver the signal files.
        data = neo_reader(file_name = sample_signals_path + File,
                          file_format = file_format) # the file is read.
        file_idx = sample_list.index(File) # records the file index in the file list.
        no_row = file_idx # assigns the file index to a row index in the array.
        channel_idx = 0 # initial channel index.
        for block in data: # loop that iterates bucle que itera sobre los bloques de datos del archivo.
            for seg in block.segments: # loop that iterates over the file parts.
                for an_sig in seg.analogsignals: # loop that iterates over each signal channel.
                    array = np.transpose(np.array(an_sig)) # transposes the array that has the signals.
                    for ind_sig in array: # loop that iterates over each signal.
                        fCoefs = (np.fft.fft(ind_sig,
                                  n = no_cols)/
                                  len(ind_sig)) # Fourier coefficients are calculated.
                        ampls = np.absolute(fCoefs) # frequency amplitudes are calculated.
                        ampls[1:len(ampls)] *= 2 # amplitudes are standardized.
                        arr[channel_idx,no_row,:] = ampls # amplitude values are inserted oredered in the array.
                        channel_idx = channel_idx + 1 # increases by one the channel index number.
```

**Figure 3. Python code fragment by which the matrix is created, which that holds the amplitudes of the signals in order according to the recording microelectrode.** This code fragment shows the part of the script in which, by reading the signal files and by a series of loops that iterate over them and their different segments, the frequency amplitudes of the signals are calculated and inserted in order into the matrix according to their recording microelectrode. Once these values are in the matrix, the arithmetic mean and standard deviation can be calculated in each set of signals, to finally create the templates that represent an average signal.

In the templates, after calculating the arithmetic means and standard deviations of the signals, we need to establish the number of points that will form each template. In the algorithm, a maximum number of points per signal had been selected to standardize the number of Fourier coefficients and amplitudes that would be calculated by applying the Fourier transform. However, this number can be extremely high, which would reduce the processing speed when using the comparison algorithm. To decrease the number of values that make up each template, 2 ways have been devised to reduce the number of values per template.

The user, when executing the script, decides how the templates will be created considering their number of values. To do this, the user introduces a string as input from the processor terminal when the script is running, which will indicate the method with which the number of values that will form the templates will be selected. The number of values can be chosen considering 2 different ways: the first consists of creating the template considering a percentage of the cumulative sum of the arithmetic means of the amplitudes of the frequency domain, in order to analyze the percentage of the frequency domain information; and the second consists of using only the values of the first “n” frequencies, starting from the 0 Hz frequency, indicating up to what frequency the arithmetic mean and standard deviation values that become part of the templates are kept.

The first form of value selection, based on the calculation of the cumulative sum of the amplitudes and selection of a percentage of the cumulative sum, is since the electrophysiological signals are complex signals, composed of many frequency amplitudes. It is important to note that most of the frequencies with the highest amplitude that form the electrophysiological signals of rat brains are low, mainly between 0 and 15 Hz [21]. Each of the frequencies that make up the electrophysiological signal contributes to forming the signal through its amplitude. The amplitude of the frequencies decreases as the frequency increases in this class of signals. We can think of the signal as a wave for which there are frequencies with greater amplitude that, for this reason, are more relevant when constituting the signal. The idea of selecting from the frequency domain only a set of frequencies considering the cumulative sum of the amplitudes, allows selecting from the initial frequency those amplitude values of the frequencies that are most relevant when constituting the signal.

On the other hand, the second method of selecting values to create the template is based on indicating a frequency, up to which its amplitude values will be incorporated, starting from the frequency of 0 Hz. This method has the advantage over the previous one that it will be able to elaborate templates from signals that have little neuronal activity, contrary to the high number of frequencies that a template created from the cumulative sum of the amplitudes would have if they are registering brain waves which barely show activity.

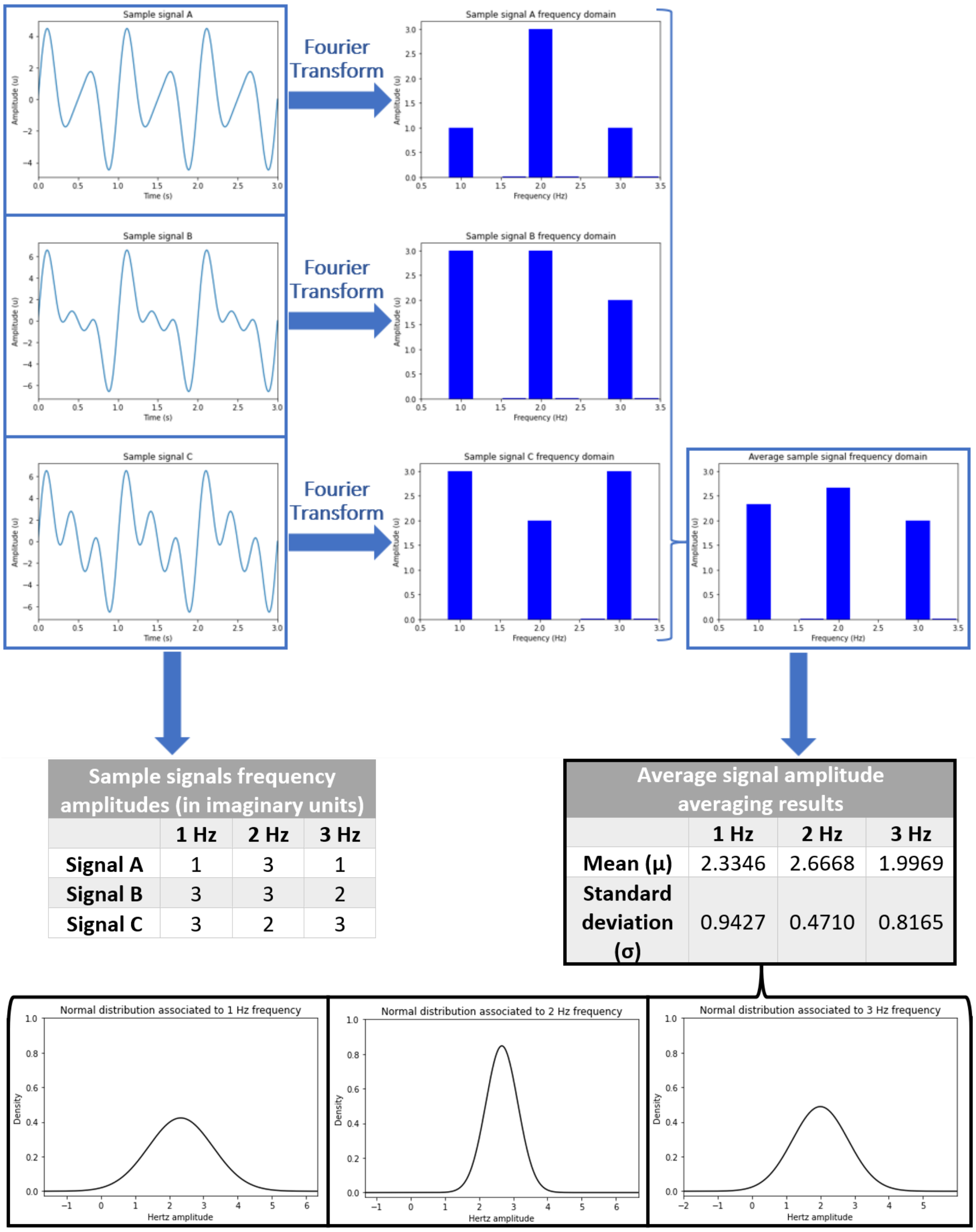
Both methods for the generation of the templates seek that the number of frequencies they contain is relatively small, always without causing a loss of information such that the average signals that are recorded lose their characteristics, due to the fact that the comparison of signals that are performed by the second program uses a loop to compare the similarity between the template signal and an incoming signal, frequency-wisely comparing them, and then calculate a final value of similarity between both.

The lines of code responsible for processing each of the value selection methods that will form the templates are distributed throughout the original script and cannot be shown in their entirety in an image so that they can be seen as they are in the script. However, all the code can be freely accessed from: http://www.github.com/JorgeVindelAlfageme/TFM

Finally, the arithmetic mean and standard deviation values that have been obtained after averaging a set of signals and reducing the number of values, these are written in TXT files that will constitute the templates. Each TXT file will represent a template, and each template collects the information from an average signal. Each signal corresponds to the average of a set of signals recorded by a single microelectrode in each case. The data in the template is represented in 2 columns. In the first column, the data of the arithmetic mean of the amplitudes of the frequencies are shown in order, and in the second column the values of standard deviation can be found. All this data is saved in TXT files using Pandas module.

A small text file is also generated at the end of the script that contains the maximum number of points for the signals. This number will serve so that, in the algorithm for comparing the signals, when the Fourier transform is applied, the same number of Fourier coefficients and amplitudes are calculated in the incoming signals that come from new files. In this way, the comparison between the values of the templates and those of the new signals can be made between the corresponding frequencies, so that the similarity value between the signals is correctly calculated.

The whole process of this algorithm is represented by the diagram in Figure 4, which shows a toy example with a set of signals composed of a reduced number of frequency amplitudes to be averaged.


Sample signal A
Fourier Transform
Sample signal A frequency domain
Sample signal B
Fourier Transform
Sample signal B frequency domain
Sample signal C
Fourier Transform
Sample signal C frequency domain
Average sample signal frequency domain
Amplitude (u)
Time (s)
Frequency (Hz)
Normal distribution associated to 1 Hz frequency
Normal distribution associated to 2 Hz frequency
Normal distribution associated to 3 Hz frequency
Density
Hertz amplitude

| Sample signals frequency amplitudes (in imaginary units) | | | |
|---|---|---|---|
| | 1 Hz | 2 Hz | 3 Hz |
| Signal A | 1 | 3 | 1 |
| Signal B | 3 | 3 | 2 |
| Signal C | 3 | 2 | 3 |

| Average signal amplitude averaging results | | | |
|---|---|---|---|
| | 1 Hz | 2 Hz | 3 Hz |
| Mean (μ) | 2.3346 | 2.6668 | 1.9969 |
| Standard deviation (σ) | 0.9427 | 0.4710 | 0.8165 |

**Figure 4. Toy example that represents a diagram of how the template generation script works.** In this scheme, the operations carried out by the template creation algorithm are summarized through tables and graphs. To exemplify what the algorithm does, 3 simple signals have been represented by their component frequency amplitudes (top left), which will be used to design a small template. The characteristics of the signal frequencies appear in the table on the left, under the representation of the

three signals. The Fourier transform is applied to these 3 signals, and then their frequency domains are averaged (right). After making the average, each of the frequencies that make up the signals will correspond to an arithmetic mean and a standard deviation (right), with which a template will be formed that collects them, and each mean and standard deviation will serve to draw a normal distribution (below) that will later be used to calculate a similarity ratio between signals.

Now, we will explain what the algorithm of the script that compares template signals with new incoming signals consists of in greater depth. As indicated above, the template values for each frequency will be used to create a normal distribution associated with each frequency. Then, when a new signal is registered, the Fourier transform will be applied to it, the amplitudes of its frequencies will be calculated, and the values of the amplitudes will be introduced in each of the corresponding normal distributions. Thus, a probability density can be calculated in each of them. This probability density will be divided by the maximum probability density according to the mean of the normal distribution using equation (13). The result will be a proportion from the ratio of probability densities and, finally, an average of these coefficients will be calculated, and a similarity proportion will be associated with the comparison made between the signals.

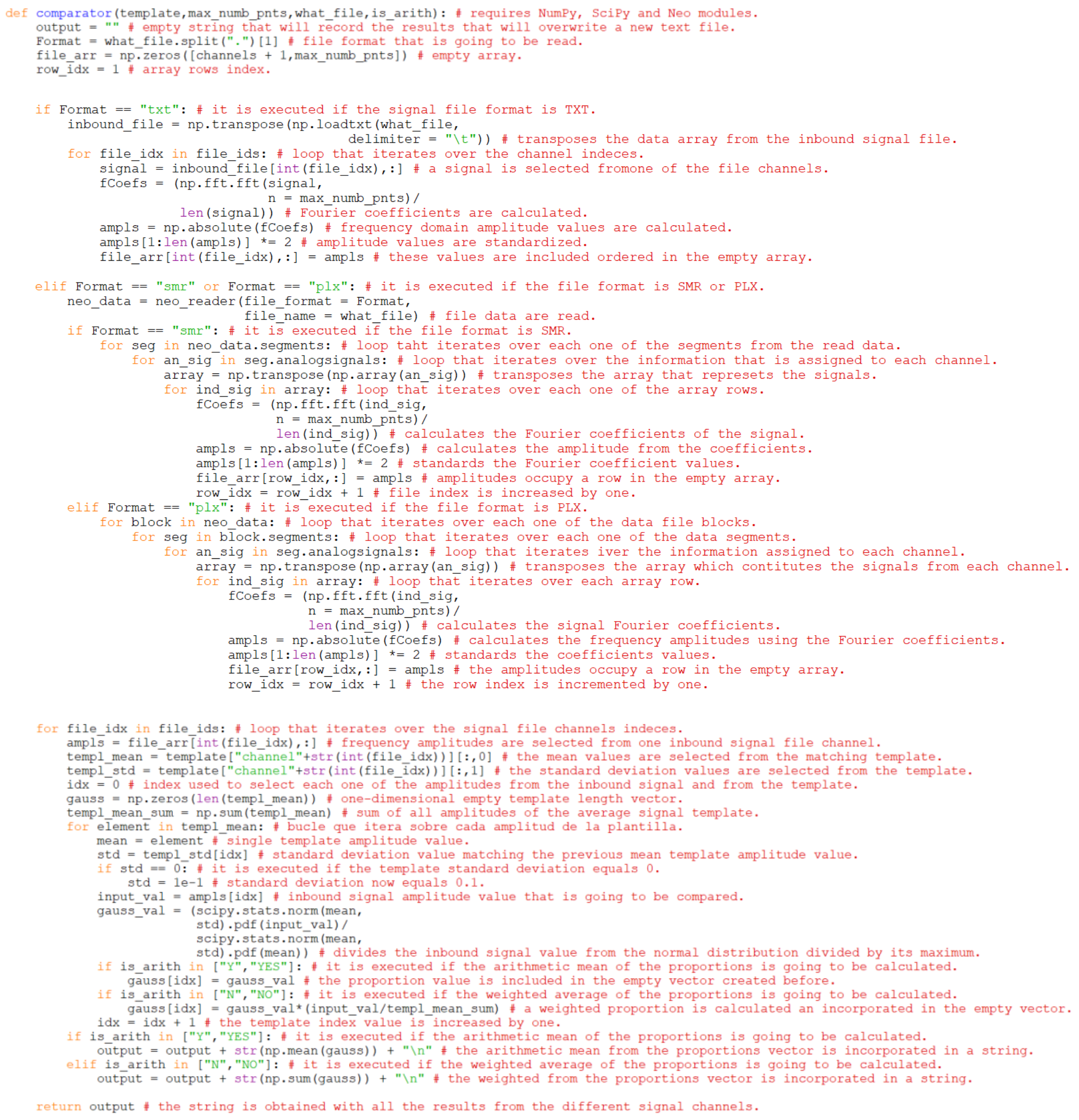
The signal comparison algorithm script has to execute and output the results as new incoming signals reach the processor where it is running. The BTBI is designed so that the encoder animal transmits the neuronal signals and that, simultaneously, the decoder animal receives the processed signal and acts accordingly. For that reason, it is necessary that as new signals arrive, the script is executing and producing the results.

The signal comparison script first reads the data from the templates and stores it in an array, where each average signal is ordered, depending on which template it came from. Next, the script creates a directory in the same location where it is being executed, where the new signal files are expected to arrive, whose signals will be compared with those of the templates.

The comparator algorithm (Fig. 5) uses the matrix containing the information of the templates and the new signal file that has arrived in the directory. The algorithm extracts the signals from the incoming signal file, applies the Fourier transform and obtains the amplitudes of the frequencies of the new signals. Next, it compares the amplitudes by plugging them into the corresponding normal distributions and calculating the ratio of probability densities based on equation (13). Then, the average of the proportions is computed and the result of the similarity between the signals is obtained. This calculation is made with all the templates generated, and each one of the signals that are in the files.

The average of the proportions can be calculated in 2 possible ways. As in the previous script, 2 options are given to the user. The script gives the possibility to calculate either the arithmetic mean of the proportions to calculate the similarity value between signals, or a weighted average of the ratios. This weighted average considers the sum of the products of the probability distribution quotient, represented in equation (13), multiplied by the quotient of the amplitude of the corresponding frequency between the cumulative sum of all the amplitudes of the frequency domain of the signal $\left(\frac{\mu_i}{\sum_{i=1}^{n}\mu_i}\right)$. This weighted average is represented by the following equation:

$$\sum_{i=1}^{n}\left(\frac{\mu_i}{\sum_{i=1}^{n}\mu_i}\cdot\frac{\phi_{\mu,\sigma^2}(x_i)}{\phi_{\mu,\sigma^2}(\mu_i)}\right)\in[0,1] \qquad (14)$$

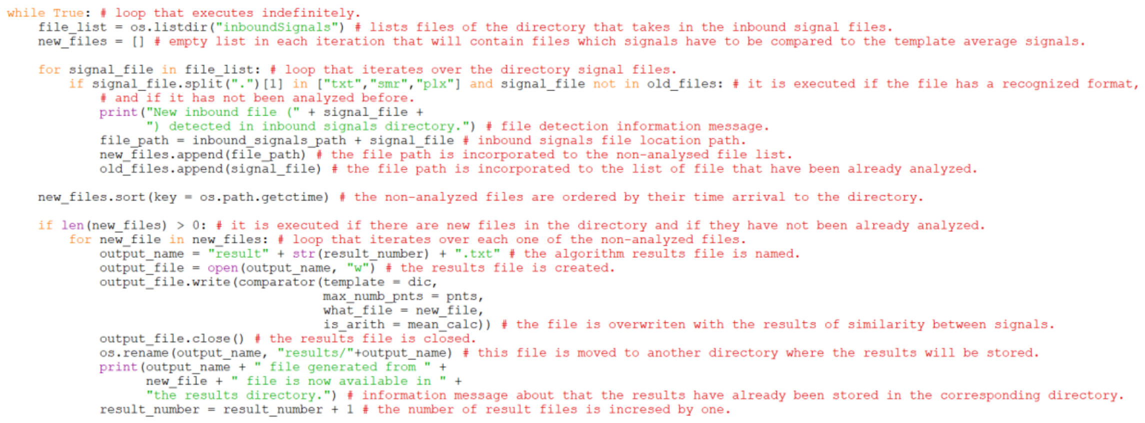


```python
def comparator(template,max_numb_pnts,what_file,is_arith): # requires NumPy, SciPy and Neo modules.
    output = "" # empty string that will record the results that will overwrite a new text file.
    Format = what_file.split(".")[1] # file format that is going to be read.
    file_arr = np.zeros([channels + 1,max_numb_pnts]) # empty array.
    row_idx = 1 # array rows index.


    if Format == "txt": # it is executed if the signal file format is TXT.
        inbound_file = np.transpose(np.loadtxt(what_file,
                                             delimiter = "\t")) # transposes the data array from the inbound signal file.
        for file_idx in file_ids: # loop that iterates over the channel indeces.
            signal = inbound_file[int(file_idx),:] # a signal is selected fromone of the file channels.
            fCoefs = (np.fft.fft(signal,
                                   n = max_numb_pnts)/
                       len(signal)) # Fourier coefficients are calculated.
            ampls = np.absolute(fCoefs) # frequency domain amplitude values are calculated.
            ampls[1:len(ampls)] *= 2 # amplitude values are standardized.
            file_arr[int(file_idx),:] = ampls # these values are included ordered in the empty array.

    elif Format == "smr" or Format == "plx": # it is executed if the file format is SMR or PLX.
        neo_data = neo_reader(file_format = Format,
                               file_name = what_file) # file data are read.
        if Format == "smr": # it is executed if the file format is SMR.
            for seg in neo_data.segments: # loop taht iterates over each one of the segments from the read data.
                for an_sig in seg.analogsignals: # loop that iterates over the information that is assigned to each channel.
                    array = np.transpose(np.array(an_sig)) # transposes the array that represets the signals.
                    for ind_sig in array: # loop that iterates over each one of the array rows.
                        fCoefs = (np.fft.fft(ind_sig,
                                    n = max_numb_pnts)/
                                    len(ind_sig)) # calculates the Fourier coefficients of the signal.
                        ampls = np.absolute(fCoefs) # calculates the amplitude from the coefficients.
                        ampls[1:len(ampls)] *= 2 # standards the Fourier coefficient values.
                        file_arr[row_idx,:] = ampls # amplitudes occupy a row in the empty array.
                        row_idx = row_idx + 1 # file index is increased by one.
        elif Format == "plx": # it is executed if the file format is PLX.
            for block in neo_data: # loop that iterates over each one of the data file blocks.
                for seg in block.segments: # loop that iterates over each one of the data segments.
                    for an_sig in seg.analogsignals: # loop that iterates iver the information assigned to each channel.
                        array = np.transpose(np.array(an_sig)) # transposes the array which contitutes the signals from each channel.
                        for ind_sig in array: # loop that iterates over each array row.
                            fCoefs = (np.fft.fft(ind_sig,
                                        n = max_numb_pnts)/
                                        len(ind_sig)) # calculates the signal Fourier coefficients.
                            ampls = np.absolute(fCoefs) # calculates the frequency amplitudes using the Fourier coefficients.
                            ampls[1:len(ampls)] *= 2 # standards the coefficients values.
                            file_arr[row_idx,:] = ampls # the amplitudes occupy a row in the empty array.
                            row_idx = row_idx + 1 # the row index is incremented by one.


    for file_idx in file_ids: # loop that iterates over the signal file channels indeces.
        ampls = file_arr[int(file_idx),:] # frequency amplitudes are selected from one inbound signal file channel.
        templ_mean = template["channel"+str(int(file_idx))][:,0] # the mean values are selected from the matching template.
        templ_std = template["channel"+str(int(file_idx))][:,1] # the standard deviation values are selected from the template.
        idx = 0 # index used to select each one of the amplitudes from the inbound signal and from the template.
        gauss = np.zeros(len(templ_mean)) # one-dimensional empty template length vector.
        templ_mean_sum = np.sum(templ_mean) # sum of all amplitudes of the average signal template.
        for element in templ_mean: # bucle que itera sobre cada amplitud de la plantilla.
            mean = element # single template amplitude value.
            std = templ_std[idx] # standard deviation value matching the previous mean template amplitude value.
            if std == 0: # it is executed if the template standard deviation equals 0.
                std = 1e-1 # standard deviation now equals 0.1.
            input_val = ampls[idx] # inbound signal amplitude value that is going to be compared.
            gauss_val = (scipy.stats.norm(mean,
                         std).pdf(input_val)/
                         scipy.stats.norm(mean,
                         std).pdf(mean)) # divides the inbound signal value from the normal distribution divided by its maximum.
            if is_arith in ["Y","YES"]: # it is executed if the arithmetic mean of the proportions is going to be calculated.
                gauss[idx] = gauss_val # the proportion value is included in the empty vector created before.
            if is_arith in ["N","NO"]: # it is executed if the weighted average of the proportions is going to be calculated.
                gauss[idx] = gauss_val*(input_val/templ_mean_sum) # a weighted proportion is calculated an incorporated in the empty vector.
            idx = idx + 1 # the template index value is increased by one.
        if is_arith in ["Y","YES"]: # it is executed if the arithmetic mean of the proportions is going to be calculated.
            output = output + str(np.mean(gauss)) + "\n" # the arithmetic mean from the proportions vector is incorporated in a string.
        elif is_arith in ["N","NO"]: # it is executed if the weighted average of the proportions is going to be calculated.
            output = output + str(np.sum(gauss)) + "\n" # the weighted from the proportions vector is incorporated in a string.

    return output # the string is obtained with all the results from the different signal channels.
```

**Figure 5. Written function that uses the data from the average signals to compare electrophysological signals and generate a result as a proportion.** The function uses the arithmetic mean and standard deviation data to plot a normal distribution for each frequency of the average signals from the templates. Then, it calculates the frequency domain of the signals in a new file and compares them with their corresponding templates through the amplitude values in the normal distribution. Finally, it computes an average of proportions. This calculation can be done in 2 different ways: as an arithmetic mean or as a weighted average.

After being compared, a text file is created containing a similarity ratio for each template. Similarity ratios are ordered by row, with a ratio per row. Text files provide a space-saving medium to efficiently store numerical information and can be easily read by any script or program. This text file is stored in a folder destined to accumulate the text files with the results. In this directory, there will be one output file for each of the incoming signal files. These text files will be sent to a second processor connected to a pulse generator. In this processor the files will be read, the percentages of similarity will be processed and based on these values, it will send a signal to the generator indicating the number of pulses that will be sent to the brain cortex of the decoder animal.

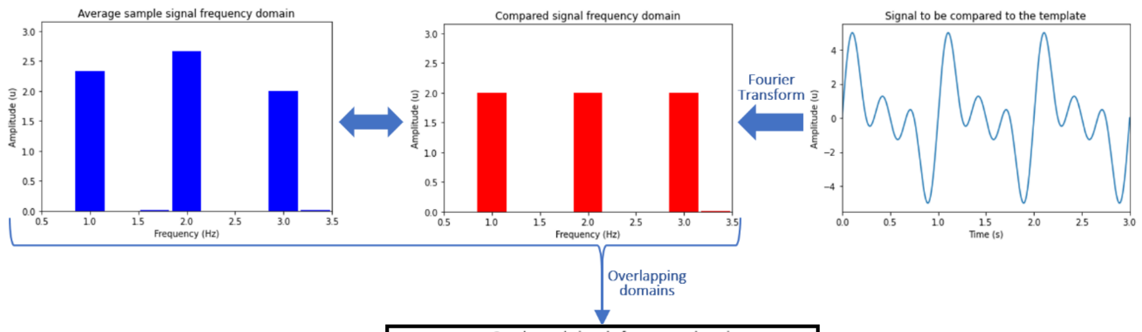
All this algorithm to compare signals and generate the result files is included within an infinite loop (Fig. 6) that allows this system to run simultaneously as new signal files are received, so that the experiments of the BTBI can be carried out in such a way that new signals are registered and the necessary results are emitted to send electrical microstimulations to the brain of the decoder animal.

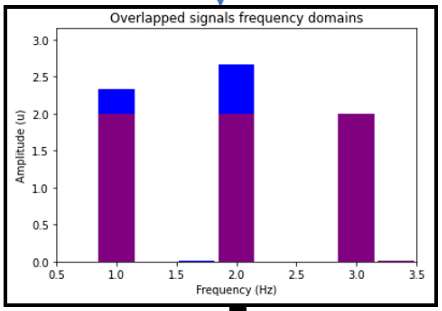


```
while True: # loop that executes indefinitely.
    file_list = os.listdir("inboundSignals") # lists files of the directory that takes in the inbound signal files.
    new_files = [] # empty list in each iteration that will contain files which signals have to be compared to the template average signals.

    for signal_file in file_list: # loop that iterates over the directory signal files.
        if signal_file.split(".")[1] in ["txt","smr","plx"] and signal_file not in old_files: # it is executed if the file has a recognized format,
            # and if it has not been analyzed before.
            print("New inbound file (" + signal_file +
                  ") detected in inbound signals directory.") # file detection information message.
            file_path = inbound_signals_path + signal_file # inbound signals file location path.
            new_files.append(file_path) # the file path is incorporated to the non-analysed file list.
            old_files.append(signal_file) # the file path is incorporated to the list of file that have been already analyzed.

    new_files.sort(key = os.path.getctime) # the non-analyzed files are ordered by their time arrival to the directory.

    if len(new_files) > 0: # it is executed if there are new files in the directory and if they have not been already analyzed.
        for new_file in new_files: # loop that iterates over each one of the non-analyzed files.
            output_name = "result" + str(result_number) + ".txt" # the algorithm results file is named.
            output_file = open(output_name, "w") # the results file is created.
            output_file.write(comparator(template = dic,
                                         max_numb_pnts = pnts,
                                         what_file = new_file,
                                         is_arith = mean_calc)) # the file is overwriten with the results of similarity between signals.
            output_file.close() # the results file is closed.
            os.rename(output_name, "results/"+output_name) # this file is moved to another directory where the results will be stored.
            print(output_name + " file generated from " +
                  new_file + " file is now available in " +
                  "the results directory.") # information message about that the results have already been stored in the corresponding directory.
            result_number = result_number + 1 # the number of result files is incresed by one.
```

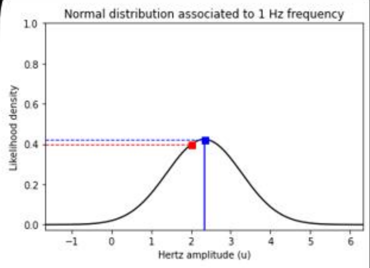
**Figure 6. Lines of code showing the infinite loop that repeatedly allows the comparison calculation between signals.** The infinite loop constantly executes the signal comparison algorithm each time a new signal file arrives at the processor. This calculation compares the signals, produces a ratio that measures the similarity between them for each compared signal, and records these values in a text file for each signal file.

A toy example has been used again to exemplify what this algorithm does in Figure 7, through various graphs, diagrams and tables.

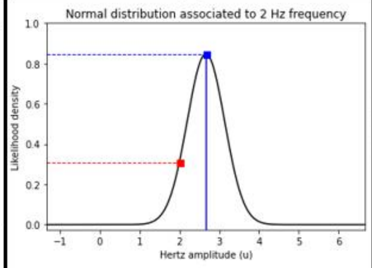




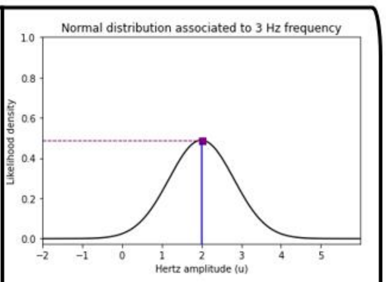
| Template signal and signal to be compared frequency domain values sheet | | | |
|---|---|---|---|
| | 1 Hz | 2 Hz | 3 Hz |
| Template mean (μ) | 2.3346 | 2.6668 | 1.9969 |
| Template standard deviation (σ) | 0.9427 | 0.4710 | 0.8165 |
| Signal to be compared amplitude (imaginary units) | 2 | 2 | 2 |



| Results of 1 Hz frequency from the template distribution | |
|---|---|
| Distribution density of compared signal value ($\phi_{\mu,\sigma^2}(x)$) | 0.3973 |
| Distribution density of mean ($\phi_{\mu,\sigma^2}(\mu)$) | 0.4232 |
| Density proportion ($\frac{\phi_{\mu,\sigma^2}(x)}{\phi_{\mu,\sigma^2}(\mu)}$) | 0.9389 |



| Results of 2 Hz frequency from the template distribution | |
|---|---|
| Distribution density of compared signal value ($\phi_{\mu,\sigma^2}(x)$) | 0.3109 |
| Distribution density of mean ($\phi_{\mu,\sigma^2}(\mu)$) | 0.8471 |
| Density proportion ($\frac{\phi_{\mu,\sigma^2}(x)}{\phi_{\mu,\sigma^2}(\mu)}$) | 0.3671 |



| Results of 3 Hz frequency from the template distribution | |
|---|---|
| Distribution density of compared signal value ($\phi_{\mu,\sigma^2}(x)$) | 0.4886 |
| Distribution density of mean ($\phi_{\mu,\sigma^2}(\mu)$) | 0.4886 |
| Density proportion ($\frac{\phi_{\mu,\sigma^2}(x)}{\phi_{\mu,\sigma^2}(\mu)}$) | 1.0000 |

| Results of proportion averaging: | Arithmetic mean of proportions $\left(\frac{\sum_{i=1}^{n}\frac{\phi_{\mu,\sigma^2}(x_i)}{\phi_{\mu,\sigma^2}(\mu_i)}}{n}\right)$: 0.7670 | Weighted average of proportions considering template frequency amplitudes $\left[\sum_{i=i}^{n}\left(\frac{\mu_i}{\sum_{i=1}^{n}\mu_i}\cdot\frac{\phi_{\mu,\sigma^2}(x_i)}{\phi_{\mu,\sigma^2}(\mu_i)}\right)\right]$: 0.7384 |
|---|---|---|

**Figure 7. Toy example showing data processing performed by the signal comparison algorithm.** Using the data of the average signal represented in the template, its values are compared with those of the frequency domain of a new signal (above). As each frequency of the template has an associated normal distribution, the values of the amplitudes of the new signal are introduced into their corresponding normal distributions, and the probability density ratio between its value and that of the mean of the normal distribution is obtained (below). Finally, proportion average can be calculated in 2 ways, according to the 2 operations available in the algorithm. These results are shown for the toy example in the inferior tables of the image.

## Conclusion

With this work, it has been possible to develop an algorithm that allows comparing electrophysiological signals. This algorithm is based on the elaboration of templates that represent average signals, built from a sample of signal files. Then, the frequency domain of new signals is used to compare the amplitude value of the frequencies point-wisely with a data template and based on this a final result is produced. The creation of this algorithm has been based on some of the principles mentioned in the signal comparison program that was used to create the first BTBI [6], where they used templates that represented average signals and the new signals were compared with the templates from a Z-score to obtain a degree of similarity between them and find out the number of microstimulations that should be sent to the brain of the decoder animal at the interface.

This algorithm must fulfill its function within the system that constitutes the BTBI. The microstimulations generated by the pulse generator and the previous signal processing reach the brain of the decoder animal so that, after a learning process in which the animals have been subjected using intracortical microstimulations before experimenting with the BTBI, they are able to identify a number of pulses and act subsequently. The authors of the original paper believe that electrical pulses on the cerebral cortex should induce neural plasticity to cause animals to memorize the meaning of each set of electrical pulses.

It has been considered what future applications an interface may have that allows several animals to communicate with each other. Until now, the BTBI serves to convey the intention to do a task. In the future, with a more complex design and with a greater number of components, the authors of the article raise the possibility that it can be used as a biological computer with which to reach heuristic conclusions to solve problems.

Until now, experiments that have been carried out using the BTBI collected electrophysiological signals from neuronal assemblies. However, in the central nervous system another series of electrophysiological signals are produced that are capable of being recorded, which could also serve to create templates that contain average signals and that could be compared with new incoming signals. Among some of the signals that could be recorded and used in the interface system, we would find electroencephalography's [17] and local field potentials or LFP [22], which are defined as transient imbalances in the ionic concentrations of the extracellular matrix of the nervous tissue, produced by the generation of potential differences in adjacent cells, which could imply the use of less invasive methods.

The result of the signal comparison algorithm is a ratio, a value that ranges from 0 to 1, indicating the degree of similarity between an average signal and the signal that has been compared to it. The number of signals to be recorded per file and templates will depend on the number of recording electrodes. This allows the same algorithm to be used to compare signals from the recording of many electrodes placed at different points in the nervous system, which offers the possibility of encoding a wide variety of actions, which increases dramatically with the use of one more electrode in each case. Therefore, this algorithm, despite the simplicity that it may imply to obtain a single proportion as a result, the combination of many different values can be used to encode very complex responses in the interfaces that use it.

This algorithm, which has been developed from scratch, not only has the possibility of being applied in systems such as the BTBI, but also in BMIs that use a system based on the creation of templates for the subsequent comparison of signals. The numerical results of script operations can not only be used to generate pulses that are sent to the cortex of other animals but they can also be used to produce responses in other processors directly. Therefore, the algorithm created offers versatility whenever it is used to process electrophysiological signals of neuronal origin composed of low-frequency waves.

## Bibliography

[1] Jackson, A. & Zimmermann, J. B. "Neural interfaces for the brain and spinal cord restoring motor function". *Nat Rev Neurol* **8**, 690–699 (2012).

[2] Ethier, C. *et al.* "Restoration of grasp following paralysis through brain-controlled stimulation of muscles". *Nature* **485**, 368–371 (2012).

[3] Koralek, A. *et al.* "Corticostriatal plasticity is necessary for learning intentional neuroprosthetic skills". *Nature* **483**, 331–335 (2012).

[4] Lebedev, M. A. *et al.* "Future developments in brain-machine interface research". *Clinics (Sao Paulo) 66 Suppl* **1**, 25–32 (2011).

[5] Moritz, C. T. *et al.* "Direct control of paralysed muscles by cortical neurons". *Nature* **456**, 639–642 (2008).

[6] Pais-Vieira, M. *et al.* "A Brain-to-Brain Interface for Real-Time Sharing of Sensorimotor Information". *Sci Rep* **3**, 1319–1329 (2013).

[7] Borcherds, P. H. "Pyhton: a language for computational physics". *Comput Phys Commun* **177**, 199–201 (2007).

[8] Lubbock, A. L. R. & López, C. F. "Programmatic modelling for biological systems". *Curr Opin Syst Biol* **27**, 1–16 (2021).

[9] Dubois, P. F. *et al.* "Numerical Python". *Comput Phys* **10**, 262–268 (1996).

[10] Mckinney, W. "Data Structures for Statistical Computing in Python". Proceedings of the 9th Python in Science Conference (2010).

[11] García, S. *et al.* "Neo: an object model for handling electrophysiology data in multiple formats". *Front Neuroinform* **8**, 1–10 (2014).

[12] Virtanen, P. *et al.* "SciPy 1.0: fundamental algorithms for scientific computing in Python". *Nat Methods* **17,** 261–272 (2020).

[13] Hodgkin, A. L. *et al.* "Measurement of current-voltage relations in the membrane of the giant axon of Loligo". *J Physiol* **116**, 424–448 (1952).

[14] Hodgkin, A. L. & Huxley, A. F. "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo". *J Physiol* **116**, 449–472 (1952).

[15] Hodgkin, A. L. & Huxley, A. F. “The components of membrane conductance in the giant axon of Loligo”. *J Physiol* **116**, 473–496 (1952).

[16] Hodgkin, A. L. & Huxley, A. F. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. *J Physiol* **117**, 500–544 (1952).

[17] Haas, L. F. "Hans Berger (1873-1941), Richard Caton (1842-1926), and electroencephalography". *J Neurol Neurosurg Psychiatry* **74**, 7–9 (2003).

[18] Woeppel K. *et al.* “Recent advances in neural electrode-tissue interfaces”. *Curr Opin Biomed Eng* **4**, 21–31 (2017).

[19] Seashore, H. G. “Methods of expressing test scores”. *Test Service Bulletin* **48**, 7–10 (1955).

[20] van der Walt, S. *et al.* “The NumPy array: a structure for efficient numerical computation”. *Comput Sci Eng* **13**, 22–30 (2011).

[21] Llinás, R.R. "The intrinsic electrophysiological properties of mammalian neurons: insights into central nervous system function". *Science* **242**, 1654–1664 (1988).

[22] Legatt, A. D. *et al.* "Averaged multiple unit activity as an estimate of phasic changes in local neuronal activity: effects of volume-conducted potentials". *J Neurosci Methods* **2**, 203–217 (1980).