Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT)

Laboratorio Nacional de Fusión (LNF)

Internship Report

Identification and classification of Zonal Flows from the electrostatic potential of a turbulent plasma

by

Victor Letzelter

Laboratorio Nacional de Fusión

Ciemat

Supervisor: **Ulises Losada Rodríguez**
Co-Supervisor: **Patrick Ganster**

June - August 2021

Contents

0 Acknowledgements and context of work

This report describes an internship realised in CIEMAT (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas), in cooperation with the french engineering school EMSE (École des Mines de Saint-Étienne).

Acknowledgements

I would like to thank Ulises not only for his guidance during the internship, but also for the fascinating discussions we had together. Ulises introduced me to the work of researcher in a relaxed atmosphere, and taught me to be pragmatic about the dream of Fusion Energy, and about the discoveries as a researcher in general. Otherwise, I would also thank Xavier BAY and Patrick GANSTER, my teachers at EMSE. Xavier BAY was one of my mathematics teachers, and advised me some ways to explore in this exciting internship. And finally, I thank the physics teacher Patrick GANSTER, for its support during the stay in Spain. These three months were also a chance to discover the beautiful city of Madrid, and the Spanish culture.

Global description

CIEMAT is a Spanish public research institution in energy and environment located in Madrid. The work was perform in the LNF (Laboratorio Nacional de Fusión) department, the Spanish center of reference for magnetically confined plasma fusion studies. The department owns its own reactor, the TJ-II, a medium-size heliac-type Stellarator. The start of design and fabrication of the TJ-II was made in 1991, and the first plasmas were produced in 1997. While continuing studies on the TJ-II, the LNF is also involved in many international projects (JT-60SA, ITER and IFMIF). [1] The internship was attached to the Experimental Physics Division of the TJ-II group.

[1] See `http://www.fusion.ciemat.es/international-projects/`

1 Introduction

On the 13^{th} of August, the US National Oceanic and Atmospheric Agency (NOAA) states that the month of July was the warmest month ever recorded for the Earth. [1] As the effects of climate change are beginning to be felt by such announcements, thought about new sustainable ways to produce energy have risen. Nowadays, the world primary electric mix is based on Coal $(39,2\%)$, Natural Gas $(22,8\%)$, Hydroelectricity $(16,3\%)$, Nuclear power $(10,6\%)$, Wind, solar and Geothermal energy $(4,9\%)$, Oil $(4,1\%)$ and Biomass $(2,2\%)$. In this framework, Fusion energy is deemed as a hope, because it would provide carbon-free and abundant energy, without the drawbacks of Fission energy (risk of chain reaction, radioactive wastes).

The reaction that occurs in Fusion Energy Reactors (Stellarators, Tokamaks) is frequently compared with the Sun's process for energy production. This comparison isn't accurate, as the Sun is plunged into the void space, but the plasma that flows into a reactor is a few meters from a wall, which involves *boundary* processes. Among other things, the plasma from a fusion reactor is subject to huge gradients of physical quantities : for instance the temperature required in a Fusion reactor is greater than 100 million degrees Celsius. These gradients generates electromagnetic flows and turbulences, phenomenons which will be at stake for this internship and for which the comprehension is crucial for the development of an effective nuclear fusion reactor.

The principal limits of Fusion relate to three characteristics:

- Respect the Lawson criterion: produce more energy than what is needed to maintain the plasma. For that, the *triple product* $nT\tau_{\mathrm{E}}$, where n, T and τ_{E} are respectively the density, the temperature and the confinement time, has to be maximized. [2]
- Retrieve energy from the reactor and produce the electricity.
- Maintain the fusion and the plasma without disruptions (violent events that leads to a sudden loss of magnetic confinement).

Today, many hopes are pinned on the ITER (International Thermonuclear Experimental Reactor) project, that takes place in Saint-Paul-lès-Durance (France), which emanates from an international cooperation : ITER runs by seven member parties ; European Union, China, India, Japan, Russia, South Korea, United States, the United Kingdom and Switzerland. This project aims to demonstrate the possibility to produce a ten-fold gain of plasma heating power ; with an input power of about 50 MW, a reactor has to create a plasma of 500 MW, for periods of 400 to 600 seconds.

To reach nuclear fusion, two light nuclei (like Deuterium, Tritium) have to fuse to form a single one. [3] For that, the two cores of atoms have to overcome electrostatic

[1] `https://www.cbsnews.com/news/climate-change-july-2021-hottest-month-noaa/`

[2] ITER should then be the first reactor to which the Lawson criterion.

[3] The most famous fusion reaction is this one

$$ {}^{2}_{1}\mathrm{D} + {}^{3}_{1}\mathrm{T} \longrightarrow {}^{4}_{2}\mathrm{He} + {}^{1}_{0}\mathrm{n} $$

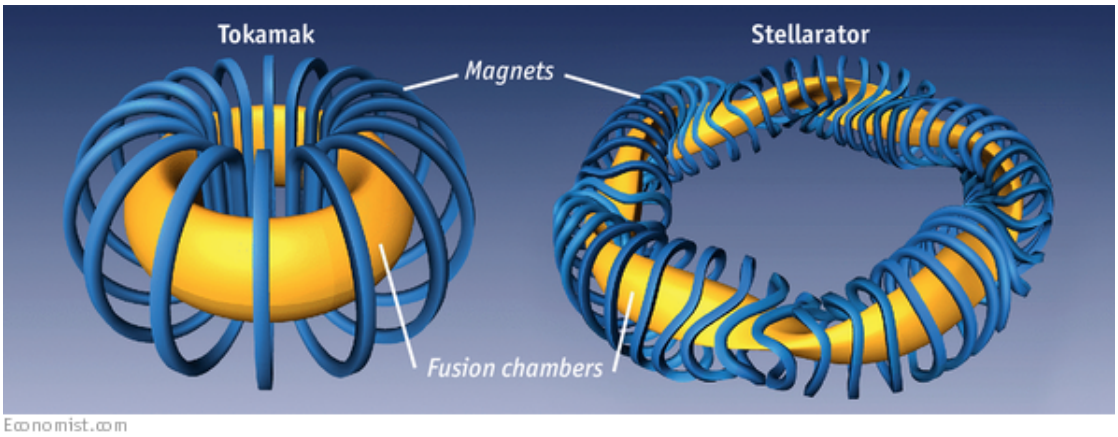

Figure 1: Difference of Geometry between a Tokamak and a Stellarator (from `Economist.com`)

repulsion due to their charges, which are both positive. Fusion reaction requires temperature and density conditions in which the matter is in Plasma state, a Plasma that needs to be confined. There is currently two main ways to confine a plasma ; Inertial and magnetic confinement Fusion. In the framework of this internship, magnetic confinement is adopted,in a device called a *Stellarator* (see 1). This device presents some differences with Tokamaks, which is the most famous device used in controlled nuclear fusion. [1]

This reaction is exothermic, with an energy of about 17.6 MeV

[1] See `http://fusionwiki.ciemat.es/wiki/Stellarator_reactor` for the differences between Tokamaks and Stellarators.

2 Generalities about plasma Physics and Fusion Energy

This section aims to describe the basics in Plasma physics, for the reader to figure out the stakes of the internship.

2.1 Definition of a plasma

Definition 1 (Plasma)**.** *A plasma is **ionized gas**, **globally neutral**, that displays **collective effects**. **Globally neutral** signifies that the plasma is neutral in a scale that is higher than a defined scale (In this case the Debye Length λ_D).* [1] *The **collective effects** imply that one-to-one interactions are weak.*

Proposition 1. *If we consider a plasma as a system of N different charged species with charges $(q_j)_{j\in[\![1,n]\!]}$, with densities $(n_j(\mathbf{r}))_{j\in[\![1,n]\!]}$ at position $\mathbf{r}$, the Poisson equation is verified by the electrostatic potential inside the plasma*

$$\varepsilon\nabla^2\Phi(\mathbf{r}) = -\sum_{j=1}^{N} q_j n_j(\mathbf{r}) - \rho_{\text{ext}}(\mathbf{r}) \tag{1}$$

Where $\varepsilon := \varepsilon_r\varepsilon_0$ is the permittivity of the medium and ρ_{ext} is a charge density external to the medium.

Proposition 2. *Assuming that the the system to be in thermodynamic equilibrium with a heat bath at absolute temperature T, the concentration of the j-th charge species can be described by the Boltzmann distribution*

$$n_j(\mathbf{r}) = n_j^0 \exp\left(-\frac{q_j\Phi(\mathbf{r})}{k_{\text{B}}T}\right)$$

In the high temperature limit $q_j\Phi(\mathbf{r}) \ll k_{\text{B}}T$, and assuming that the plasma is globally neutral $(\sum_{j=1}^{N} n_j^0 q_j = 0)$ the linearized Poisson-Boltzmann equation is

$$\nabla^2\Phi(\mathbf{r}) = \lambda_{\text{D}}^{-2}\Phi(\mathbf{r}) - \frac{\rho_{\text{ext}}(\mathbf{r})}{\varepsilon}$$

Where

$$\lambda_{\text{D}} = \left(\frac{\varepsilon k_{\text{B}}T}{\sum_{j=1}^{N} n_j^0 q_j^2}\right)^{1/2}$$

[2] *More specifically, the Debye length for electrons in vacuum can be written as*

$$\lambda_{\text{De}} = \sqrt{\frac{\varepsilon_0 k_{\text{B}}T_e}{n_e e^2}}$$

[1] See the proposition 1

[2] When the temperature of each charges (q_j) are not uniform, the expression becomes

$$\lambda_{\text{D}} = \left(\frac{\varepsilon k_{\text{B}}}{\sum_{j=1}^{N} \frac{n_j^0 q_j^2}{T_j}}\right)^{1/2}$$

Which is a good approximation of λ_D *when* $T_e \gg T_i$. [1]

In plasma physics, the Debye length has therefore a relevant meaning, as it corresponds to the maximum scale at which the plasma deviates from neutrality.

2.2 Magnetic confinement of a plasma

The magnetic confinement of a plasma is based on electromagnetic pinches, which take profit of the compression by electromagnetic forces. The three pinches geometry at stake are the θ-pinch, the Z-pinch, and the screw pinch. The Z and θ Pinch correspond to two configurations for the orientations of the current and the magnetic field. The Figures 2 and 3, taken from [Wik04a] depict the two different configurations of the directions of the magnetic field (in purple), and those of the currents (in yellow). The θ-pinch makes plasma more resistant to instabilities, while the Z-pinch enhances the confinement properties. The screw-pinch tends to take profit of both of these aspects.

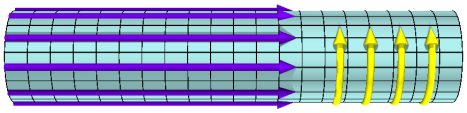

Figure 2: θ-pinch

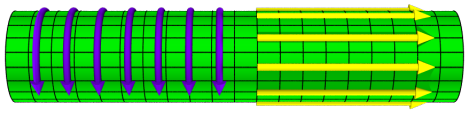

Figure 3: Z-pinch

Therefore, the magnetic field in a Tokamak is characterized by two components : The poloidal filed $\vec{B}_{pol}$ and the toroidal component $\vec{B}_{tor}$ (**Figure 4**). The resulting field $\vec{B}$ has then an helicoidal shape.

2.3 The drifts

Motion in a static and uniform magnetic field

A particle in a static and uniform magnetic filed $\vec{B}$ follows an helical trajectory, which is characterized by two speeds ; $\vec{v}_{\|}$ the speed parallel to the field, and $\vec{v}_{\perp}$ the speed perpendicular to it. The simplest case for an helical trajectory is therefore characterized by a parallel speed $\vec{v}_{\|}$ uniform straight and $\vec{v}_{\perp}$ in a uniform circular motion around the field line with a radius ρ equal to [2] $\rho = \frac{mv_{\perp}}{|q|B}$ (See the left picture, **Figure 5**).

[1] T_e and T_i are the temperature of electrons and ions

[2] The Larmor radius.

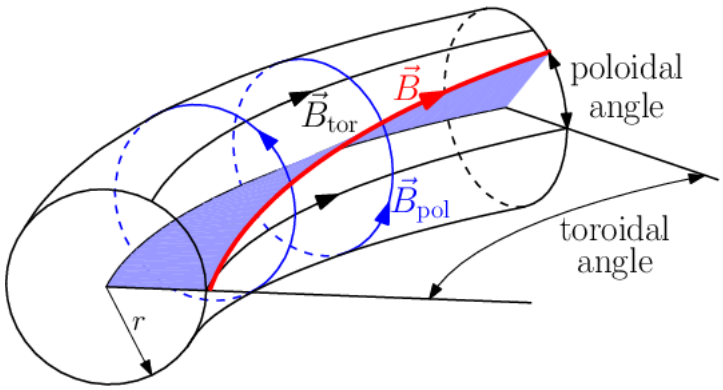

Figure 4: Poloidal and toroidals fields in a Tokamak

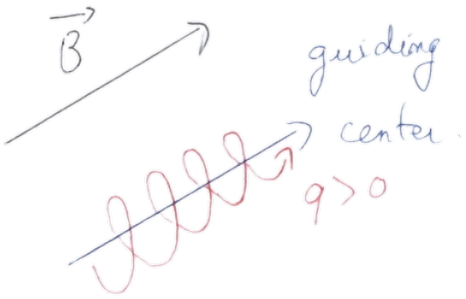

Figure 5: Motion in uniform and static $\vec{B}$ field

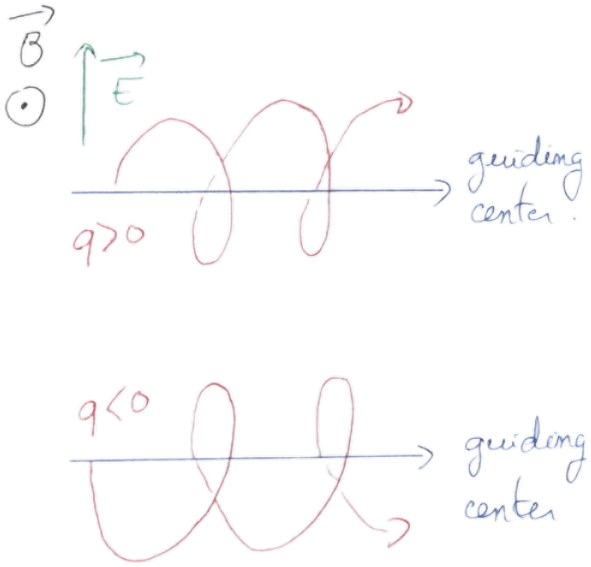

Figure 6: Motion in uniform and static $\vec{B}$ and $\vec{E}$ fields

Motion in a static and uniform electric and magnetic field

When a charged particle is subject to a magnetic $\vec{B}$ and an electric field $\vec{E}$, its motion will be the superposition of a gyro motion and a drift motion $\vec{v}_E$ given by this expression

$$\vec{v}_E = \frac{\vec{E}_\perp \times \vec{B}}{B^2}$$

Where $\vec{E}_\perp$ refers to the component of $\vec{E}$ which is perpendicular to the magnetic field.

This drift is called a $\vec{E} \times \vec{B}$ drift, and is a particular case of drift as it derives from the electrostatic force $\vec{F} = q\vec{E}$. The general definition is given below.

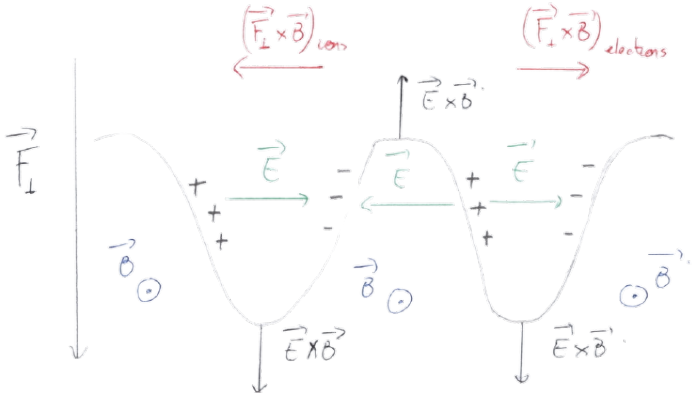

Figure 7: Alternating Electric field generated by a force $\vec{F}$

Proposition 3 (Drift). *A particle of charge q, plunged into a magnetic filed $\vec{B}$ and subject to a Force $\vec{F}$ feels the drift*

$$\vec{v}_F = \frac{\vec{F}_\perp \times \vec{B}}{qB^2} \tag{2}$$

As a plasma is a collection of electrons and ions of opposite charges, the effect of the drift due to $\vec{F}$ will be a separation of the positive and the negative charges into the plasma. [1] Thus, this generates an electric field $\vec{E}$ inside the plasma, that therefore produces $\vec{E} \times \vec{B}$ drifts. This otherwise highlights the crucial importance of gradients consideration in the plasma (pressure, temperature, density), as it generates forces involved in these drifts.

More precisely, the electric field generated by the force $\vec{F}$ is periodic and alternates perpendicularly to the magnetic field (See **Figure 7**). The result is called a sheared $\vec{E} \times \vec{B}$ flow, which are involved into the phenomenon at stake in this internship (See **Section 3**). These $\vec{E} \times \vec{B}$ hence increase the original perturbation. This instability is called a Rayleigh Taylor instability in a plasma [Sin20].

Turbulence

The phenomenon at stake in this internship is called *drift wave–zonal flow turbulence* (See **Section 3**) involves both turbulence and drift, and how these phenomenons interact with each other. Thus, a review of the dynamics of turbulent events is made.

Turbulence can be described by the Navier-Stokes equation for the fluid velocity $\mathbf{v}$ of a an incompressible neutral fluid

$$\begin{cases} \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \frac{1}{R_e}\Delta \mathbf{v} \\ \nabla \cdot \mathbf{v} = 0 \end{cases}$$

[1] This is due to the presence of q in the equation 2

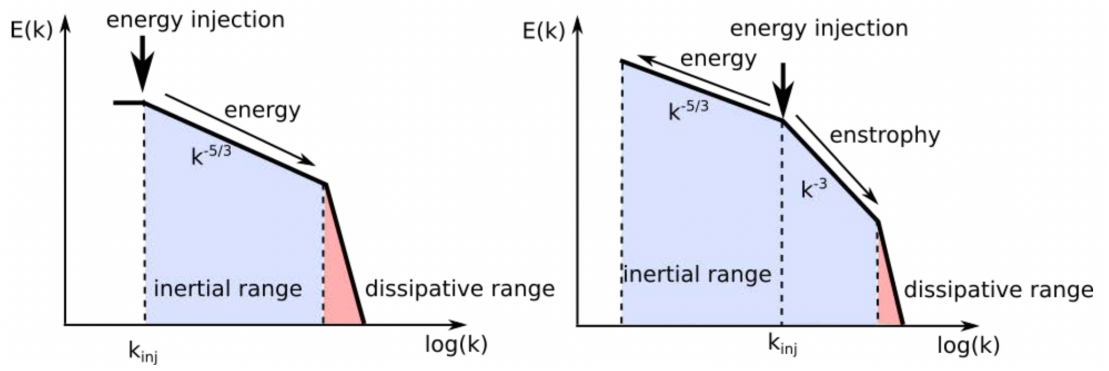

Figure 8: Spectral energy $E(k)$ against the wave number k for 3D (left) and 2D (right) turbulence (taken from [Med17])

Where p is the pressure, and R_e is the Reynolds number defined by the relation

$$R_e = \frac{u_0 L \rho}{\mu}$$

Where u_0 is the mean velocity, L is the characteristic system scale, ρ is the mass density and μ the viscosity.

This Reynolds number precisely describes how *turbulent* a flow is. For high R_e values, turbulence develops and eddies of different sizes are formed. Then, they can split or merge. These two cases correspond respectively to the direct and the indirect *cascade*. The process in which the energy is transferred to small scales until the smallest scales where energy is dissipated is called the direct cascade, and the indirect cascade corresponds to the opposite case. The first exact mathematical theory of turbulent cascades was developed by Kolmogorov (K41 Theory, see [FK95]). The dynamic of vortices differs in 2D and 3D turbulence.

The K41 theory states that energy is injected into the system at some large scale (that refers to k_{inj} in **Figure 8**, injection range), then transferred towards smaller scales (inertial range) and finally dissipated through viscosity (dissipative range). In 2D fluids, energy is transferred from injection scale towards larger scales and enstrophy towards smaller scales. The two directions are called inverse cascade for energy and direct cascade for enstrophy transfer (See **Figure 8**).

Turbulence suppression by shear flow

The main point to figure out the stakes of this internship is that the sheared $\vec{E} \times \vec{B}$ flows interact with the turbulence eddies ; they stretch and suppress them (See **Figure 9**). This process decreases the turbulence diffusion coefficient, and thus the radial transport to the reactor wall (See [Med17]). Then it enhances the quality of the magnetic confinement.

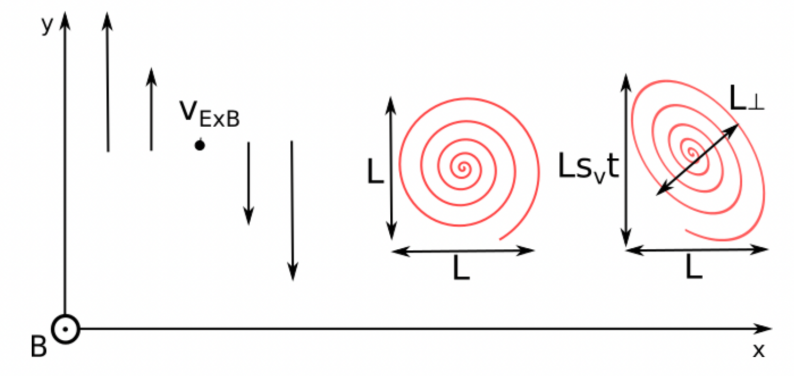

Figure 9: Stretching of eddies by $\vec{E} \times \vec{B}$ sheared flows

The energy from the turbulence is then transferred to another structure called a *Zonal Flow*, which will be investigated in this internship.

Definition 2 (Zonal Flow)**.** *A zonal flow is a toroidally symmetric band-like shear flow* ($n = 0$)*, driven by non-linear interactions including electrostatic potential fluctuations with finite radial wavenumber* k_r*. Zonal flows are generated through the Reynold Stress* $\langle v_\theta v_r \rangle$*, and they have low frequency and wavenumber.*

Definition 3 (Safety Factor)**.** *Let's consider a torus* [1] *with a major radius of* R*, and* r *is the minor radius of a particular* (*circular*) *flux surface. Given a magnetic surface, the safety factor is defined as*

$$q(r) = \frac{1}{2\pi} \oint \frac{r}{R} \frac{B_{tor}}{B_{pol}} \mathrm{d}\theta$$

Where the integral is calculated over a poloidal circuit around the flux surface. A resonance exists if the safety factor takes a rational value $q = \frac{m}{n}$ *on a particular flux surface. In this case the field lines close on themselves after performing* m *poloidal and* n *toroidal turns.* [2] *The integers* m *and* n *are called respectively poloidal and toroidal modes.*

[1] The definition can be generalized to Stellarators as well.
[2] If $q \notin \mathbb{Q}$ the field lines never close on themselves and cover the entire magnetic surface.

3 Summary of the theoretical physics background

Turbulence in Plasma drives radial particles transport and then, it reduces significantly the efficiency of magnetic confinement. To counterbalance, Zonal Flows arise by the self-organization of turbulence into larger flows. They are driven by sheared flows, that stretches the eddies, turning them into patterns that are toroidally and poloidally symmetric. Sheared flows are themselves generated by drift waves, which themselves are generated by Free Energy Sources, gradients of temperature, and gradients of density. ZF are regulating turbulence due to drift waves that generates these sheared $\vec{E} \times \vec{B}$ flows. The interaction between Drift Waves and Zonal Flows can be described as a predator-prey model [Med17] ; ZF are feeded by the turbulence, but they dissipate their energy due to the viscosity of the fluid then turbulence disappear.

Formally, ZF are characterized by $n = 0$ and $m \simeq 0$, where m and n are respectively the poloidal and torodial modes. Otherwise, in the context of a Tokamak, [Dia+05] defines it as *azimuthally symmetric band-like shear flows*. The study of these ZF is crucial, in the sense that the dynamic of these $\vec{E} \times \vec{B}$ drifts and ZF, which are driven exclusively by non-linear interactions, plays a key role in the L-H transition : the transition from Low to High confinement mode.

4 About the experiment and the set up

4.1 The Stellarator TJ-II

The Stellarator TJ-II is the second largest operational stellarator in Europe (after W7-X in the Max Planck Institute for Plasma Physics, Germany).

The global description of the TJ-II given in `http://www.fusion.ciemat.es/tj-ii-2/` is the following

TJ-II has a major radius of 1.5 m, an averaged minor radius 0.2 m and a magnetic field of 1 T. The highest temperatures are achieved by microwave heating (800 kW at 53 GHz) and by accelerated neutral hydrogen beam (1.6 MW and 30 keV). TJ-II has 92 access windows for experimental diagnostics and machine systems.

4.2 The experiment

It will be question here of detecting when ZF appear during the periods of NBI (Neutral Beam Injection) and ECRH (Electron Cyclotron Resonance Heating) in the Stellarator TJ-II. This detection can be achieved by detecting patterns [1] in the time series of the plasma electrostatic potential.

More precisely, two Langmuir Probes are used for the detection of ZF in the Plasma. Their location in the Stellarator is given in **Figure 13**, one is a Rake Probe (B) and the

[1] See below for more precision

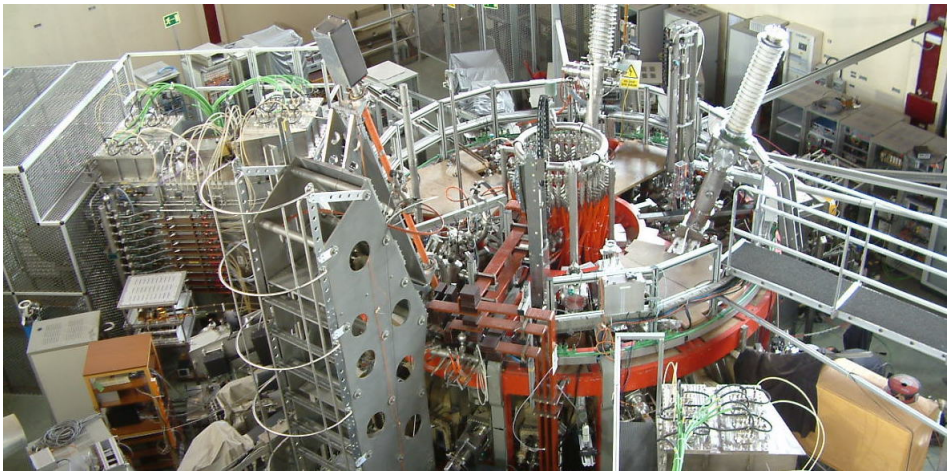

Figure 10: Top view of the TJ-II Stellarator

Figure 11: Picture in front of the TJ-II

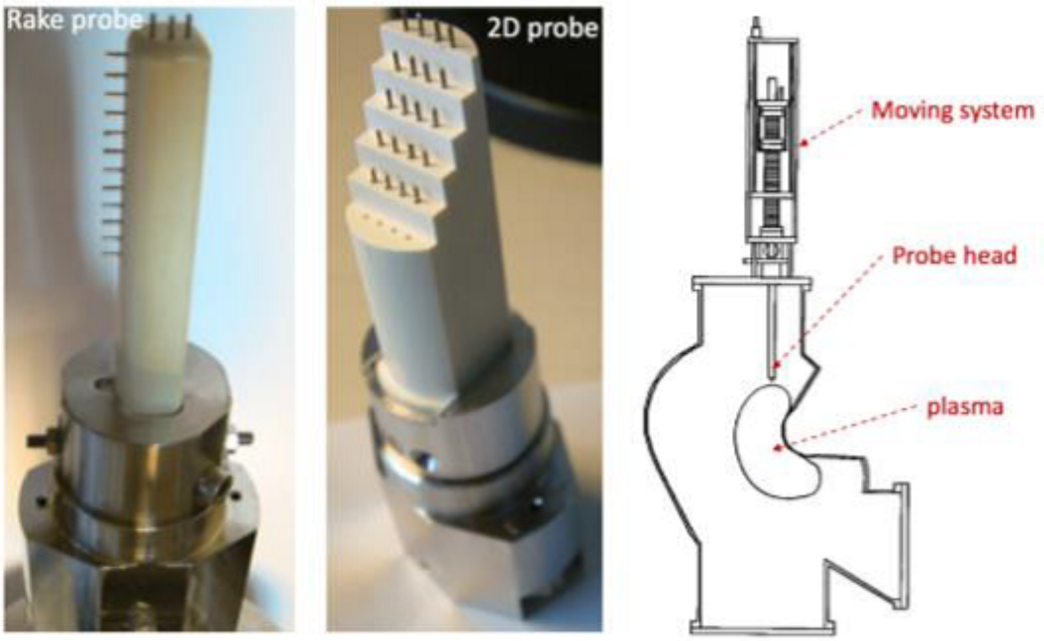

Figure 12: Two probes used in the experiment

other is a 2D Probe (D) (cf **Figure 12**).

Both Probes B and D can measure the floating *electrostatic potential* of the plasma, which is slightly lower than the *plasma potential*. Indeed, when the electrode of a probe is placed into the plasma, the flux of electrons and ions that reaches the electrode will be unequal, mostly due to the higher mobility of ions. As a result, the electrode will be charged negatively until the electron flux is reduced by electrostatic repulsion, and become equal to the ions flux. It is this value of the potential reached by the probe which is called the floating potential, and the difference between the *floating potential* and the plasma potential will be called the *sneath potential*.

For our experiment, the sneath potential will be assumed to be constant as such manner that events that occur in the plasma potential can also be detected in the floating potential of the plasma.

5 Formulation of the problem

The data, which is plotted in **Figure 14**, consists on the two times series of the floating electrostatic potential, which are measured by both probes. The period between the two first vertical red lines stands for the ECRH, and the one between the second and the third line represents the period during which the plasma is heated by NBI.

It will be now be question of highlighting some characteristics of the floating potential time series when ZF occur.

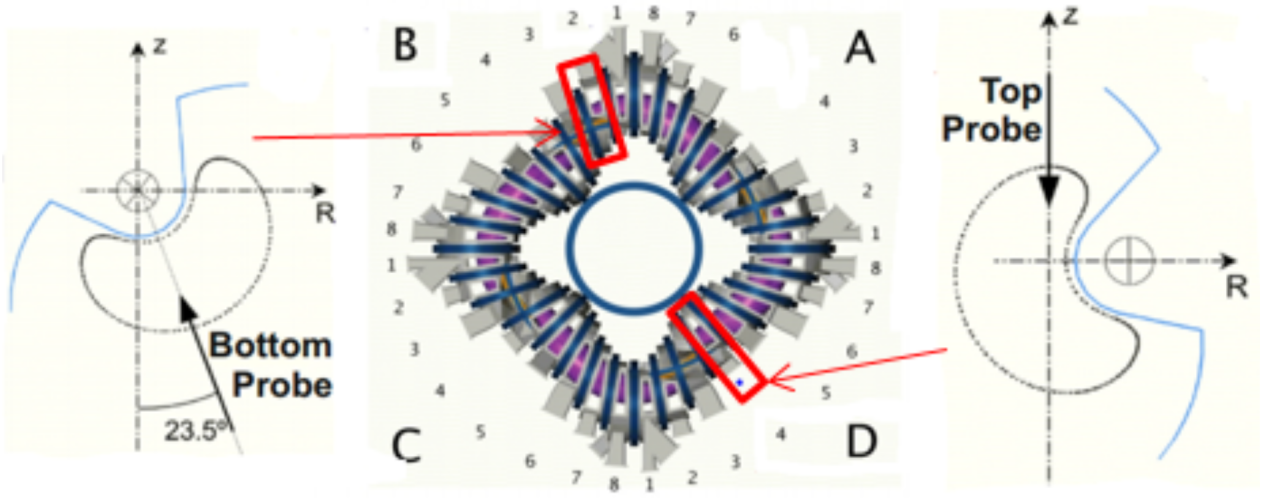

Figure 13: Locations of the probes in the Stellarator

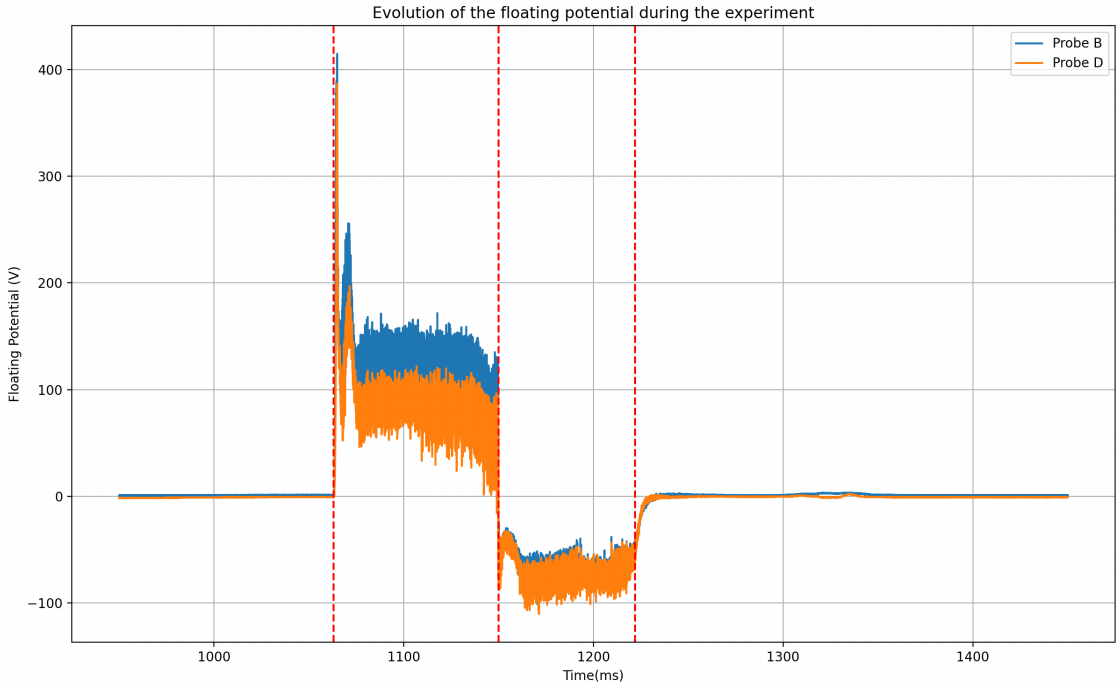

Figure 14: Presentation of the Data

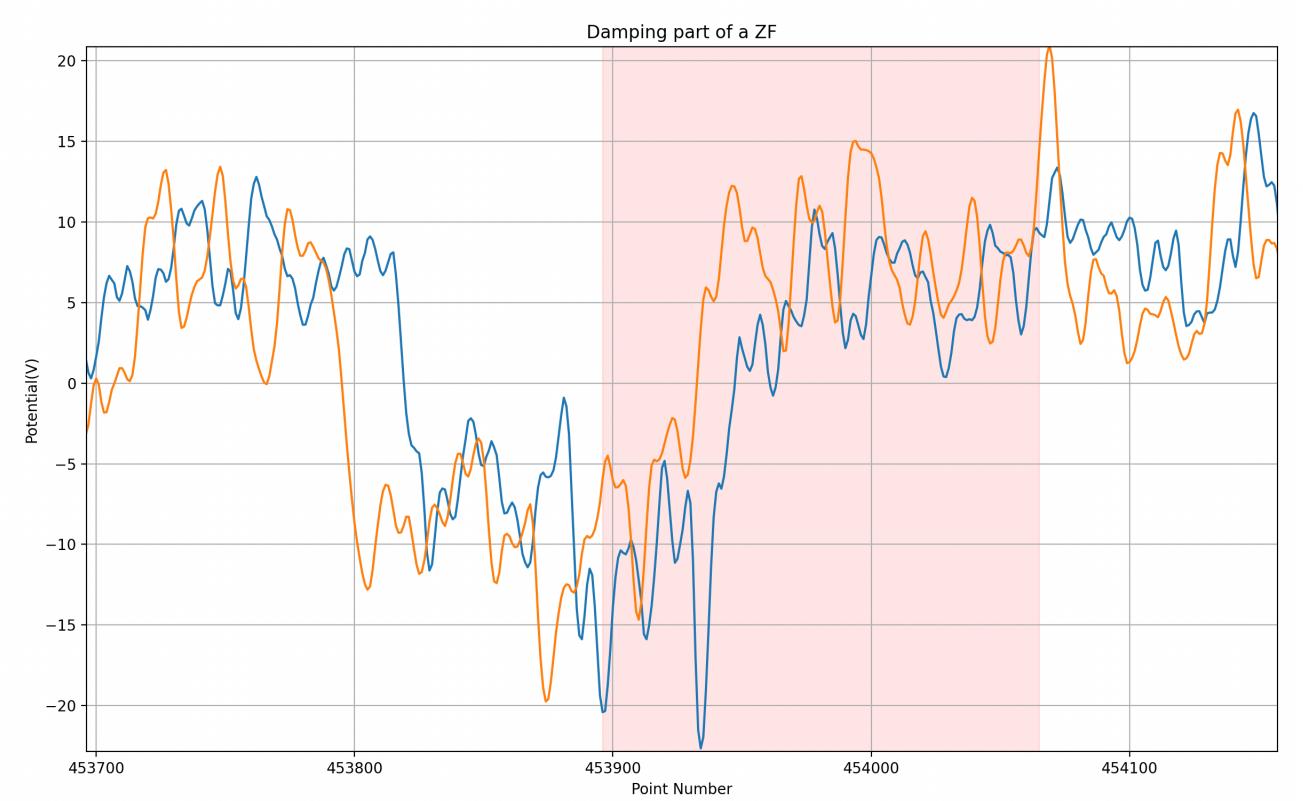

Figure 15: Example of a ZF that satisfy all criteria

5.1 The correlation between the signals

The events we are looking for in the plasma, which are the ZF, are *Global events*, in the sense that these events occur globally in a magnetic flux surface. As a result, these collective low frequency oscillations are characterized by a high level of correlation between the two floating potential on the same magnetic surface [Sán+13]. It will therefore be assumed that probes B and D are located at different position of the same magnetic surface. The correlation considered is the Long Range Correlation (LRC) between the two probes. This correlation study has been carried out both in the temporal and in the spectral domain.

5.2 The Exponential Decay

As explained, the relationship between ZF and turbulence can be described as a Predator-Prey model : the turbulence feeds the ZF by stretching due to the Reynolds stress forces. As a result, when the turbulence consumed, the driving Reynolds stresses vanish, and then floating potential in plasmas are subject to exponential decay due to magnetic viscous damping [Alo+12]. The floating potential V_E, the drive term $R(t)$ and the mag-

netic viscous damping term ν satisfy therefore the Equation 3.

$$\frac{\mathrm{d}}{\mathrm{d}t}V_{\mathrm{E}} = R(t) - \nu V_{\mathrm{E}} \tag{3}$$

It has been found empirically that the damping time of the exponential ν^{-1} is between 10 and 150 μs.

5.3 The RMS of the signal

Zonal flows, from which an example is given in **Figure 15** are characterized by peaks of the potential before the exponential decay. These peaks must necessary be in a distance from the mean value of the signal, that is larger than a given value of it. [1] This provides an additional condition for the detection of the ZF.

5.4 Goal of the internship

The 3 conditions presented in 5.1, 5.2 and 5.3 allow to detect ZF using the two temporal series of the potential measured by the probes. However, it would be more convenient for researchers to detect the ZF using just one probe instead of two. The aim of the internship is then to develop a machine learning model that would permit, given substantial amount of labeled data with the ZF intervals provided, to detect ZF using only one probe for ZF intervals detection. For that, a efficient method to detect ZF intervals given the data of 2 probes is needed. This data will therefore be used to train the machine learning model that will be developed in the second part of the internship. As highlighted in **Section 12**, a Neural Network (NN) has been used. The NN has been trained at first using simulated data (**Section 11**), which was generated thanks to a probabilistic model developed. The parameters of this probabilistic model will be chosen rightly according to the results obtained in the ZF detection of the real data. After the adjusting of weights by training of this simulated data, the performance of the NN will then be evaluated on the real data (**Section 12.3**). The summary of this strategy in given in **Figure 16**.

6 Strategy deployed for the detection of ZF given the signals of two probes

For a better visualisation of the data and a better comprehension of the strategy employed, the data presented in **Figure 14** was splitted into a smaller time lapse, which is presented in **Figure 17**.

The strategy will require different steps, that will be described in the next sections. Obviously, each step of the ZF detection require adjusting parameters. For a good fit of the parameters with the ZF pattern, some ZF intervals were labelled by hand. This process of parameters adjustment is detailed in **Section 9**.

[1] Usually, the RMS value is used

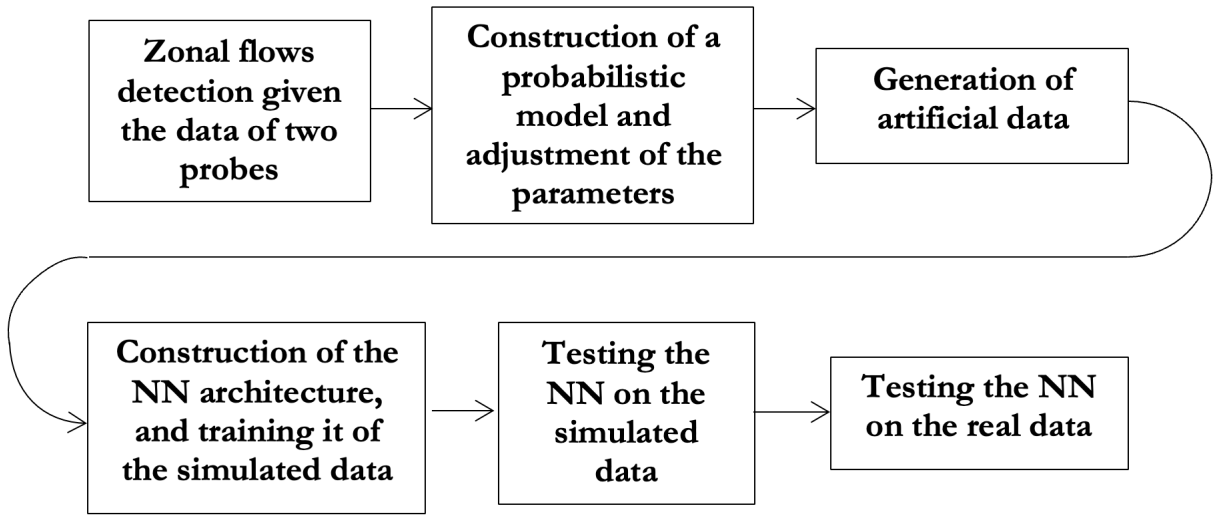

Figure 16: Strategy and steps

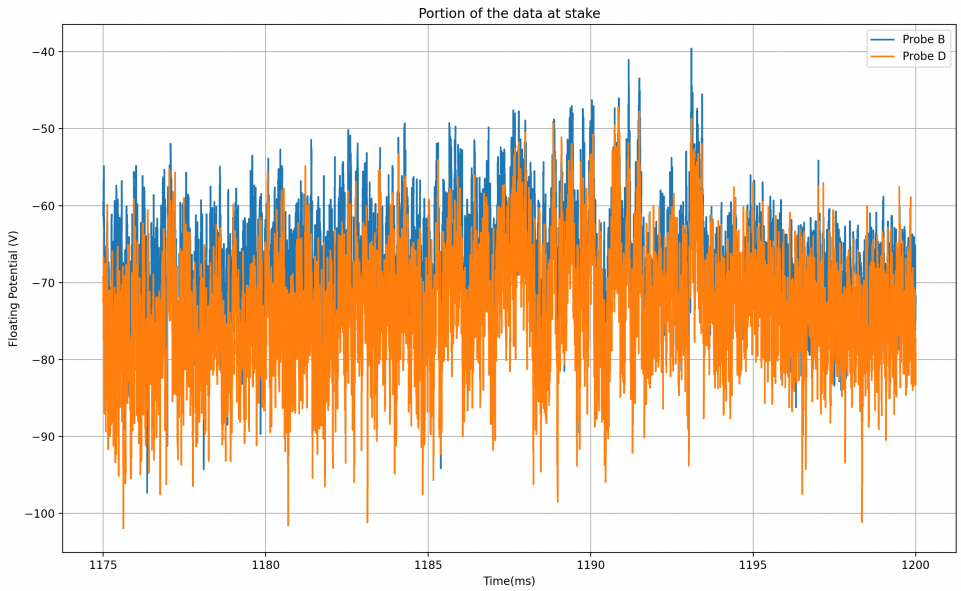

Figure 17: Portion of the Data used for the Strategy explanation

6.1 Smoothing the signal

For commodity reasons, the Data at stake, which is plotted in **Figure 17** is, that first, centered by Savitzky-Golay method. For that, a Savitzky–Golay filter [1]. A window length of about a tenth of the data length was used.

The effect of this filtration is plotted in **Figure 18**. After performing this filtration,

[1] In this instance, the Savitzky–Golay was used with a polynomial of order 2, see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html`

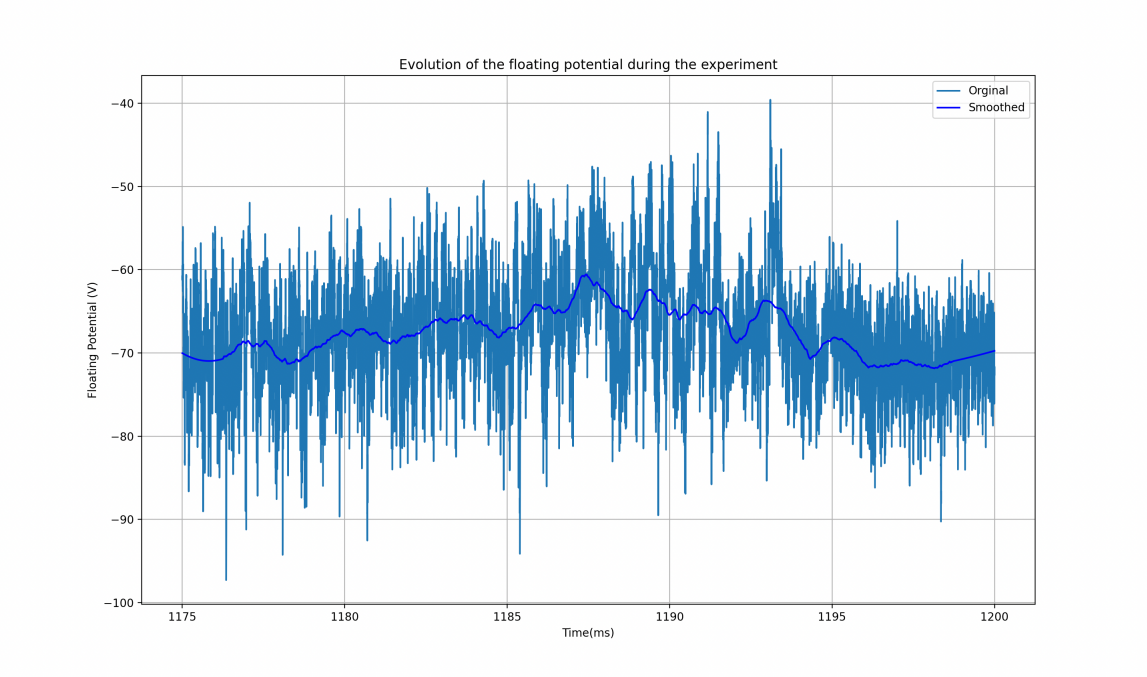

Figure 18: Effect of the smooth on the data of Probe B

the data obtained is plotted in **Figure 19**.

6.2 Spectral Coherence of the Signals

ZF are characterized by low-frequency coherent oscillations of the electrostatic potential. A natural method to detect low-frequency oscillations in a signal is the use of Fourier Transform. More precisely, because the data is not stationary, the result needed evolution of the spectrum of the signal *as a function of the time*. To perform this time frequency analysis, the STFT (Short-time Fourier Transform) has been used due to its Low Computational Complexity with regards to the other time frequency analysis method (see [Wik04b]).

The Short-time Fourier Transform of a signal $x(t)$ is defined as

$$\mathrm{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}dt$$

Where w in the Window function, in our case, a Gaussian Window function with a standard deviation σ_w of about $100\ \mu s$ has been used [1]. Therefore, the spectrogram of the both signals, which is defined as

$$\mathrm{spectrogram}\{x(t)\}(\tau,\omega) \equiv |X(\tau,\omega)|^2$$

can be calculated, using the Discrete-time STFT. Here, only the low frequency oscillations matters, and only the values of frequencies lower than $20\ kHz$ [2] will be considered

[1] See **Section 9** for the reason of the choice of this value
[2] See **Section 9**

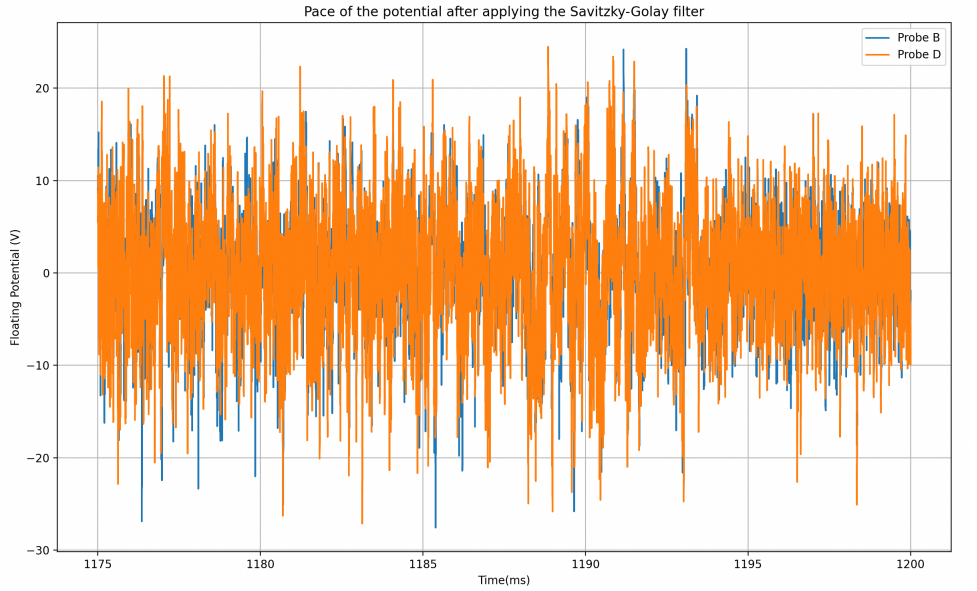

Figure 19: Centered data obtained after smoothing

in the spectrogram.

Having these two spectrograms, we need to get a measure of :

- The presence of low frequency oscillations in these signals
- The correlation between the low frequency oscillations of the signals of the two probes.

For achieving these two tasks simultaneously, it has been found that simply taking the dot product between the two vectors of values than constitute the spectogram at a given moment of time, provides satisfactory results with regard to the usual definition of the Squared coherency spectrum (see [PLZ19]). The results obtained are plotted in **Figure 20**. [1]

However, one should keep in mind that the area which are characterized by a significant Spectral Coherent are not necessary ZF. Others filtration which take account of the conditions 5.2 and 5.3 are necessary. Therefore, the process of selection of the time areas which are chosen for further investigations that take account of these other conditions is precised in the next section.

Choice of the time intervals at stake

The threshold value of the coherence coefficient ι in the considered time window for a point to be "at stake" using the process described in **Section 9**. [2]

[1] Brightness of the colors plotted are proportional to the values on the spectrogram
[2] See **Figure** 21

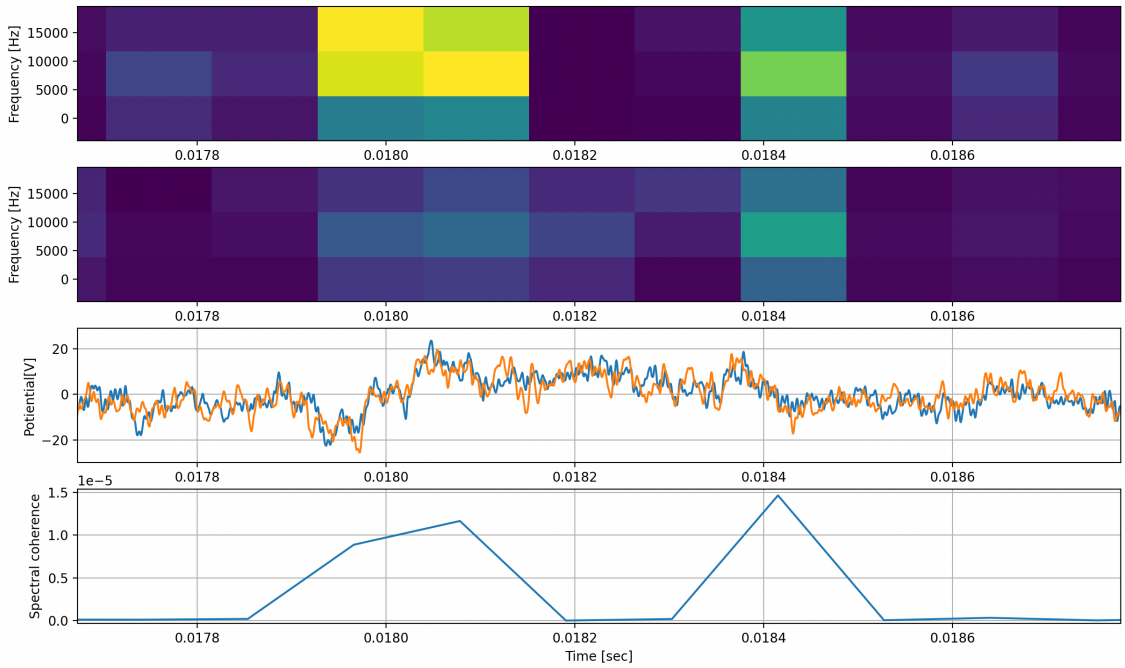

Figure 20: Spectrogram of both signals, and measure of their *spectral coherence* using the dot product of the spectograms vectors at each intervals of time

Once the values of times where the coherence coefficient is higher than ι are detected, we can perform the next filtration which is described in the next section.

6.3 Filtration based on the RMS of the Signal

For the next two steps of the examination, in which we take account of the conditions 5.2 and 5.3, we need to detect peaks in the signal, mainly to see if they match with the beginning of the last part of a ZF that is characterized by an exponential decay.

For that, we begin by finding the minimums and the maximums of the Signal along the whole period of time. Then, for each of these peaks, we check if they are in the selected time periods which where defined in paragraph 6.2. [1] If this is not the case, they are deleted from the list of peaks.

Then, we check, for each of these maximums of this updated list, whether or not their potential value is out of in the *Bollinger bands* of the signals, which are plotted in **Figure 22**. These bands spread around a moving average of a period of $S = 100$ points. [2] and the half gap between the bands corresponds to ρ times the rolling standard deviation taken with the same period. The value of ρ depends of the quality of ZF we are looking for, the value of chosen is $\rho \simeq 1.13$ (See **Section 9**).

[1] Or sufficiently close – at a distance smaller than 50 points (empirical value) – from these periods
[2] For the determination of this value of S, see **Section 9**

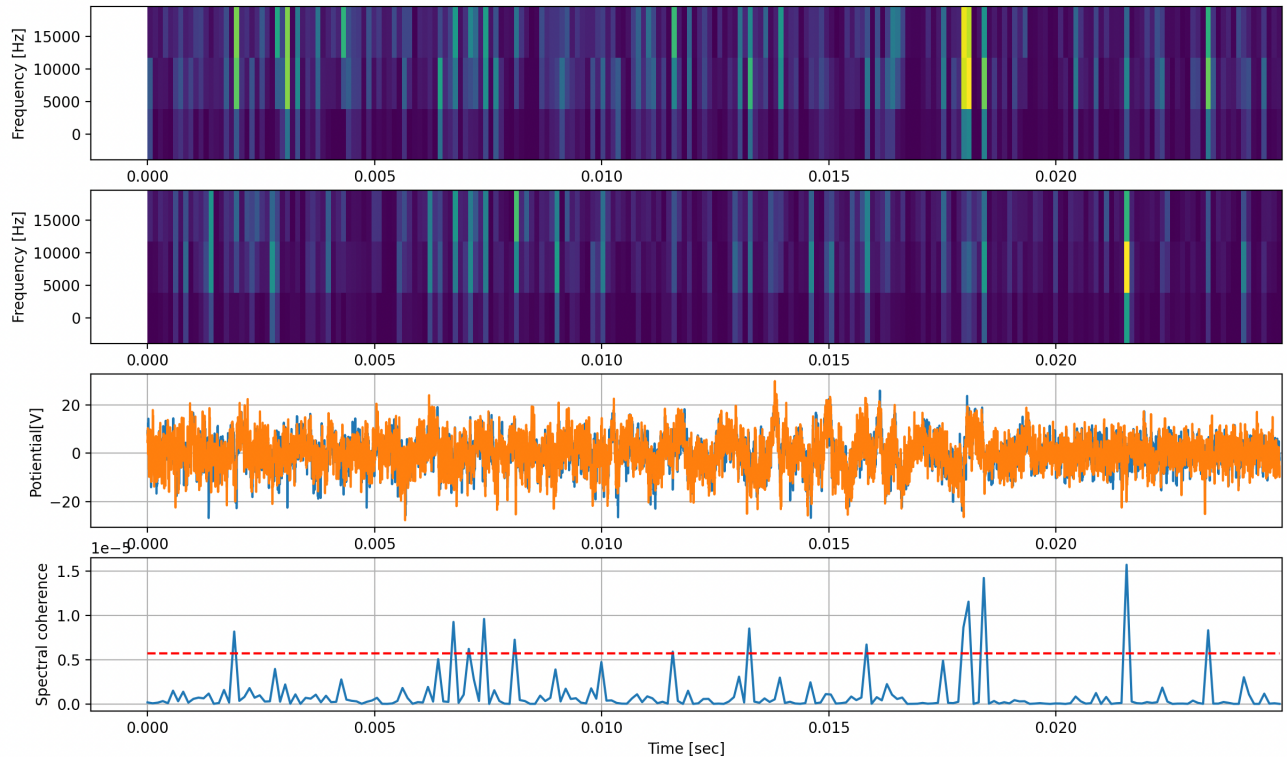

Figure 21: Threshold value for the coherence coefficient

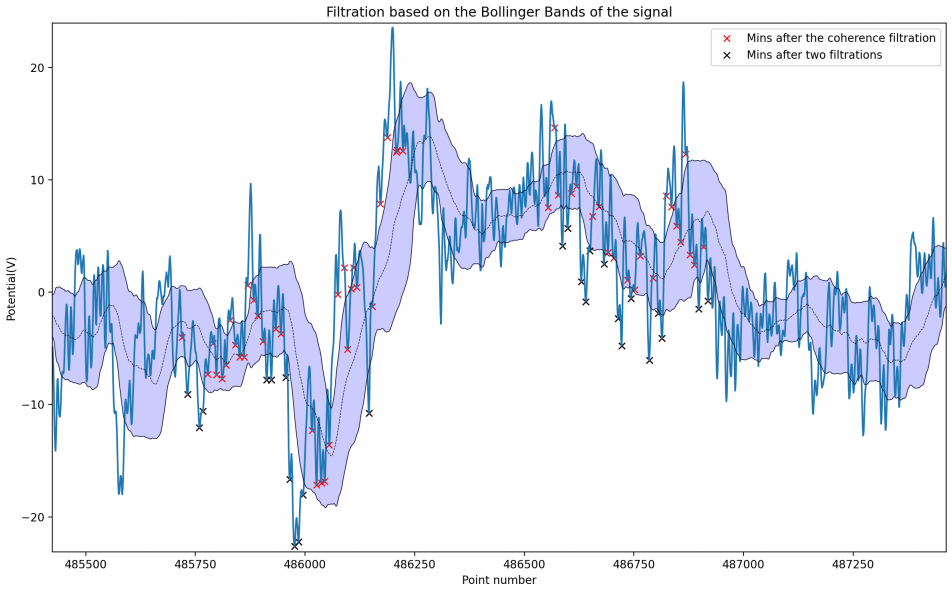

Figure 22: Example of the Bollinger filtration process applied to minimum peaks found in 6.2 section

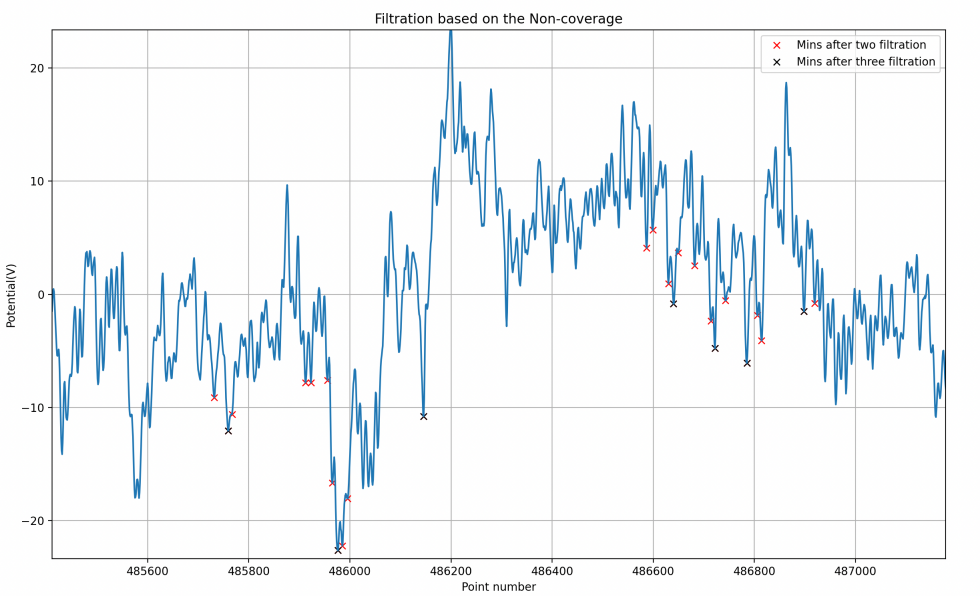

Figure 23: Filtration based on the non converge of the signals

6.4 The non-coverage of the ZF from the same data

After these two filtration on the data of one probe, we are now dealing with two lists of points, which are potentials peaks (minimums & maximums) of the signal that are likely to belong to ZF periods. However, many of these peaks may be part of the same ZF periods and the exponential fit test at the end of the ZF just need to be made once of each ZF period. As a result, a filtration has to be made to take account of this *non-coverage* of the periods. For that, we filter, for each of the two lists, the peaks which are in a distance closer from 60 points ; only the *best* indexes are kept [1]) (See **Section 9**) in the x-axis (see **Figure 23** for an example).

6.5 Exponential Fit

As highlighted in the introduction, the ZF are characterized by an exponential decay when the Reynolds stresses disappear. The goal of this section is then to check if the points fit with an exponential shape.

To be more precise, we have to determine, the each points x from the filtered list, if it exists period τ_x such as the function

$$t \in [x, x + \tau_x] \longmapsto a\exp(b(t - x)) + c$$

fits correctly with the shape of the potential along this same time period. This will be done using the `scipy.optimize.curve fit` function ; the parameters a, b and c chosen

[1] Among those whose distance is closer than this given value, the one with the minimum value is kept for the minimums, and the one with the maximum value when it comes to the minimums

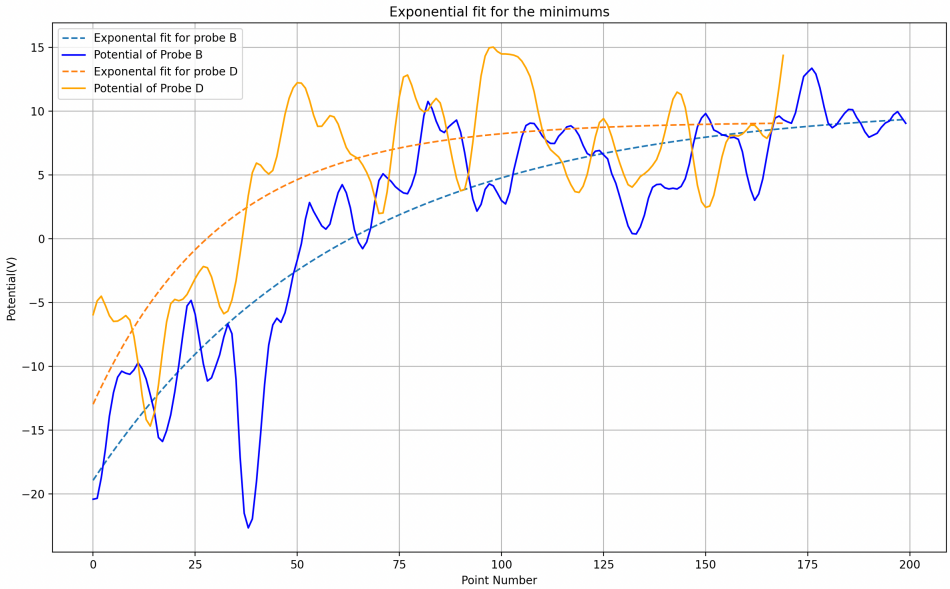

Figure 24: Exponential fits for the minimums

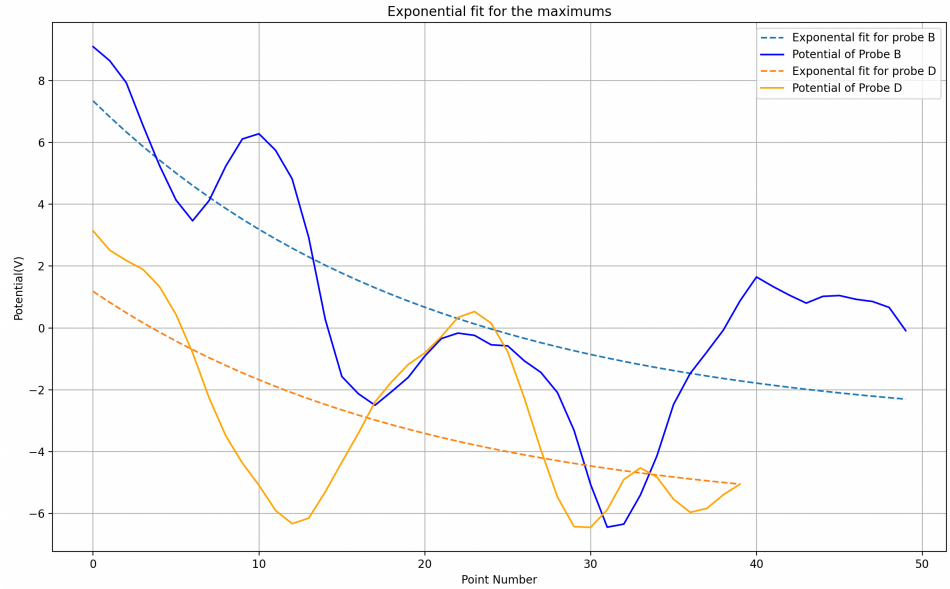

Figure 25: Exponential fits for the maximums

will be those which minimize the residual sum of squares (RSS) defined as

$$RSS = \sum_{i=1}^{n} \left(y_i - f\left(x_i\right)\right)^2$$

We will have $a > 0$ when testing the exponential decay with a maximum (see **Figure 25**), and $a < 0$ when testing with a minimum (see **Figure 24**). b will be negative in both cases, and taking account of the fact a the exponential decay has to be between 20 and 150 μs (see paragraph 5.2), in the resolution of the data chosen ($2\ MHz$), b has to be between $-1/300$ and $-1/20$ [1].

Determination of τ_x

As precised, the exponential decay time has to be between in a given interval of time. One possibility for the determination of τ_x is then to test, which value of $j \in [\![0, 20]\!]$, the smoothing with $\tau_x = 10 + 10j$ is the best.

Two criteria have been used for the measure of the quality of the smoothing. At first, the values of the errors on the coefficient a, b and c, and the value of the RMSE in the given period of time. The errors on the coefficients a, b and c can be defined as the variance of the coefficients given by the `scipy.optimize.curve fit`, normalized by the length of the interval of points considered : $n = \tau_x + 1$:

$$\begin{cases} a_{er} = \frac{\sqrt{\mathbb{V}(\hat{a})}}{n} \\ b_{er} = \frac{\sqrt{\mathbb{V}(\hat{b})}}{n} \\ c_{er} = \frac{\sqrt{\mathbb{V}(\hat{c})}}{n} \end{cases}$$

Then, given that the interval of variation of b is about 2 orders of magnitude below the one of the two other coefficients, it has been chosen to chose as criteria for the best value of τ_x the optimization of the quantity

$$\frac{a_{er} + 100 b_{er} + c_{er}}{102}$$

However, it has been found, that the RMSE should be also considered for the choice of value of τ_x, in order to prevent from too large value of τ_x where the exponential fit become not visual any more. Therefore, RMSE has also be included in the criteria, with a relative weight of about 0.1%.

After having determined τ_x for each points, one should find a criteria to reject the peaks for which the fit with an exponential shape is not satisfactory. After analysis of the labelled data, threshold values have been found for keeping or rejecting the corresponding points. [2]

[1] This is applicable, of course, if t does not refer to the time value, but to the point number

[2] See **Section 9**

After having applied these successive filters on the data of one probe, one gets the starting and finishing points of the DP associated to the data of one probe. However, this process of filtration needs to be applied to the data of the other probe too. We then get two subset of peaks $\mathcal{P}^1$ and $\mathcal{P}^2$ which are the peaks that should correspond to the beginnings and the ends of the DP of ZF for data 1 and data 2.

7 Two additional criteria about the coverage of the intervals found for both probes

The process described in the previous sections provides two subsets of peaks $\mathcal{P}^1$ and $\mathcal{P}^2$ described at the end of the previous section. Then, two other criteria have been chosen for enhance the quality of the detection :

Two intervals $((x_1, x_2), (y_1, y_2)) \in \mathcal{P}^1 \times \mathcal{P}^2$ are kept if :[1]

1. $$[x_1, x_2] \cap [y_1, y_2] \neq \varnothing$$

2. $$\frac{\mu([x_1, x_2] \cap [y_1, y_2])}{\mu([x_1, x_2] \cup [y_1, y_2])} \geq p$$

Where $\mu : B(\mathbb{R}) \to \mathbb{R}$ is the Lebesgue measure, and $p \in]0, 1[$ is threshold value of the coverage proportion which was determined using the hand-labelled data in **Section 9**. Physically, this criteria is related to the fact that exponential decays must concern both electrostatic potential measured by the two probes, which are part of the same magnetic surface.

8 Deduction of the ZF intervals

Let's summarize the filtration performed in previous paragraphs.

The notations are the following :

- $\mathcal{P}_1^i$, for $i \in \{1, 2\}$ are the subset of peaks from data i from time intervals *at stake* according to the **Section 6.2**.
- $\mathcal{P}_2^i$ is the subset of peaks from $\mathcal{P}_1^i$ that satisfy the RMS criteria from **Section 6.3**
- $\mathcal{P}_3^i$ is the subset of peaks from $\mathcal{P}_2^i$ that satisfy the non-coverage criteria from **Section 6.4**.
- $\mathcal{P}_4^i$ is the subset of peaks from $\mathcal{P}_3^i$ that satisfy the exponential fit from both probes (**Section 6.5**).

[1] The condition 2. implies the condition 1. but the condition 1. is checked before for computational costs reasons ; the condition 2. will then be checked only for intervals that satisfy the condition 1.

- $\mathcal{P}_5^i$ is the subset of peaks from $\mathcal{P}_4^i$ that satisfy the two coverage criteria from **Section 7**.

One could deduce the DP of ZF associated with the data i as

$$\mathcal{E}^i = \left\{ [x, x + \tau_x] \,,\ x \in \mathcal{P}_5^i \right\}$$

This definition of the ZF intervals was chosen for the rest of the internship. Although, according [Alo+12] and the equation 3 the ZF intervals are characterized by a growing period (when the drive term $R(t)$, which correspond to the Reynolds Stresses, increases), and by a damping period (which occurs when Reynolds Stresses vanishes).

9 Parameters adjustment

This section describes how to fit the values of the parameters mentioned in the previous sections. For that, statistics about the values of the parameters associated with the ZF intervals need to be done. Therefore, the method to label by hand data is detailed in **Section 9.1**. Then, the method to get the threshold values of the parameters is highlighted in **Section 9.2**.

9.1 Method to label by hand the data

The hand label of the data was performed in the intervals of values which was plotted in **Section 17**. For that, the Event handling and picking methods from the matplotlib library were used. [1]

The method is the following ; the user executes the **learning**() function that gets the plot of the signals of the two probes, with blues zones (See **Figure 26**) where the correlation is high enough. [2] As a matter of fact, the DP of the ZF are likely to occur in or just after these blues zones.

To label the data, the user has to double click of 6 points per ZF intervals, 3 points for each data set ; one point at the beginning of the GP, one point at the peak before the DP, and one point at the end of the DP. Automatically, the coordinates of the points where the double-click was added to a list z_{int}.

After performing this hand labeling, the list z_{int} is processed using the **treatment** function that, from the list of the z_{int} values, generates four lists $DPmin$, $DPmax$, $GPmin$ and $GPmax$, that are lists of the couples of values associated to the beginning and the end from each regime : The Growing Part (GP) and the Damping Part (DP). Otherwise, the suffixes min and max precise whether the DP is ascendant or descendant (See **Section 6.5**).

[1] See `https://matplotlib.org/stable/users/event_handling.html`

[2] If the process of labeling is done for the first time, i.e no ZF have been labelled yet, these blues zones will not appear, because in this case there is not yet references values for the coherence. Otherwise, the method for getting the threshold coefficient ι is given in **Section 9.2**.

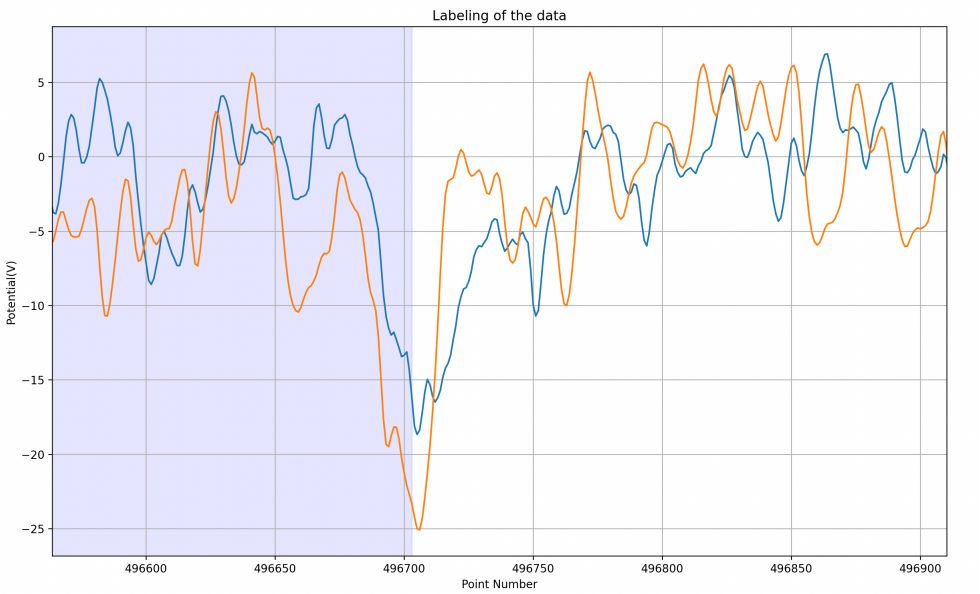

Figure 26: Labeling of the Data

Technical details about the treatment function

While doing these 6 clicks for each detection, the user has to double click to three points for each times series. The choice of the data 1 or 2 for the first three clicks doesn't matter as the function **treatment** automatically detect from which data the points chosen belong to. [1] If otherwise users made non voluntary double clicks on the graph, these points will not be taken account. Indeed, a 6-tuple $(x_1, x_2, x_3, x_4, x_5, x_6)$ from the z_{int} will be considered if $x_1 < x_2 < x_3$ and $x_4 < x_5 < x_6$, and if $|x_1 - x_4| \leq 100$

Using this method, 22 ZF intervals were labelled by hand between $X_1 = 450000$ and $X_2 = 500000$.

9.2 Optimisation of the parameters at stake

Getting this labelled data allows us to do statistics about the values of the parameters used in the different steps of the ZF detection.

To be more precise, the parameters to be adjusted are :

- The maximum value of the frequency that is considered for Coherence calculation $N \in \mathbb{N}$. [2]
- The standard deviation σ_w of the Gaussian window function w used in the STFT (See **Section 6.2**).

[1] To be more accurate, the function detects from which time series the point chosen is closer on the vertical axis

[2] One should write Nf where f is the resolution of the frequency in the resolution of the STFT frequency resolution

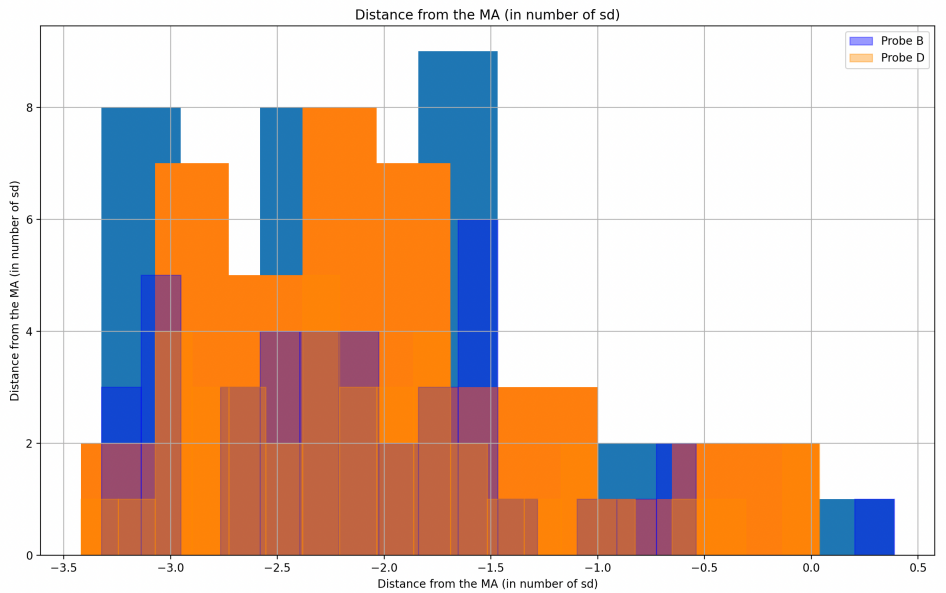

Figure 27: Histogram of the distances between the labelled peaks in the beginnings of the DP (ρ values), and the MA with $S = 100$

- The length of the windows S (in number of points) involved in the calculation of the moving average (MA), and the moving standard deviation (MSD) which is used to keep extremums away enough from the moving average value at a time moment of time (**Section 6.3**).
- The threshold value of the Coherence coefficient ι (Coherence in the sense of the definition given in **Section 6.2**).
- The optimal values ρ for the quotient between the distance of the peak with the MA and the MSD (**Section 6.3**).
- The threshold values on the errors on the exponential fitting (See **Section 6.5**) a^l_{eropt}, b^l_{eropt} and c^l_{eropt}. Where $l \in \{0, 1\}$ allows to distinguish between these optimal values in case of ascendant or descendant damping parts.
- The optimal value of the coverage proportion p_{opt}, whose definition is given in **Section 7**.

As an example, the histogram of the ρ values, with a given value S for the window length of the MA and the MSD is plotted in **Figure 27**. Then, an optimal value for the ρ coefficient that fits well this the data with a *keep rate* of α that be chosen to be the quantile of order α. For $\alpha = 90\%$ the optimal value for ρ is chosen to be the less strict between those of the two labelled data set

$$\rho_{opt} := \min\left(\rho^1_{opt}, \rho^2_{opt}\right)$$

This process for choosing optimal values of the parameters is the same for having ι, a^l_{eropt}, b^l_{eropt}, c^l_{eropt}, and p_{opt}. However, this process can be performed for these parameters

only if N, σ_w and S are known. Then, the next paragraph describes how to get correct values of these 3 parameters when it comes to the fit with the labelled data.

Values of N, σ_w and S chosen for the good fit with the labelled data

As a first method it has been chosen to generate a 3 dimensional grid for these three parameters, and to evaluate, for each point of the grid, what is the percentage of DP of ZF from the labelled data is kept. [1]

By designing the grid as

$$\{(N, \sigma_w, S), (N, \sigma_w, S) \in [\![3, 7]\!] \times \{50 + 10k, k \in [\![0, 6]\!]\} \times \{50 + 10k, k \in [\![0, 9]\!]\}\}$$

It has been found empirically that the values $N = 4$, $\sigma_w = 100$ and $S = 100$ provide the best results for a keeping percentage of 58% (execution time 5216.97 seconds).

10 Detection results

For measuring the efficiency of the detection algorithm, many metrics can be adopted.

At first, it has been chosen to use the F_1 score, which is an efficient tool for measuring a model's accuracy. The F_1 score is defined as the harmonic mean between the recall and the precision

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}$$

Where recall $= \frac{\text{tp}}{\text{tp}+\text{fn}}$ and precision $= \frac{\text{tp}}{\text{tp}+\text{fp}}$ [2]

Hence, a criteria has been designed to establish, whether or not a ZF has been detected. If we denote $(\mathscr{Z}_k^{\text{real}})_{k\in I}$ the real positions of ZF, and $(\mathscr{Z}_k^{\text{detect}})_{k\in J}$ the positions of the detected ZF, we can, in first approximation suppose that, for $k \in I$ [3]

$$\mathscr{Z}_k^{\text{real}} \text{ has been detected} \Leftrightarrow \exists\, l \in J,\ \mathscr{Z}_k^{\text{real}} \cap \mathscr{Z}_l^{\text{detected}} \neq \varnothing$$

Thus, the measure of the performance of the algorithm provides the result which is function of the *keep rate* α mentioned in paragraph 9.2 (See **Figure 28**).

The following paces of the recall and the precision are coherent ; with lower values of α, only the *best* ZF are kept in the detection, and consequently, the precision, which measure the quality of the result is better. Likewise, higher values of α correspond to a larger amount of ZF kept, and a better recall, which measure the quantity of positive

[1] The calculus has also been performed with a random grid, and similar results were obtained. For an example of parameters optimization using a random grid, see **Section 23**

[2] NB : $\text{t}, \text{f}, \text{p}, \text{n}$ are abbreviations for true, false, positive and negative

[3] It is otherwise possible to use a recoverage criterion as in paragraph 7

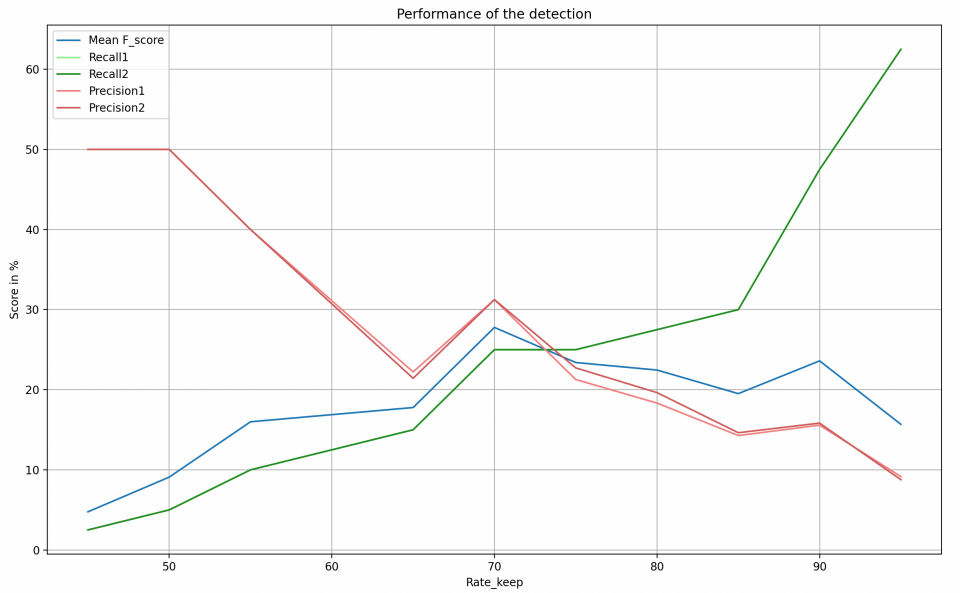

Figure 28: Detection scores for the two probes against α

values chosen.

It has been thus chosen to choose $\alpha \simeq 70\%$, as an optimal keep rate.

11 Simulation of Data

The simulation of the probe data for training the deep learning model mentioned in **Section 5.4** is useful for several reasons. At first, because the amount of data provided by the probe is limited, and the ZF detection carried out in the previous sections is not perfect. Furthermore, the construction of the NN for ZF classification needs several steps and therefore, it is more convenient to test it on *simplified* artificial data at first (see **The First probabilistic model** below), before sophisticating the model to make it matching with the real data.

As mentioned in **Section 5.4**, the parameters of the model will be adjusted thanks to the results of the ZF detection which was carried out in the previous section.

11.1 First probabilistic model

This section describes the first probabilistic model which was chosen for data generation.

Let's denote $(S_t)_{t\geq 0}$ the stochastic process associated with the signal, and $(\mathscr{Z}_k)_{k\in I}$ the ZF intervals, where $I \subset \mathbb{N}$. For each zone $\mathscr{Z}_k$, let's define $t_k := \min(\mathscr{Z}_k)$ which is the

time appearance of the ZF. The process of the construction of the $(\mathscr{Z}_k)$ is the following

$$\begin{cases} t_1 \sim \mathcal{G}(p) \\ \forall k > 1,\ t_{k+1} \sim t_k + h + \mathcal{G}(p) \end{cases} \tag{4}$$

Where

- $\mathcal{G}(p)$ stands for a geometric law of parameter p.
- $h := \mu(\mathscr{Z}_k)$ is the length of the ZF intervals, which has been chosen to be constant at first.

Let's then define $\mathscr{Z}$ as [1]

$$\mathscr{Z} = \bigcup_{k \in I} \mathscr{Z}_k$$

And $(S_t)_{t \geq 0}$ as

$$S_t = \sigma_0 \varepsilon_t \left(1 - \mathbf{1}_{\mathscr{Z}}(t)\right) + \sigma_1 \varepsilon_t \mathbf{1}_{\mathscr{Z}}(t) + \sum_{k \in I} a_k e^{-b_k (t - t_k)} \mathbf{1}_{\mathscr{Z}_k}(t) \tag{5}$$

Where $\varepsilon_t \sim \mathcal{N}(0,1)$ and (a_k), (b_k) are parameters to adjust. At first, it has been chosen to take (a_k) and (b_k) constants, denoted as a and b, and $\sigma_1 = \sigma_0$.

Therefore the parameters to adjust in this model are p, a, b, σ_0 and h. The process of adjustment of these parameters using the ZF detector from last section is given in the Annex in **Section 19**.

Then it has been chosen at first to take :[2]

$$\begin{cases} p = 0.0009 \\ h = 50 \\ a = -14 \\ b = 0.04 \\ \sigma_0 = 5.866 \end{cases}$$

The simulated data obtained using this first model is plotted in **Figure 29**.

However as highlighted in **Figure 30** The fit with real data is not realistic, mainly because the simulated presents abrupt and too fast variations. For that, a second probabilistic model, which is based on this first one has been designed in paragraph 11.2.

11.2 Second probabilistic model

For a better match between the simulated and the real data, the simulated data needs to be smoothed, and its variations slowed down. For that, a moving average is an efficient

[1] This union is expected to be disjoint as the ZF intervals do not recover

[2] From this point on, only those ZFs that start with a minimum were considered.

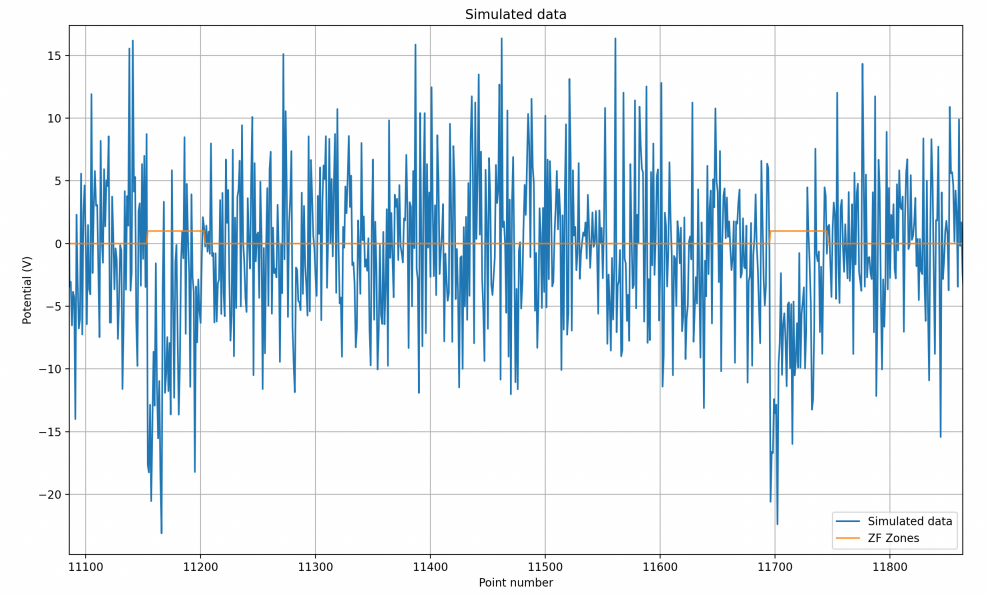

Figure 29: Example of simulated data with the first probabilistic model

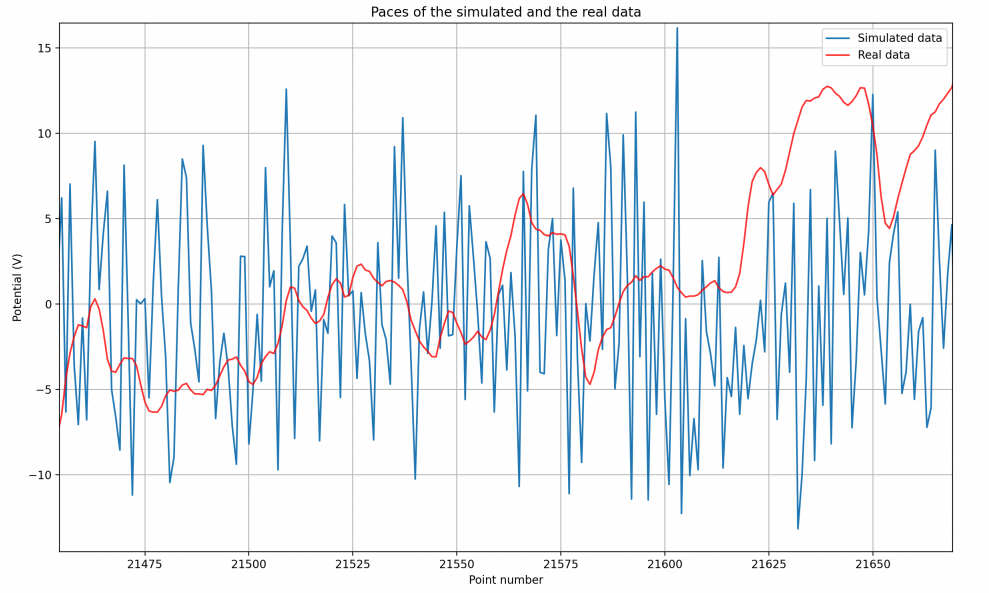

Figure 30: Closer look on the simulated and the real data

method, provided that ZF intervals added after the averaging. [1]

More precisely, keeping the same notation as in the previous section, the second probabilistic model is the following :

1. At first, let's define (S_t) with the following equation

$$S_t = \sigma_0 \varepsilon_t$$

2. Then, let's apply a moving average of the last $N = 100$ values of S_t, to define $\overline{S}_t$ such as

$$\overline{S}_t = \frac{1}{N} \sum_{l=0}^{N-1} S_{t-l}$$

3. In order to keep the same range of variation for the signals S_t, $\overline{S}_t$ is multiplied by the scale factor $\frac{\sigma(S_t)}{\sigma(\overline{S}_t)}$

$$\tilde{S}_t = \frac{\sigma(S_t)}{\sigma(\overline{S}_t)} \overline{S}_t$$

4. Then, to smooth locally the signal a Savitzky-Golay filter is applied on $\tilde{S}_t$, with a window length of $l = 7$ points, and a polynomial degree $d = 2$

5. Therefore, another Savitzky-Golay filter is applied on $\tilde{S}_t$, with a much larger window length ($l = len(data)/10 + 1$), as this filter was also applied on the real data (see **Section 6.1**).

6. After that, the ZF intervals are then added. Keeping the same notations as in the previous section, we then got a signal X_t defined as

$$X_t = \tilde{S}_t + \sum_{k \in I} a_k e^{-b_k(t-t_k)} \mathbf{1}_{\mathscr{Z}_k}(t)$$

At first, the same values of the parameters p, a, b, σ_0 and h were chosen. The pace of the simulated signal (X_t), compared with the real data is given in **Figure 31**.

11.3 Improvement of the model

In order to obtain a more realistic model, it has been decided to choose a distribution for the parameters h, a, and b, which are not constant ; their value is specific to each Zonal Flow.

[1] if not, the exponential decays are widened and not correspond any more to their shape in the real data

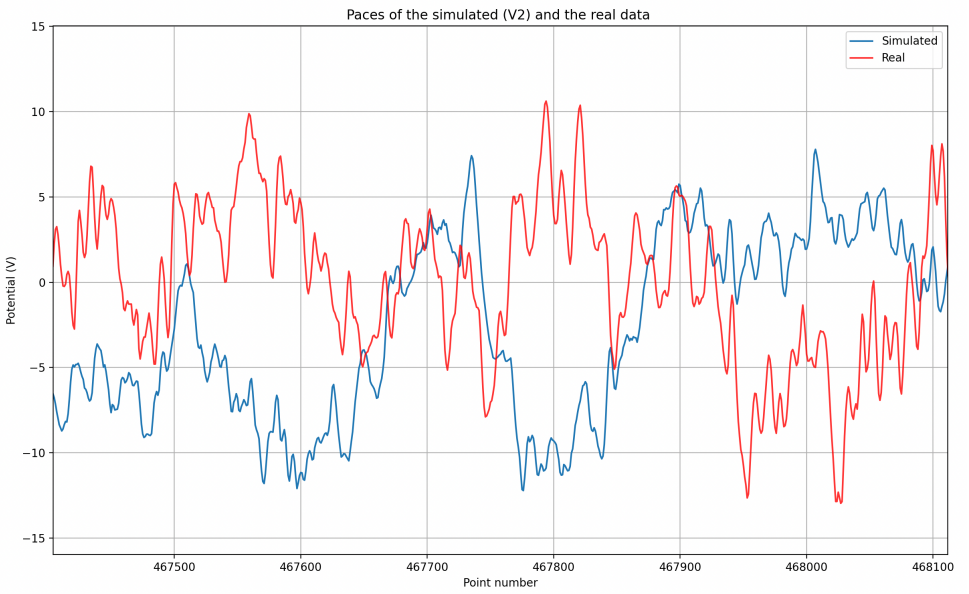

Figure 31: Close look on the simulated (V2) and the real data

The distribution of the h [1], a, b in ZF intervals of the real data are given in **Figure 32**. [2] Then, it has been chosen to match these histograms with the distribution of a Gamma law $\Gamma(\alpha, \beta)$, for which the expression is given in Equation 6.

$$f(x;\alpha,\beta) = x^{\alpha-1}\frac{\beta^{\alpha}\mathrm{e}^{-\beta x}}{\Gamma(\alpha)}\mathbf{1}_{\mathbb{R}_+^*}(x) \tag{6}$$

Where $\Gamma : z \mapsto \int_0^{+\infty} t^{z-1}\mathrm{e}^{-t}\,\mathrm{d}t$ is the Gamma function.

Using the method of moments [3], it can be shown that given the realizations $y_1, y_2, ..., y_n$ of a random variable Y, a good approximation of the parameters α, and β can be obtained by the following expressions

$$\tilde{\alpha} = \frac{\left(\bar{Y}_n\right)^2}{s_Y^2}, \quad \tilde{\beta} = \frac{\bar{Y}_n}{s_Y^2}$$

Where $s_Y^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{Y}_n\right)^2$ is the empirical variance and $\bar{Y}_n$ is the empirical mean.

Using these expressions, the values chosen are

$$\begin{cases} (\alpha_h, \beta_h) = (5.399, 0.052) \\ (\alpha_a, \beta_a) = (5.111, 0.361) \\ (\alpha_b, \beta_b) = (8.296, 222.735) \end{cases}$$

[1] As precised before, h, which is now a random variable, provides the length of ZF interval.
[2] These distributions were obtained using the Detection Model presented in **Section 6**
[3] See `https://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-spring-2015/lecture-notes/MIT18_443S15_LEC3.pdf`

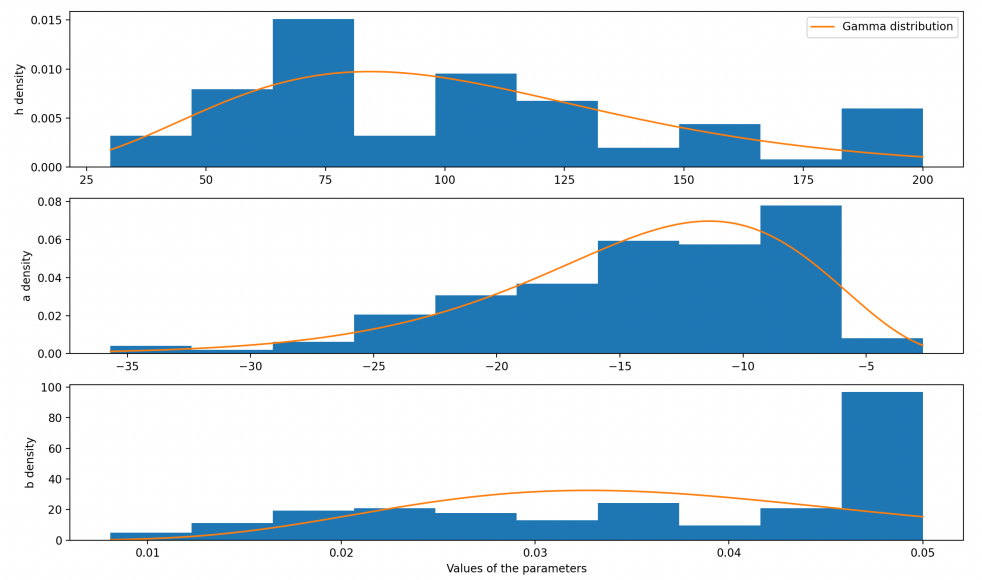

Figure 32: Fit with the Gamma distribution for h, a, and b values

Thus, the distributions of the parameters $\Gamma(\alpha_i, \beta_i)$ chosen are plotted in **Figure 32**. [1]

Using this model improved, 10^7 points of labelled data were generated, with about ~ 500000 points generated for testing the model, which is presented in the next section.

12 Neural network model for ZF recognition

Deep Neural Networks (NN) have seen very successful applications in the last years. Due to their performance in time series classification, they thus have been chosen to perform automatic ZF detection.

12.1 Choice of architecture and hyperparameter tuning

For this section, it will be supposed that the reader has a basic knowledge of NN architectures. If not, the article [Faw+19] can be read.

The inputs and outputs

The first step for the NN construction is the choice in the inputs and outputs.

Let's note S_{t_i} the value of the potential at the point number $i : t_i = i\Delta t$. [2] An input

[1] The density of the Gamma law in defined in $\mathbb{R}_+^*$ but the distribution can also represent negative values by taking the symmetric of the distribution with respect to the ordinate axis. Indeed, the parameter a, that takes negative values in follow the distribution $-\Gamma(\alpha_a, \beta_a)$

[2] In our case, given the resolution of the data, $\Delta t = 10^{-6}\ s$

of the neural network has been chosen to be a window of length ℓ

$$\mathbf{X}^{(i)} := \left(S_{t_i}, S_{t_{i+1}}, ..., S_{t_{i+\ell}}\right)$$

The corresponding image of $\mathbf{X}^{(i)}$ by the network is therefore

$$\mathbf{Y}^{(i)} := (\mathbf{1}_{\mathscr{Z}}(t_i), \mathbf{1}_{\mathscr{Z}}(t_{i+1}), ..., \mathbf{1}_{\mathscr{Z}}(t_{i+\ell}))$$

In this instance a rolling window of length $\ell = 100$ has been chosen, as it is the typical size of a ZF interval.

We're facing a *multi-label classification problem*. The aim of the neural network is, given sufficient amount of labelled data $(\mathbf{X}^{(i)}, \mathbf{Y}^{(i)})_{i\in I}$ provided, to construct a function

$$f : \begin{cases} \mathbb{R}^\ell \to [0,1]^\ell \\ \mathbf{X} \to \mathbf{Y} = f(\mathbf{X}) \end{cases}$$

By adjusting the weights (ω_k) corresponding to each layer l, that minimize a Loss function $\mathscr{L}$ that has been chosen in the next section.

Thus, considering a point time t_k, the probability that t_k belongs to a ZF, $\mathbb{P}(k \in \mathscr{Z})$, can be deduced, in a different process for each type of window [1] chosen.

A first possibility for the choice of the training data is to choose non overlapping successive windows of length ℓ, i.e $I = \{0, \ell, 2\ell, ...\} = \ell\mathbb{N}$. This solution is preferable in term of computation cost, and in this case, the probability $\mathbb{P}(k \in \mathscr{Z})$ can be deduced by considering the rest r and the quotient q of the euclidean division of k by ℓ, and the r^{th} component of $f(\mathbf{X}^{(q)})$.

However, it was deduced that such a window leads to anomaly in ZF detection, as only portions part of them are correctly detected when the ZF interval straddles two drive vectors (see **Figure 34**). It has thus been chosen to use overlapping windows, i.e $I = \{0, \frac{\ell}{n}, \frac{2\ell}{n}, ...\} = \frac{\ell}{n}\mathbb{N}$ where n is such as $\frac{\ell}{n} \in \mathbb{N}$. [2] In this case, we can deduce $\mathbb{P}(k \in \mathscr{Z})$ by

$$\mathbb{P}(k \in \mathscr{Z}) = \max_{\begin{cases} \underline{i} + \underline{m} - 1 = k \\ \underline{m} \in [\![1, \ell]\!] \\ \underline{i} \in I \end{cases}} (f(\mathbf{X}^{(i)}))_m$$

Where $(f(\mathbf{X}^{(i)}))_m$ designs the m^{th} component of $f(\mathbf{X}^{(i)})$. In other words, for a time t_k that correspond to a point number k, the value chosen for the probability that k belongs to a ZF has been chosen to be maximum of the probabilities that refer to the different overlapping windows containing this point. Another possibility considered was to choose the mean instead of the max. It has been shown that for low values of n among the dividers of ℓ, the performance of max outperforms those of mean, and the opposite for high values for n. In this context, the max function has been chosen given that the limited performance of the computer used in the internship did not allow to use high values of n that generates obviously a very large amount of data.

[1] overlapping or not

[2] For $\ell = 100$, the possible values for n are $\{1, 2, 4, 5, 10, 20, 25, 50, 100\}$

Choice of architecture

This sections precises the architecture of the NN, and also the reasons that lead to this choice of architecture.

Given the inputs and outputs mentioned in the last section, the hyper parameters of the neural network to be tuned are :

1. The number of layers L
2. The number of neurons in each layers l ; $n^{[l]}$
3. The activation functions of each layers l ; $f^{[l]}$
4. The initialization method for the weights of each layers.
5. The choice of the loss function $\mathscr{L}$ that will me minimized
6. The choice of the metric function $\mathscr{M}$ to judge the performance of the model.
7. The choice of the optimizer of the loss function.
8. The adjusting parameters of the optimizer chosen.
9. The dropout value of each layers d.
10. The batch size b.
11. The number of epochs e to consider.

As precised before, the simulated data was splitted into a validation and a training part. After passing through each batch, the weights are updated to minimize the loss function $\mathscr{L}$. Otherwise, the model is evaluated using the metric function $\mathscr{M}$ after each epoch (passes of the entire training dataset).

On that account, the tf.keras.callbacks.EarlyStopping function has been used to choose automatically the number of epochs e to consider before the metric function $\mathscr{M}$ applied to training set (See **Figure 33**) begin to increase instead of decreasing [1] ; this stop occurs when the model begins to overfit with the training data. Therefore, the number of epochs e don't need to be tuned as it is automatically optimized.

Some of these parameters can be directly deduced given the formalization of the problem.

[1] To be more accurate, this function takes also a Patience parameters p thanks to which the metric $\mathscr{M}$ continues to be evaluated after the stopping signal. This can be useful when of the increase loss function on the training set is temporary and begin to decrease again.

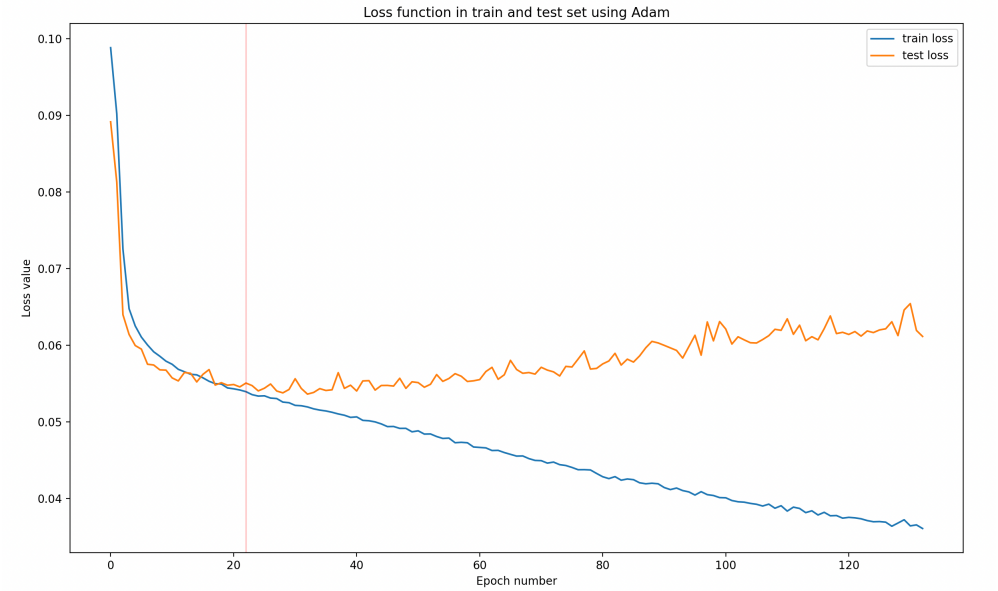

Figure 33: Automatic choice of the number of epoch (see the red line)

To begin with, as the problem involves an output vector in $[0,1]^{\ell}$, the activation function of the last layer $f^{[L]}$ will naturally be chosen to be a sigmoid function σ. [1]

Otherwise, using a random grid for hyper parameters tuning (See Annex 23) the following parameters were chosen :

1. $L = 4$
2. $(n^{[1]}, n^{[2]}) = (128, 256)$
3. $(f^{[1]}, f^{[2]}) = (\text{selu}, \text{selu})$[2]
4. initialization method of each layers : lecun normal
5. Loss function $\mathscr{L}$ used : RMS error
6. Metric function $\mathscr{M}$ used : RMS error
7. Optimizer of the loss function chosen : Adam
8. Adjusting parameters of the optimizer chosen : $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}, \text{decay} = 0.0$

[1] Which is defined as

$$\sigma : x \in \mathbb{R} \mapsto \frac{1}{1+e^{-x}} \in [0,1]$$

[2] The selu function is

$$x \in \mathbb{R} \mapsto \lambda \begin{cases} \alpha\left(e^{x}-1\right) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

With parameters $(\lambda, \alpha) = (1.0507, 1.67326)$

9. $d = 0.01$
10. $b = 256$

Otherwise, for computation cost and performance reasons, the overlapping parameter $n = 10$ with window sizes of $h = 100$ were chosen.

12.2 Results with the simulated data

The visual overview of the results of the ZF detection with the Architecture presented in the last section in the same point number, and with different overlapping parameters is given in **Figure 34**. As highlighted in **Figure 34**, the highest values of n provide a better detection of the ZF interval ; a compromise has to be found between the quality of ZF detection, and a satisfactory computation cost. For that, the value $n = 5$ has been chosen.

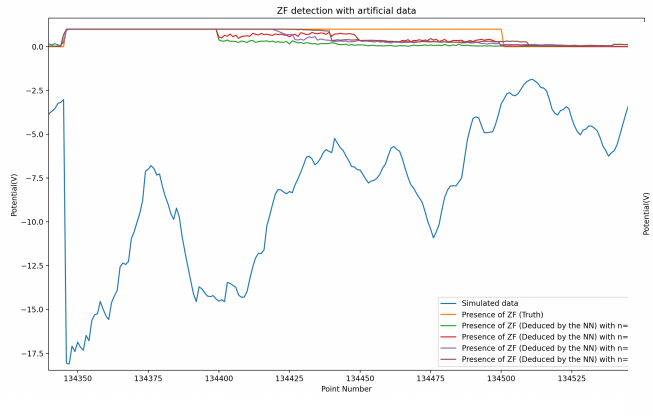

Figure 34: Performance for different n values

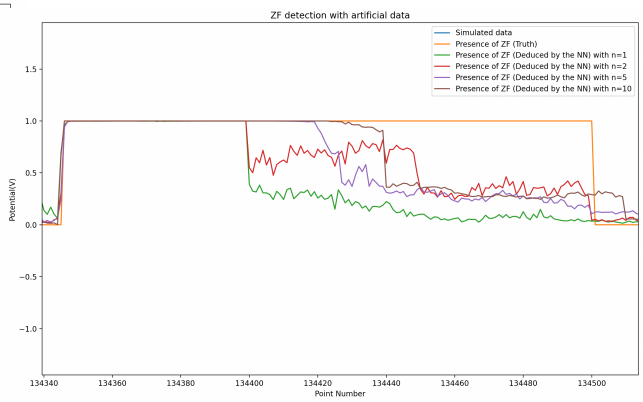

Figure 35: Zoom on the Figure

For this value of n, another pace of the result of the NN is given in **Figure 36**. Visually, the results are already very satisfactory. The value of the RMS error on the test set (with $\simeq 460000$ lines of data) can be directly be read of the **Figure 33**, and is about $5.5\ \%$.

12.3 Results with the real data

The results of the model [1] the real data, although slightly less good than on the simulated data, are still satisfactory. Applying the model on the real data with $h = 100$ and $n = 5$, and by filtering the ZF intervals taking only account of the values ≥ 0.999. [2] The pace of the results is given in **Figure 37**. Another filter has therefore been applied, to remove the detected ZF intervals whose length is lower than a threshold value of 30 points (See **Figure 38**).

[1] Which was trained on the simulated data

[2] By omitting this last filter, the NN tends to overreact ; it detects ZF too frequently

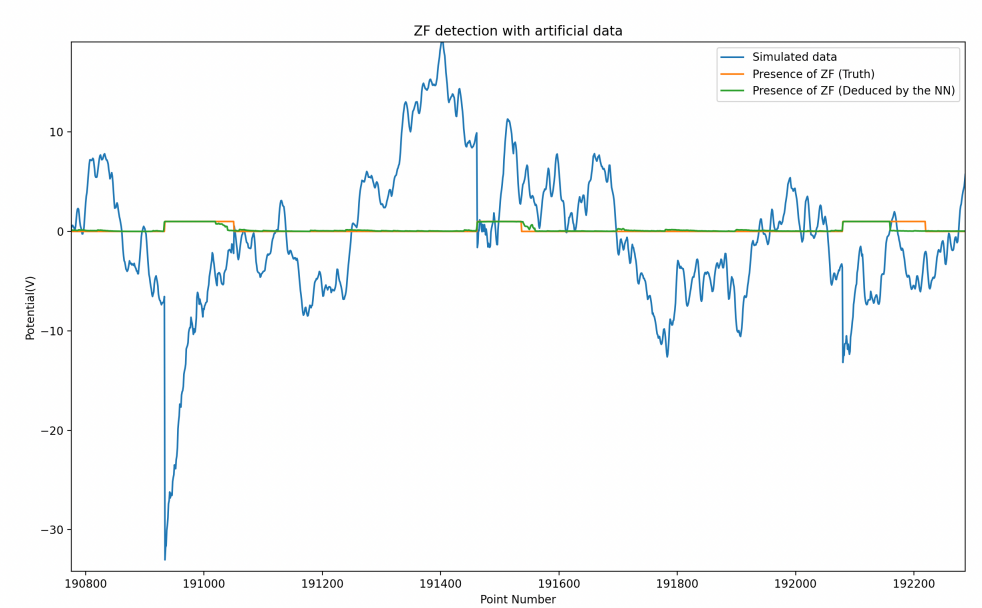

Figure 36: Result of the NN applied to the simulated data

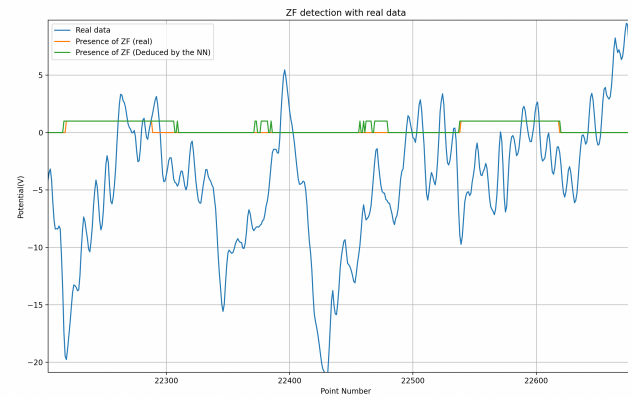

Figure 37: Detection of ZF on the real data

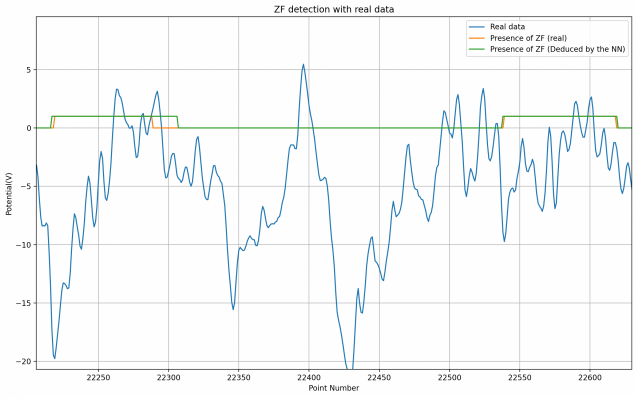

Figure 38: After applying the second filter

When it comes to the performance of this new detection, it has been found that the RMSE increased almost fivefold (reaching $\simeq 25\ \%$) with regards to the results obtained with simulated data.

13 Conclusions

To sum up, the ZF detection was carried out in several steps.

1. Some ZF were labelled by hand using the `Event handling` method from matplotlib.
2. The ZF were detected using the time series of electrostatic potential of two Langmuir Probes, using the three criteria which are : The high correlation between the signals, the exponential decay, the distance between the peak and the MSD (Moving Standard Deviation) of the signal. The adjustment of the parameters to optimize the detection has been performed by doing statistics on the ZF labelled by hand in the first step.
3. The previous detection algorithm has provided statistics on the ZF characteristics, which allowed us to design a probabilistic model that matches on the real data pace.
4. The probabilistic model allowed to generate massive amount of labelled data. A NN model was then designed to detect these events, trained and tested in the data generated by the probabilistic model.
5. The NN model was tested on the real data, and its performance was evaluated.

The generation of the probabilistic model was designed for two main reasons : at first, because the amount real data provided was not sufficient to train properly a NN model, and also because the method for labelling data is not $100\ \%$ efficient. The probabilistic model allows to overcome these drawbacks. However, the training on a simulated data presents the risk of an overfit with the simulated pace of the data, which is follow not perfectly the same dynamic as the real data. Therefore, the results on the real data can be deteriorated.

Thus, new ways can be explored for boosting the performance of the model. First, a possibility for improving the NN could be to use Generative Adversarial Networks (GANs) ; they use the loss function of 'real data' versus 'fake' to push simulated data closer to real data. [1] This could possibly be the next step for improving the results obtained in this 3-month-intership.

Another possibility of improvement could be, while keeping the same format for the inputs and outputs of the NN, to obtain a biggest amount of real data [2], and to train the model on the real data without using the probabilistic model.

Otherwise, the architecture of the NN, which was presented in **Section 12.1** can also be enhanced. Indeed, the hyperparameters tuning a very long and costly process, and the computer which was used in this internship (MacBook Pro M1, 2020, 8 Go of RAM) was not adapted for these costly and long tests.

[1] See `https://towardsdatascience.com/learning-from-simulated-data-ff4be63ac89c`

[2] What was not possible during my internship

Bibliography

[FK95] Uriel Frisch and Nikolaevich Kolmogorov Andre. *Turbulence: the legacy of AN Kolmogorov*. Cambridge university press, 1995.

[Wik04a] Wikipedia contributors. *Plagiarism — Wikipedia, The Free Encyclopedia*. 2004. URL: https://en.wikipedia.org/wiki/Pinch_(plasma_physics).

[Wik04b] Wikipedia contributors. *Plagiarism — Wikipedia, The Free Encyclopedia*. 2004. URL: https://en.wikipedia.org/wiki/Time%E2%80%93frequency_analysis.

[Dia+05] Patrick H Diamond et al. "Zonal flows in plasma—a review". In: *Plasma Physics and Controlled Fusion* 47.5 (2005), R35.

[Alo+12] J A Alonso et al. "Dynamics of zonal-flow-like structures in the edge of the TJ-II stellarator". In: *Plasma Physics and Controlled Fusion* 55.1 (Dec. 2012), p. 014001. ISSN: 1361-6587. DOI: 10.1088/0741-3335/55/1/014001. URL: http://dx.doi.org/10.1088/0741-3335/55/1/014001.

[Sán+13] J. Sánchez et al. "Dynamics of flows and confinement in the TJ-II stellarator". In: *Nuclear Fusion* 53.10 (Sept. 2013), p. 104016. DOI: 10.1088/0029-5515/53/10/104016. URL: https://doi.org/10.1088/0029-5515/53/10/104016.

[Med17] Anna Medvedeva. "Experimental study of turbulence at the plasma edge of ASDEX Upgrade tokamak with an ultra-fast swept reflectometer". PhD thesis. Université de Lorraine; Technische Universität München, 2017.

[Faw+19] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data mining and knowledge discovery* 33.4 (2019), pp. 917–963.

[PLZ19] Gaoyuan Pan, Shunming Li, and Yanqi Zhu. "A Time–Frequency Correlation Analysis Method of Time Series Decomposition Derived from Synchrosqueezed S Transform". In: *Applied Sciences* 9.4 (2019). ISSN: 2076-3417. DOI: 10.3390/app9040777. URL: https://www.mdpi.com/2076-3417/9/4/777.

[Sin20] Sukhmander Singh. "Dynamics of Rayleigh-Taylor Instability in Plasma Fluids". In: *Fluid-Structure Interaction*. IntechOpen, 2020.

14 Annex: Technical configuration used

The device used for performing this internship is a MacBook Pro M1, 2020. The programming languages used were Python and R Studio. The versions of Python used were Python 3.8.0 (for running TensorFlow) and 3.9.0 (for running the other libraries, including scipy).

The libraries used were : Tensorflow (2.4.0), numpy (1.21.1), scipy (1.6.0), pandas (1.3.0), matplotlib.pyplot, stockstats, nitime, math, random, eqsig, statsmodels, time, mpl toolkits, warnings, copy, os.

15 Annex: Code for the management of the correlation between the signals

The management of the spectral and temporal correlation between the signals has been performed using mainly the three following functions. The value of seuil opt which is the ι coefficient (See **Section 6.2**), is obtained by the hand-labeling method, which is presented in the next section.

```
def Correlation(D,D2,R=50) : #D and D2 are the two data frames
 #R is the value of the rolling average for the time correlation calculus

#This function returns the points numbers which are considered to
#be highly correlated (in the temporal domain) in 'Potential(V)'
#component of the two data frames D and D2


    Corr=D['Potential(V)'].rolling(R).corr(D2['Potential(V)'])

    #First filtration
    t=(Corr>0.90).astype(int)

    return(t)

def iscorr(e,t,M=100) : #This function returns, for a point e
#whether it is correlated given the result of the previous function,
#and a tolerance gap value M
    for i in range (e-M,e+M) :
        if i>=x1 and i<x2 :
            if t[i] :
                return(1) #1 TRUE
    return(0)

def Correlation_s(D,D2,seuil_opt,x1,x2,N=4,F=100) : #seuil_opt
```

```
#stands for the optimal value of the cross spectral product, for a
#point to be considered as 'high spectral correlation point' : this
#value was determined by the hand-labelling of the data

    Fs=2e6
    f, t, Sxx = signal.spectrogram(D['Potential(V)'], Fs, window=('gaussian',F))
    f=f[0:N]
    Sxx=Sxx[0:N,:]

    f2, t2, Sxx2 = signal.spectrogram(D2['Potential(V)'], Fs,window=('gaussian',F))
    f2=f2[0:N]
    Sxx2=Sxx2[0:N,:]

    T=len(t)

    SXX=[Sxx[:,t] for t in range (0,T)]
    SYY=[Sxx2[:,t] for t in range (0,T)]

    Corr=[0]*T

    for j in range (T) :
        Corr[j]=np.dot(SXX[j],SYY[j])

    seuil=np.mean(Corr)

    pas=int((x2-x1)/(T))
    indices=np.arange(x1,x2,pas)
    Corr2=pd.DataFrame([0]*len(D['Potential(V)']),index=D.index)

    for j in range (len(indices)-2) :
        Corr2[0][indices[j]]=Corr[j]*10**(9)

    for k in range (x1,x2) :
        if Corr2[0][k]==0 :
            Corr2[0][k]=np.nan

    Corr2=Corr2.interpolate(method='linear') #To obtain the right
    #number of points, a linear interpolation has been performed

    #First filtration
    t=(Corr2>seuil_opt).astype(int)
    t=t[0]

    return(t)
```

16 Annex: Non for non coverage of ZF

This Annex consists of the three functions that allow the non-coverage of ZF presented in **Section 6.4**.

```
def diff(l) : #This function provides the list of the differences
#between the values of the list l
    L=[]
    for i in range (1,len(l)) :
        L.append(l[i]-l[i-1])
    return(L)

def count(Diff,S=60) : #This function counts the number of values
#of Diff whose value is lower than S, a threshold value
    N=0
    for e in Diff :
        if e<=S :
            N+=1
    return(N)

def update(i_3,MAX,data,S=60) : #i_3" is a list of indexes
#if MAX=1 if MAX, -1 if MIN
#S is the thresold value from the precedent function

#This function filters the list of indexes i_3, to prevent the
#coverage of ZF ; among the group of indexes which are too close to
#each other, only the best are kept (the minimums or maximums,
#depending on the times of points specified by the variable MAX)
    i_4=copy.deepcopy(i_3)
    Diff=diff(i_4)
    To_del=[]

    while count(Diff,S)>0 :

        To_del=[]

        Diff=diff(i_4)
        for i in range (len(Diff)) :

            if Diff[i]<=S :

                Liste=[data['Potential(V)'][i_4[i+1]],data['Potential(V)'][i_4[i]]]

                if MAX==-1 :

                    k=Liste.index(max(Liste))
```

```
                    To_del.append(i_4[i+(1-k)])

                elif MAX==1 :

                    k=Liste.index(min(Liste))
                    To_del.append(i_4[i+(1-k)])

        for p in To_del :
            if p in i_4 :
                i_4.remove(p)

        Diff=diff(i_4)

    return(i_4)
```

17 Annex: Optimisation of the detection algorithm by hand-labelling

The following function were created for hand-labelling of the data, and for optimizing the parameters of the ZF detection algorithm.

```
def split(dat,x1,x2) : #This function splits the data frame
    l=len(dat)
    dat1=dat.tail(l-int(x1))
    dat1=dat1.head(int(x2-x1))
    return(dat1)

def onclick(event): #For a point to be considered, the user will
#have to double-click on it
    if event.dblclick :
            zi.append((event.xdata,event.ydata))

def learning() : #This function allows to label by hand data, by
#clicking on the temporal time series. The relevant values are then
#stored in a variable called 'zi'

    Dt=split(data1,X1,X2)
    D2t=split(data2,X1,X2)

    def smooth(D,D2,x1,x2) : #This function allows to smooth the
    #signal using savgol_filter

        S_1=savgol_filter(D['Potential(V)'],int(len(D)/10)+1, 2)
        S_2=savgol_filter(D2['Potential(V)'],int(len(D2)/10)+1, 2)
```

```
        D_s=pd.DataFrame.copy(D)
        D2_s=pd.DataFrame.copy(D2)

        for i in range (x1,x2) :
            D_s['Potential(V)'][i]=S_1[i-x1]
            D2_s['Potential(V)'][i]=S_2[i-x1]

        for i in range (x1,x2) :
            D['Potential(V)'][i]=D['Potential(V)'][i]-D_s['Potential(V)'][i]
            D2['Potential(V)'][i]=D2['Potential(V)'][i]-D2_s['Potential(V)'][i]

        return((D,D2))

    Da=smooth(Dt,D2t,X1,X2)
    D=Da[0]
    D2=Da[1]

    t=Correlation_s(D,D2,seuil_opt,X1,X2) #Spectral correlation btw the 2 signals

    def fill2(t,c1='blue') : #This function fills the zones with
    #high correlation, it facilitates the labelling in the right
    #intervals
        i=0
        while i<len(t) :
            if t[x1+i]==1 :
                X=[]
                while t[x1+i]==1 :
                    X.append(x1+i)
                    i+=1
                plt.fill_between(X,-30 ,30, color=c1, alpha=0.1)
            i+=1

    fig, ax = plt.subplots(1,1)

    fill2(t)

    ax.plot(Dt['Potential(V)'])
    ax.plot(D2t['Potential(V)'])

    ax.grid()

    cid = fig.canvas.mpl_connect('button_press_event', onclick)
    plt.show()

def traitement(zi) : #This function supposes that empty lists
#called  GP2_min, DP2_min, GP_min, DP_min, GP2_max, DP2_max, GP_max
```

```
#and DP_max were created. It allows to fulfill these lists using the
#zi variable which was created by hand labelling

    Dt=split(data1,X1,X2)
    D2t=split(data2,X1,X2)

    i=0

    while i<len(zi)-5 :

        if zi[i+2][0]>zi[i+1][0] and zi[i+1][0]>zi[i][0] and zi[i+4][0]>zi[i+3][0] and

            if abs(zi[i][0]-zi[i+3][0])<100 : #Idem

                a0=[abs(D2t['Potential(V)'][int(zi[i][0])]-zi[i][0]),abs(D2t['Potentia
                b0=[abs(Dt['Potential(V)'][int(zi[i][0])]-zi[i][0]),abs(Dt['Potential

                if min(b0)<min(a0) :
                    zi[i],zi[i+1],zi[i+2],zi[i+3],zi[i+4],zi[i+5]=zi[i+3],zi[i+4],zi[i

                else :
                    None

                a0=[abs(D2t['Potential(V)'][int(zi[i][0])]-zi[i][0]),abs(D2t['Potentia
                b0=[abs(Dt['Potential(V)'][int(zi[i][0])]-zi[i][0]),abs(Dt['Potential

                a1=[abs(D2t['Potential(V)'][int(zi[i+1][0])]-zi[i+1][0]),abs(D2t['Pote
                b1=[abs(Dt['Potential(V)'][int(zi[i+1][0])]-zi[i+1][0]),abs(Dt['Potent

                a2=[abs(D2t['Potential(V)'][int(zi[i+2][0])]-zi[i+2][0]),abs(D2t['Pote
                b2=[abs(Dt['Potential(V)'][int(zi[i+2][0])]-zi[i+2][0]),abs(Dt['Potent

                a3=[abs(D2t['Potential(V)'][int(zi[i+3][0])]-zi[i+3][0]),abs(D2t['Pote
                b3=[abs(Dt['Potential(V)'][int(zi[i+3][0])]-zi[i+3][0]),abs(Dt['Potent

                a4=[abs(D2t['Potential(V)'][int(zi[i+4][0])]-zi[i+4][0]),abs(D2t['Pote
                b4=[abs(Dt['Potential(V)'][int(zi[i+4][0])]-zi[i+4][0]),abs(Dt['Potent

                a5=[abs(D2t['Potential(V)'][int(zi[i+5][0])]-zi[i+5][0]),abs(D2t['Pote
                b5=[abs(Dt['Potential(V)'][int(zi[i+5][0])]-zi[i+5][0]),abs(Dt['Potent

                n0=int(zi[i][0])+a0.index(min(a0))
                n1=int(zi[i+1][0])+a1.index(min(a1))
                n2=int(zi[i+2][0])+a2.index(min(a2))

                n3=int(zi[i+3][0])+a3.index(min(a3))
```

```
                n4=int(zi[i+4][0])+a4.index(min(a4))
                n5=int(zi[i+5][0])+a5.index(min(a5))

                if zi[i+1][1]<zi[i][1] :

                    GP2_min.append((n0,n1))
                    DP2_min.append((n1,n2))
                    GP_min.append((n3,n4))
                    DP_min.append((n4,n5))

                elif zi[i+1][1]>zi[i][1] :

                    GP2_max.append((n0,n1))
                    DP2_max.append((n1,n2))
                    GP_max.append((n3,n4))
                    DP_max.append((n4,n5))

                print(i)
        i+=1

def fill(Int_min,Int_max,c1='blue',c2='red') : #This function
#enables to fulfill the alleged ZF intervals with colors

    for i in range (0,len(Int_max)) :
        (a,b)=Int_max[i][0],Int_max[i][1]
        X=np.arange(a,b,1)
        ax.fill_between(X,-30 ,30, color=c1, alpha=0.1)

    for i in range (0,len(Int_min)) :
        (a,b)=Int_min[i][0],Int_min[i][1]
        X=np.arange(a,b,1)
        ax.fill_between(X,-30 ,30, color=c2, alpha=0.1)

def plot_mouse() : #This function allows to add new points on zi
#that are not yet chosen

    fig, ax = plt.subplots()
    fill(DP_min,DP_max)
    fill(DP2_min,DP2_max,'green','orange')
    fill(GP_min,DP_max)
    fill(GP2_min,GP2_max,'green','orange')
    cid = fig.canvas.mpl_connect('button_press_event', onclick)
    ax.grid()

    ax.plot(D['Potential(V)'])
    ax.plot(D2['Potential(V)'])
```

```
    plt.title('Labelled ZF')
    plt.show()

def Coh_opt(DP,DP2,Rate_keep=95,N=4,F=100,X1=450000,X2=500000) :
#DP and DP2 are lists of couples, that correspond to ZF intervals
#for both probes.
#Rate_keep is a parameter that adjusts the iota coefficient in
#accordance with the quality of the ZF chosen for the detection
#N adjusts the maximum frequency to be considered
#F is the size of the gaussian window chosen

#This function returns the optimal value of the coherence : the iota coefficient.

    Dt=split(data1,X1,X2)
    D2t=split(data2,X1,X2)

    Fs=2e6
    f, t, Sxx = signal.spectrogram(Dt['Potential(V)'], Fs, window=('gaussian',F))
    f=f[0:N]
    Sxx=Sxx[0:N,:]

    f2, t2, Sxx2 = signal.spectrogram(D2t['Potential(V)'], Fs,window=('gaussian',F))
    f2=f2[0:3]
    Sxx2=Sxx2[0:N,:]

    T=len(t)

    SXX=[Sxx[:,t] for t in range (0,T)]
    SYY=[Sxx2[:,t] for t in range (0,T)]

    Corr=[0]*T

    for j in range (T) :

        Corr[j]=np.dot(SXX[j],SYY[j])

    pas=int((X2-X1)/(T))
    indices=np.arange(X1,X2,pas)
    Corr2=pd.DataFrame([0]*len(Dt['Potential(V)']),index=Dt.index)

    for j in range (len(indices)-1) :
        Corr2[0][indices[j]]=Corr[j]*10**(9)

    for l in range (X1,X2) :
        if Corr2[0][l]==0 :
            Corr2[0][l]=np.nan
```

```
    Corr2=Corr2.interpolate(method='linear')
    Corr2=Corr2[0]

    Liste_coh=[]
    Liste_coh2=[]

    for e in DP :
        e=e[0]
        Liste_coh.append(Corr2[e])

    for e in DP2 :
        e=e[0]
        Liste_coh2.append(Corr2[e])

    Seuil1=np.percentile(Liste_coh,100-Rate_keep)
    Seuil2=np.percentile(Liste_coh2,100-Rate_keep)

    Seuil_opt=min(Seuil1,Seuil2)

    return(Seuil_opt)

def valeur_opt(DP,DP2,MAX,Rate_keep=80) : #MAX=1 if DP and DP2
#contains damping intervals that begin with a maximum value, and -1
#else
#Rate_keep : adjusting parameter that refers to the rate of 'real
#ZF' to be kept

#This function provides the optimal values for the a, b and c
#coefficient for the ZF detection algorithm

    Dt=split(data1,X1,X2)
    D2t=split(data2,X1,X2)

    C=[]
    A_e=[]
    B_e=[]
    C_e=[]
    T=[]

    for e in DP :

        if MAX==1 :
            res=score_max(e[0],Dt)
        else :
            res=score_min(e[0],Dt)
```

```
        score=res[0]
        s=res[1]
        T.append(10+10*s)
        C.append(score)

    for e in DP2 :

        if MAX==1 :
            res=score_max(e[0],D2t)
        else :
            res=score_min(e[0],D2t)

        score=res[0]

        T.append(10+10*s)
        C.append(score)

    for elt in C :
        A_e.append(elt[0])
        B_e.append(elt[1])
        C_e.append(elt[2])

    am=np.median(A_e)
    bm=np.median(B_e)
    cm=np.median(C_e)

    a_eropt=np.percentile(A_e, Rate_keep)
    b_eropt=np.percentile(B_e, Rate_keep)
    c_eropt=np.percentile(C_e, Rate_keep)

    plt.hist(T)
    tm=np.median(T) #100

    return((a_eropt,b_eropt,c_eropt))

#a_eropt_min,b_eropt_min,c_eropt_min=valeur_opt(DP_min,DP2_min,-1)
#a_eropt_max,b_eropt_max,c_eropt_max=valeur_opt(DP_max,DP2_max,1)

### Distance with the average ?

###For the mins
def Distances_opt(DP,DP2,Rate_keep=90,MAX=-1,S=100) : #This
#function provides the optimal distance with the average when a ZF
#appears
```

```
    S=100
    MAX=-1
    Rate_keep=90
    DP=DP_min
    DP2=DP2_min

    Dt=split(data1,X1,X2)
    D2t=split(data2,X1,X2)

    Distances1=[]
    Distances2=[]
    D_m=Dt.rolling(S).mean()['Potential(V)']
    D2_m=D2t.rolling(S).mean()['Potential(V)']
    D_sd=Dt.rolling(S).std()['Potential(V)']
    D2_sd=D2t.rolling(S).std()['Potential(V)']
    D_rms=np.sqrt(Dt.pow(2).rolling(S).apply(lambda x: np.sqrt(x.mean())))['Potential
    D2_rms=np.sqrt(D2t.pow(2).rolling(S).apply(lambda x: np.sqrt(x.mean())))['Potenti

    for elt in DP :
        Distances1.append((Dt['Potential(V)'][elt[0]]-D_m[elt[0]])/D_sd[elt[0]])

    for elt in DP2 :
        Distances2.append((D2t['Potential(V)'][elt[0]]-D2_m[elt[0]])/D2_sd[elt[0]])

    if MAX==-1 :

        Distance_opt=max(abs(np.percentile(Distances1, Rate_keep)),abs(np.percentile(D

    elif MAX==1 :

        Distance_opt=max(abs(np.percentile(Distances1, 100-Rate_keep)),abs(np.percenti

    return(Distance_opt)


def plotidist() : #This function plots the distogram of distances
#with the average (in number of sd)

    plt.hist(Distances1)
    plt.hist(Distances2)
    plt.grid()
    plt.hist(Distances1,bins=20,color='blue',ec='blue',alpha=0.4,label='Probe B')
    plt.hist(Distances2,bins=20,color='darkorange',ec='darkorange',alpha=0.4,label='Pr
    plt.xlabel('Distance from the MA (in number of sd)')
    plt.ylabel('Distance from the MA (in number of sd)')
    plt.legend()
```

```
    plt.title('Histogram of the distance from the MA (in number of sd)')
    plt.show()

def liste_quotient(Int,Int2) : #This Function provides the list of
#length quotients between the two intervals that recover

    Int=DP_min
    Int2=DP2_min

    ind_=indicator(Int)
    ind2_=indicator(Int2)

    product=ind_*ind2_

    Lp=[]

    for (a,b) in Int :
        for (c,d) in Int2 :
            if (a<c and b>c) or (a<c and d<b) or (c<a and d>a) or (c<a and b<d) :
            #The two intervals recover

                liste=[product[e] for e in range (min(a,c),max(b,d))]

                n=len(liste)
                liste=np.asarray(liste)
                p=len(np.where(liste==1)[0])/n

                Lp.append(p)

    return(Lp)
```

18 Annex: ZF detection algorithm

```
import os

print("Current Working Directory " , os.getcwd())
os.chdir("/Users/victorletzelter/Desktop/Projet_python")

#Files that contains the useful functions
exec(open('imports_gestion.py').read()) #The importation of modules
exec(open('Coherence_gestion.py').read()) #The functions related to the correlation
exec(open('Non_coverage.py').read()) #The functions related to the non coverage
exec(open('Exp_gestion.py').read()) #The functions related to the exponential fit

#exec(open('Synthese.py').read())
```

```
os.chdir('/Users/victorletzelter/Desktop/Dat')

data1=pd.read_csv("/Users/victorletzelter/Desktop/Dat/converted_data1.txt",delimiter=
data2=pd.read_csv("/Users/victorletzelter/Desktop/Dat/converted_data2.txt",delimiter=

x1=450000
x2=525000 #bounds used for the detection

X1=450000
X2=500000 #bounds used for the training

D=split(data1,x1,x2)
D2=split(data2,x1,x2)

D=round(D,2) #To improve calculation time
D2=round(D2,2)

os.chdir("/Users/victorletzelter/Desktop/Projet_python")
exec(open('RMS_gestion_2.py').read()) #The functions related RMS
exec(open('/Users/victorletzelter/Desktop/Projet_python/opt.py').read())
#The function related to parameters optimisation

### Smoothing the Signal, then Removing the offset by Savitzky-Golay method

Da=smooth(D,D2,x1,x2) #in RMS_gestion
D=Da[0]
D2=Da[1]

### Loading of the training part

exec(open("/Users/victorletzelter/Desktop/Projet_python/sampleziNEW4.txt").read())
#Getting the zi variable

###Optimal values for the next step :

GP2_min=[]
DP2_min=[]
GP_min=[]
DP_min=[]

GP2_max=[]
DP2_max=[]
GP_max=[]
DP_max=[]
```

```
traitement(zi)
Rate_keep=70

###GENERAL CODE (Values chosen N=4, F=100 and S=100)
###Value of the thresold factor for the coherence

seuil_min=Coh_opt(DP_min,DP2_min,Rate_keep,N=4,F=100,X1=450000,X2=500000)
seuil_max=Coh_opt(DP_max,DP2_max,Rate_keep,N=4,F=100,X1=450000,X2=500000)
seuil_opt=min(seuil_min,seuil_max)

###Optimal values for the distance of the peak with the MA
#(in number of moving SD)

NSD_min=Distances_opt(DP_min,DP2_min,Rate_keep=Rate_keep,MAX=-1,S=100)
NSD_max=Distances_opt(DP_max,DP2_max,Rate_keep=Rate_keep,MAX=1,S=100)
#Not enough data for the max
NSD_opt=min(NSD_min,NSD_max)

###Optimal values of the errors on the exponential fitting

a_eropt_min,b_eropt_min,c_eropt_min=valeur_opt(DP_min,DP2_min,MAX=-1,Rate_keep=Rate_ke
a_eropt_max,b_eropt_max,c_eropt_max=valeur_opt(DP_max,DP2_max,MAX=1,Rate_keep=Rate_kee

###Optimal values of the coverage of the invervals thresold rate

Lp=liste_quotient(DP_min,DP2_min)
Lp2=liste_quotient(DP_max,DP2_max)
p_opt=min(np.percentile(Lp, 100-Rate_keep),np.percentile(Lp2, 100-Rate_keep))

### Detection of the intervals of Zonal Flows

def Detect(data,c1='blue',c2='red',NSD=NSD_opt,N=4,F=100,S=100) :

    #Detection of local minimums
    indMin, _=find_peaks(-data['Potential(V)']+max(abs(data['Potential(V)'])),height=0

    #Detection of local maximums

    indMax, _=find_peaks(data['Potential(V)'],height=0)
    #indMax : local maximum indices

    #We keep only the peaks where the Corr is >0.8 / we use a confidence interval

    indMin=indMin+x1
    indMax=indMax+x1
```

```
###First filtration based on LRC :

indMin_2=[]
indMax_2=[]

t=Correlation_s(D,D2,seuil_opt,x1,x2,N,F) #Spectral correlation
#btw the 2 signals

for e in indMax :
    if iscorr(e,t,100) :
        indMax_2.append(e)

for e in indMin :
    if iscorr(e,t,100) :
        indMin_2.append(e)

###Second filtration based on the RMS / STD :

indMin_3=[]
indMax_3=[]

def times_rms(data,S) :
    data_m=data.rolling(S).mean()
    data_rms=np.sqrt(data.pow(2).rolling(S).apply(lambda x: np.sqrt(x.mean())))
    data_sd=data.rolling(S).std()
    sup=data_m['Potential(V)']+NSD*data_sd['Potential(V)']
    inf=data_m['Potential(V)']-NSD*data_sd['Potential(V)']
    t=(data['Potential(V)']>=sup).astype(int)
    t_=(data['Potential(V)']<=inf).astype(int)
    return(t,t_)

t,t_=times_rms(data,S)

for e in indMax_2 :
    if isok_1(e,1,t,t_) :
        indMax_3.append(e)

for e in indMin_2 :
    if isok_1(e,-1,t,t_) :
        indMin_3.append(e)

### Third filtration based on the non-coverage

indMin_4=copy.deepcopy(indMin_3)
indMin_4=update(indMin_3,-1,data)
indMax_4=update(indMax_3,1,data)
```

```
    ###Fourth filtration based on the exponential decay

    ### Exp with the Data 1
    filter_min=filter_exp_min(indMin_4,data)
    indMin_5=filter_min[0]
    Sorties_min=filter_min[1]
    filter_max=filter_exp_max(indMax_4,data)
    indMax_5=filter_max[0]
    Sorties_max=filter_max[1]

    indMin_6=indMin_5
    indMax_6=indMax_5

    ###Deduction of the ZF intervals :
    Int_max=[]

    for i in range (0,len(filter_max[0])) :
        x=indMax_6[i]
        y=Sorties_max[i]
        Int_max.append((x,x+10*(y+1)))
        X=np.arange(x,x+10*(y+1),1)
        plt.fill_between(X,-30 ,30, color=c1, alpha=0.1)

    Int_min=[]

    for i in range (0,len(filter_min[0])) :
        x=indMin_6[i]
        y=Sorties_min[i]
        Int_min.append((x,x+10*(y+1)))
        X=np.arange(x,x+10*(y+1),1)
        plt.fill_between(X,-30,30, color=c2, alpha=0.1)

    return((Int_min,Int_max))

I=Detect(D,'blue','red')
Int_min=I[0]
Int_max=I[1]
I2=Detect(D2,'green','orange')
Int_min2=I2[0]
Int_max2=I2[1]

###Next step : We keep only the times where the two intervals
#recover (red/orange or blue/green)

def indicator(Int) :
```

```
    ind=[0]*(x2-x1)
    ind=pd.DataFrame(ind)
    ind=ind.set_index(np.arange(x1,x2,1))

    for i in range (len(Int)) :

        for l in range (Int[i][0],Int[i][1]) :

            ind[0][l]=1

    return(ind[0])

ind_min=indicator(Int_min)
ind2_min=indicator(Int_min2)

ind_max=indicator(Int_max)
ind2_max=indicator(Int_max2)

product_min=ind_min*ind2_min
product_max=ind_max*ind2_max

def filter(Int,MAX) : #MAX=1 for max et MAX=-1 for min

    F_Int=[]

    for (a,b) in Int :

        keep=0 #We do not keep

        for e in range (a,b) :

            if MAX==-1 :
                if product_min[e]==1 :
                    keep=1

            elif MAX==1 :
                if product_max[e]==1 :
                    keep=1

        if keep==1 :

            F_Int.append((a,b))

    return(F_Int)
```

```
F_Int_min=filter(Int_min,-1)
F_Int_max=filter(Int_max,1)
F_Int_min2=filter(Int_min2,-1)
F_Int_max2=filter(Int_max2,1)

###Next step again : We keep only, among the intervals that
#recover, those for which the quotient of the lengths remains reasonable.

def filter_length(Int,Int2,MAX) :

    F_Int=[]
    F_Int2=[]

    for (a,b) in Int :
        for (c,d) in Int2 :

            if (a<c and b>c) or (a<c and d<b) or (c<a and d>a) or (c<a and b<d) :
            #The two intervals recover
                if MAX==1 :
                    liste=[product_max[e] for e in range (min(a,c),max(b,d))]
                else :
                    liste=[product_min[e] for e in range (min(a,c),max(b,d))]

                n=len(liste)
                liste=np.asarray(liste)
                p=len(np.where(liste==1)[0])/n

                if p>=p_opt :
                    if not((a,b) in F_Int) :
                        F_Int.append((a,b))
                    if not((c,d) in F_Int2) :
                        F_Int2.append((c,d))

    return(F_Int,F_Int2)

F_Int_min,F_Int_min2=filter_length(F_Int_min,F_Int_min2,-1)
F_Int_max,F_Int_max2=filter_length(F_Int_max,F_Int_max2,1)

name1='MaxRatekeep{}'.format(Rate_keep)
name2='MinRatekeep{}'.format(Rate_keep)
save(F_Int_min,'F_int_min',name2+'1')
save(F_Int_min2,'F_int_min2',name2+'2')
save(F_Int_max,'F_int_max',name1+'1')
save(F_Int_max2,'F_int_max2',name1+'2')

fill(F_Int_min,F_Int_max,'blue','red')
```

```
fill(F_Int_min2,F_Int_max2,'green','orange')

plt.grid()
plt.xlabel('Point Number')
plt.ylabel('Potential(V)')
plt.plot(D['Potential(V)'])
plt.plot(D2['Potential(V)'])

plt.show()
```

19 Annex: Adjustment of the parameters of the probabilistic model

As outlined in **Section 11**, the probabilistic model requires many adjusting parameters : a, b, σ_0, p. In the first probabilistic model, all of these parameters were considered as constants, as the adjustment was performed using the results on the ZF detection. Let's denote $\mathcal{DP}$ the subsets on couples (x_1, x_2) that correspond to the beginning and the end of the damping part of ZF that begin in a minimum which were detected by the detection algorithm presented in **Section 6**.

Adjustment of the value of p

The adjustment of the parameter p of the geometric law defined in **Section 11**, can be determined by estimating at first the expectancy m of the geometric law [1], and by using the relation

$$p \simeq \frac{1}{m}$$

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

def sort(DP) : #This function sorts the corresponding intervals
    DP_ini=np.asarray([e[0] for e in DP])
    Indexes=list(np.argsort(DP_ini,axis=0))
    Indexes=np.asarray(Indexes)
    DP_sorted=[DP[k] for k in Indexes]
    return(DP_sorted)

def histP(DP) :
    DP_sorted=sort(DP)
    WT=[]

    for i in range (len(DP_sorted)) :
```

[1] Which is the average number of points between the end of a ZF interval, and the beginning of a new one.

```
        if i>0 :
            WT.append(DP_sorted[i][0]-DP_sorted[i-1][1]+1)

    plt.hist(WT,bins=50,color='red',ec='red',alpha=0.4)
    return(WT)

WT=histP(DP_min)
p=1/(np.mean(WT)) #The parameter of the geometric law
```

Adjustment of the σ_0 value

```
def histS(D) : #Histogram of the values of the potential
    hist, bin_edges=np.histogram(D['Potential(V)'],bins=50)
    plt.plot(hist)

histS(D)
plt.show() #We check if the pace corresponds to a normal distribution

mu=np.mean(D['Potential(V)']) #Mean estimation of the distribution (close from 0)
sigma0=np.std(D['Potential(V)']) #Std estimation
```

Adjustment of the a value (the process is similar for the b value)

```
def histA(DP,Numero=1) : #Function that return the histogram of the
#a values in the ZF associated to the dataset of each probes

    A=[]

    for e in DP :

        if Numero==1 :
            a=score_min(e[0],D)[2][0]

        elif Numero==2 :
            a=score_min(e[0],D2)[2][0]

        A.append(a)

    plt.hist(A,density=True)

    return(A)
```

The function score min is defined in **Section 21**.

In the First probabilistic model, the value of a can therefore be estimated by

```
A=histA(DP,1)
a=np.median(A)
```

To fit the distribution of a with a gamma law, the following script can be used

```
def plot_gamma(H) :

    alpha=np.mean(H)**2/np.var(H)
    beta=1/(np.var(H)/np.mean(H))
    X=np.arange(min(H),max(H),1)
    Y=[scipy.stats.gamma.pdf(-e,alpha,0,1/beta) for e in X]
    plt.plot(X,Y)
    plt.show()
    return(alpha,beta)

alphaA,betaA=plot_gamma([-e for e in A]) #The minus sign takes
#account of the negative values of a values.
```

20 Annex: Code for the generation of artificial data

This section provides the complete code for the generation of labelled artificial data using the last probabilistic model, and given that the values of the adjusting parameters were determined.

```
import os
os.chdir("/Users/victorletzelter/Desktop/Projet_python")
exec(open('imports_gestion.py').read())

def gen3(N,rd_seed) : #N is the number of points number to be
#generated
#rd_seed corresponds to the random seed

#This function returns labelled artificial data
    #Generation of data without ZF at first
    mu=0
    sigma0=5.866
    p=0.0009
    a=-14
    b=0.04
    x1=0
    x2=N
    alphaH=5.399
    betaH=0.052
    alphaA=5.111 #Warning : inverse the sign
    betaA=0.361 #Warning : inverse the sign
    alphaB=8.296 #Warning : inverse the sign
    betaB=222.735 #Warning : inverse the sign

    dat=np.random.normal(mu, sigma0, N) #Generation of the normal law
```

```
dat=pd.DataFrame(dat,columns=['S'])
datM=dat.rolling(100).mean() #Applying moving average
S1=np.std(dat['S'])
S2=np.std(datM['S'])
datM=datM*(S1/S2)

#Then, the savgol_filter is applied two times

S_1=savgol_filter(datM['S'],7, 2)
dat_s=pd.DataFrame.copy(datM)

for i in range (x1,x2) :
    dat_s['S'][i]=S_1[i-x1]

S_2=savgol_filter(datM['S'],int(len(dat_s)/10)+1, 2)

dat_s2=pd.DataFrame.copy(dat_s)

for i in range (x1,x2) :
    dat_s2['S'][i]=dat_s['S'][i]-S_2[i-x1]

#plt.plot(dat_s2,label='Simulated')
#plt.plot(D['Potential(V)'],label='Real')

#The ZF are added

np.random.seed(rd_seed)
Rep1={'S':[0]*N,'zf':[0]*N}
E=pd.DataFrame(Rep1)

j=0

while j<N :

    T = min(np.random.geometric(p)+j,N-h) #Beginning of the ZF
    h=int(np.random.gamma(alphaH,1/betaH)) #The length of the ZF
    a=-np.random.gamma(alphaA,1/betaA)
    b=np.random.gamma(alphaB,1/betaB)

    for elt in np.arange(T,T+h,1) :
        E['S'][elt]=a*np.exp(-b*(elt-T))
        E['zf'][elt]=1

    j=T+h

dat_WZ=dat_s2['S']+E['S']
```

```
    return((dat_WZ,E['zf']))

def write(dat_WZ,E,N,Name) : #dat_WZ simulated data containing the
#ZF intervals
    os.chdir("/Users/victorletzelter/Desktop/Projet_python")
    file = open(Name, "w")
    file.write('{}\t{}\n'.format('S','zf'))
    for i in range (int(0.06*N),N,1) : #This lag allows to remove the NaN values
        file.write('{}\t{}\n'.format(round(dat_WZ[i],3),E['zf'][i]))
    file.close()

(datgen,E)=gen3(9500000,998)
(datgen2,E2)=gen3(500000,1222) #200000

write(datgen2,E2,9500000,'Data_WE_train_big')
write(datgen,E,500000,'Data_WE_test_big')
```

21 Annex: Functions created for exponential fitting

```
from sklearn.metrics import mean_squared_error
from math import *
import warnings
from scipy.optimize import OptimizeWarning

def func(x,popt) :
#x in a point number
#popt is a list containing the three values of the coefficients

#This function returns the value of the exponential
    a=popt[0]
    b=popt[1]
    c=popt[2]
    return(a*np.exp(b*x)+c)

# Get the optimal size of the interval :
def opt(e,MAX,data) :
#e in the point number of the extremum
#MAX : 1 if the extremum is a maximum and -1 if it is a minimum
#data is the source of data

#This function returns the optimal size of the interval for exponential fitting

    Er=0 #While Er=0, there was no warnings
    a_e=[100]*20
```

```
b_e=[100]*20
c_e=[100]*20
l_err=[100]*20
RMSE=[100]*20
j=2

while j<20 and e+10+10*j<len(data['Potential(V)'])+data.index[0] :
    E=0
    X=np.arange(0,10+10*j,1)
    Dat=[data['Potential(V)'][e+i] for i in np.arange(0,10+10*j,1)]

    with warnings.catch_warnings():
        warnings.simplefilter("error", OptimizeWarning)
        warnings.simplefilter("error", RuntimeWarning)
        warnings.simplefilter("error", RuntimeError)

        try:
            if MAX==-1 :
                popt,pcov = scipy.optimize.curve_fit(lambda t,a,b,c: a*np.exp(b*t)

            elif MAX==1 :
                popt,pcov = scipy.optimize.curve_fit(lambda t,a,b,c: a*np.exp(b*t)

            a_er=np.sqrt(np.diag(pcov))[0]/len(X)
            b_er=np.sqrt(np.diag(pcov))[1]/len(X)
            c_er=np.sqrt(np.diag(pcov))[2]/len(X)

        except OptimizeWarning:
            Er=1
            print('Maxed out calls.')

        except RuntimeError:
            print("Error - curve_fit failed")
            Er=1

        except RuntimeWarning :
            print("Invalid value")
            Er=1

        except Warning :
            print('Warning was raised as an exception!')
            Er=1

        except FloatingPointError :
            print('Warning was raised as an exception!')
            Er=1
```

```
            if Er==0 :
                a_e[j]=a_er
                b_e[j]=b_er
                c_e[j]=c_er
                l_err[j]=(a_er+100*b_er+c_er)/102
                RMSE[j]=sqrt(mean_squared_error(Dat, func(X,popt)))

            else :
                a_e[j]=100
                b_e[j]=100
                c_e[j]=100
                l_err[j]=100
                RMSE[j]=100

            j+=1

        min_value_2 = min(RMSE)
        min_index_2 = RMSE.index(min_value_2)

        min_value = min(l_err)
        min_index = l_err.index(min_value)

        if np.max(l_err)!=np.min(l_err) and np.max(RMSE)!=np.min(RMSE) :

            mix=0.999*(l_err-np.min(l_err))/(np.max(l_err)-np.min(l_err))+0.001*(RMSE-np.m

        else :

            mix=[100]*15

        min_value_3=np.min(mix)
        min_index_3 = np.where(mix==min_value_3)[0][0]

        return(min_index_3,mix,RMSE)

def score_min(e,data) : #This function provides the error
#associated with the exponential fit for the point e, when it is a
#minimum
        E=0 #While E=0, there was no warnings
        min_index,l_err,RMSE=opt(e,-1,data)
        X=np.arange(0,10+10*min_index,1)

        Dat=[data['Potential(V)'][e+i] for i in np.arange(0,min(10+10*min_index,data.index

        if len(X)==len(Dat) :
```

```
        with warnings.catch_warnings():

            warnings.simplefilter("error", OptimizeWarning)
            warnings.simplefilter("error", RuntimeError)

            try:
                popt,pcov = scipy.optimize.curve_fit(lambda t,a,b,c: a*np.exp(b*t)+c,
                a_er=np.sqrt(np.diag(pcov))[0]
                b_er=np.sqrt(np.diag(pcov))[1]
                c_er=np.sqrt(np.diag(pcov))[2]

            except OptimizeWarning:
                E=1
                print('Maxed out calls.')

            except RuntimeError:
                print("Error - curve_fit failed")
                E=1

            except RuntimeWarning :
                print("Invalid value")
                E=1

            except Warning :
                print('Warning was raised as an exception!')
                E=1
            except FloatingPointError :
                print('Warning was raised as an exception!')
                E=1

    else :
        E=1

    if E==0 :
        return((a_er,b_er,c_er),min_index,popt)

    else :
        return((100,100,100),min_index,popt)
```

Similar function were also created when the starting point is a maximum.

22 Annex: Code of the NN model

The last version of the NN model, designed using the Tensorflow library, is given below

```
### Imports

import numpy as np
import pandas as pd
import tensorflow as tf
import warnings
import os
import time
import random

os.chdir("/Users/victorletzelter/Desktop/Projet_python")
warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)

### Reading csv, labelling

data=pd.read_csv("/Users/victorletzelter/Desktop/Projet_python/Data_WE_train_big",del
data_test=pd.read_csv("/Users/victorletzelter/Desktop/Projet_python/Data_WE_test_big"
Nr=1

def to_supervised_with_recoverage(data,h=50,Nr=2) : #data is a data
#frame containing the Signal, and the labelled values of the ZF
#h in the length of the input vectors
#Nr is the recoverage parameters (a divider of h)

#It returns two tensors, that will be the inputs and outputs of the
#training part of the NN

  X=[]
  Y=[]

  for i in range (0,len(data['S']),int(h/Nr)) :
    if len(data['S'][i:i+h])==h :
      X.append(data['S'][i:i+h])
      Y.append(data['zf'][i:i+h])
  n=np.shape(X)[0]
  M=np.zeros((n-1,h))
  Sortie=np.zeros((n-1,h))
  for i in range (n-1) :
    M[i,:]=np.asmatrix(X[i])
    Sortie[i,:]=np.asmatrix(Y[i])
  In=tf.convert_to_tensor(M)
  Out=tf.convert_to_tensor(Sortie)

  return(In,Out)

def convertV3(Ys,h=50,Nr=5) : #Ys is the output of the NN
```

```
#The function returns of temporal series, which has the same length
#as the signal data['S'] initially provided

  n,p=np.shape(Ys)
  Yt=[0]*len(data['S'])
  m=1

  for j in range (0,int(h/Nr)) :
    Yt[j]=Ys[0][j]

  for l in range (1,len(Ys)) :

    m+=1

    for j in range (0,int(h/Nr)) :

      liste=[Ys[l][j]]

      k=1
      while j+k*int(h/Nr)<h and l-k<n and l-k>=0 :

        liste.append(Ys[l-k][j+k*int(h/Nr)])
        k+=1

      Yt[l*int(h/Nr)+j]=np.max(liste)

  for x in range (1,Nr) :

    for j in range (x*int(h/Nr),(x+1)*int(h/Nr)) :

      liste=[Ys[len(Ys)-1][j]]
      k=1

      while j+k*int(h/Nr)<h and l-k<n and l-k>=0 :

        liste.append(Ys[l-k][j+k*int(h/Nr)])
        k+=1

      Yt[(len(Ys)-1)*int(h/Nr)+j]=np.max(liste)

  return(Yt)

###Functions

def save(var,name,namefile) :
```

```
    file = open(namefile, "w")
    file.write("\%s = \%s\n" \%(name, var))
    file.close()

Nr=5
h=100

In,Out=to_supervised_with_recoverage(data,h,Nr)
In2,Out2=to_supervised_with_recoverage(data_test,h,Nr)

Couche1=128
Couche2=256

activation1='selu'
activation2='selu'
kernel1='lecun_normal'
kernel2='lecun_normal'
Dropout=0.01

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(units=Couche1, activation=activation1,kernel_initializer=kerne
  tf.keras.layers.Dropout(Dropout),
  tf.keras.layers.Dense(units=Couche2, activation=activation2,kernel_initializer=kerne
  tf.keras.layers.Dropout(Dropout),
  tf.keras.layers.Dense(h,activation='sigmoid')
])

#lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
#    initial_learning_rate=initial_learning_rate,
#    decay_steps=33046,
#    decay_rate=Decay,staircase=True)
#initial_learning_rate * decay_rate ^ (step / decay_steps)
#initial_learning_rate=0.001

earlystopping = tf.keras.callbacks.EarlyStopping(monitor ="val_loss",mode ="min", pati

model.compile(
        optimizer='Adam',
        loss= tf.keras.losses.MeanSquaredError(),
        metrics=[tf.keras.metrics.MeanSquaredError()],
    )

checkpoint_path = "/Users/victorletzelter/Desktop/Projet_python/Weights"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
```

```
cp_callback =tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,save_weights_

hist=model.fit(In,
Out,batch_size=256,epochs=300,validation_split=0.8,validatio
n_data=(In2,Out2), callbacks =[earlystopping])

name='Hist'
save(hist.history,'x',name)

import os
Res=model(In2).numpy() #Applying the model on the test set
os.chdir("/Users/victorletzelter/Desktop/Projet_python")
save(Res,'Res','Res')

n,p=np.shape(Res)
print(list(Res[1]))

ZF=convertV3(Res,h,Nr) #converting the output on the NN
name='NRmax{}'.format(Nr)
save(ZF,'ZF',name)

### Switch to Python 3.9 for this part
import os

os.chdir("/Users/victorletzelter/Desktop/Projet_python")
exec(open('imports_gestion.py').read())
data_test=pd.read_csv("/Users/victorletzelter/Desktop/Projet_python/Data_WE_test_big"

ListeNr=[5]
Nr=5
Listeerr=[]

exec(open("/Users/victorletzelter/Desktop/Projet_python/NRmax{}".format(Nr)).read())
ZF=pd.DataFrame(ZF)[0]
ZFp=(ZF>0.5)
Listeerr.append(np.sqrt(np.mean((ZFp-data_test['zf'])**2)))

plt.plot(data_test['S'],label='Simulated data')
plt.plot(data_test['zf'],label='Presence of ZF (Truth)')
plt.plot(ZF,label='Presence of ZF (Deduced by the NN)')
plt.grid()
plt.legend()
plt.title('ZF detection with artificial data')
plt.xlabel('Point Number')
plt.ylabel('Potential(V)')
plt.show()
```

On another note, the function for getting the F_1 score of the detection algorithm is the following

```
def performance(F_Int_min,F_Int_min2,F_Int_max,F_Int_max2) : #These four intervals re
#The variables DP_min, and DP_max stand for the true ZF i.e those which were labelled

#The function return the F_1 Score of the detection algorithm, provided that this det

    R1=indicator(F_Int_min)
    R2=indicator(F_Int_max)
    L1=indicator(DP_min)
    L2=indicator(DP_max)

    product_min1=R1*L1
    product_max1=R2*L2

    def filter2(Int,MAX) : #MAX=1 for max and MAX=-1 for min

        F_Int=[]

        for (a,b) in Int :

            keep=0 #We do not keep

            for e in range (a,b) :

                if MAX==-1 :
                    if product_min1[e]==1 :
                        keep=1

                elif MAX==1 :
                    if product_max1[e]==1 :
                        keep=1

            if keep==1 :

                F_Int.append((a,b))

        return(F_Int)

    Confusion=filter2(F_Int_min,-1)

    Precision=len(Confusion)/len(F_Int_min) #Precision=TP/(TP+FP)
    Recall=len(Confusion)/len(DP_min) #Recall=TP/(TP+FN)

    F1_score1=s.harmonic_mean([Precision,Recall])
```

```
R1=indicator(F_Int_min2)
R2=indicator(F_Int_max2)
L1=indicator(DP2_min)
L2=indicator(DP2_max)

product_min1=R1*L1
product_max1=R2*L2

def filter2(Int,MAX) : #MAX=1 for max and MAX=-1 for min

    F_Int=[]

    for (a,b) in Int :

        keep=0 #We do not keep

        for e in range (a,b) :

            if MAX==-1 :
                if product_min1[e]==1 :
                    keep=1

            elif MAX==1 :
                if product_max1[e]==1 :
                    keep=1

        if keep==1 :

            F_Int.append((a,b))

    return(F_Int)

Confusion2=filter2(F_Int_min2,-1)

Precision2=len(Confusion2)/len(F_Int_min2)
Recall2=len(Confusion2)/len(DP2_min)

F1_score2=s.harmonic_mean([Precision2,Recall2])

F1_s=s.mean([F1_score1,F1_score2])

return((F1_s,Precision,Recall,Precision2,Recall2))
```

23 Annex: Random grid for hyperparameters tuning

Optimizing the architecture of a Neural Network requires both mathematical analysis, and huge computation cost. The naive method consisting on adjusting parameters using linspace values has its limitation in computation cost. Therefore, the library `random` has been used for adjusting the parameters using uniform distribution in defined intervals. This *random* choice of parameters is performed several times, and then, the intervals for uniforms distribution are adjusted again, until the results are satisfactory enough.

As a case study, the set of values chosen for the parameters at stake were

1. For the activation of the first and the second layer : {'relu','selu','linear'}
2. For the initialisation mode of the layers {'he normal','glorot uniform','lecun normal'}
3. The initial learning rate : $10^{\text{random.uniform}(-5,\,-2)}$, the Decay rate $1 - 10^{\text{random.uniform}(-4,\,-1)}$ [1]
4. The number of neurons in the first layer and the second layers : random.uniform$(70,\,150)$ and random.uniform$(150,\,300)$. [2]
5. The Dropout value random.uniform$(0,0.1)$

For each values of the hyperparameters chosen, the model was trained and tested, and the minimum value of the metric function associated was automatically written in a text file. The process was repeated about 500 times (requiring about 10 hours of calculus), and it has been found, among other things, that the 15 best performances were obtained using 'selu' activation function.

Other functions written during this internship, or explanations about the strategy and the code can be provided on request (*e-mail : letzelter.victor@hotmail.fr*).

[1] It has finally be found that adding a schedule of the decay of the learning rate (tf.keras.optimizers.schedules.ExponentialDecay for instance) does not improves significantly the performance with regards to the conventional optimizer 'Adam'.

[2] As these values did not affect that much the results, the values 128 and 256 were chosen to prevent issues related to the *Curse of dimensionality* (See `https://en.wikipedia.org/wiki/Curseofdimensionality`)