ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ENERGETICA

TESI DI LAUREA

in
Trasporto di Particelle e di Radiazione M

GYROKINETIC MODELLING OF PLASMA TRANSPORT AND INVESTIGATION ON TEST PARTICLE DYNAMICS

CANDIDATO:
Matteo Stanzani

RELATORE:
Prof.Ing. Domiziano Mostacci

CORRELATORI:
Prof. Guido Ciraolo
Prof. Cristel Chandre

Anno Accademico 2020/21

Sessione IV

So here we are again, today, on another trail, he thought, reaching for a cup of precious gas and vacuum, a handful of different fire with which to run back up cold space, lighting our way, and take to Earth a gift of fire that might burn forever. Why? He knew the answer before the question.

– Ray Bradbury, *The Golden Apples of the Sun*

Abstract

Turbulent plasmas inside tokamaks are modeled and studied using guiding center theory, applied to charged test particles, in a Hamiltonian framework. The equations of motion for the guiding center dynamics, under the conditions of a constant and uniform magnetic field and turbulent electrostatic field are derived by averaging over the fast gyroangle, for the first and second order in the guiding center potential, using invertible changes of coordinates such as Lie transforms. The equations of motion are then made dimensionless, exploiting temporal and spatial periodicities of the model chosen for the electrostatic potential. They are implemented numerically in Python. Fast Fourier Transform and its inverse are used. Improvements to the original Python scripts are made, notably the introduction of a power-law curve fitting to account for anomalous diffusion, the possibility to integrate the equations in two steps to save computational time by removing trapped trajectories, and the implementation of multicolored stroboscopic plots to distinguish between trapped and untrapped guiding centers. The post-processing of the results is made in MATLAB. The values and ranges of the parameters chosen for the simulations are selected based on numerous simulations used as feedback tools. In particular, a recurring value for the threshold to detect trapped trajectories is evidenced. Effects of the Larmor radius, the amplitude of the guiding center potential and the intensity of its second order term are studied by analyzing their diffusive regimes, their stroboscopic plots and the shape of guiding center potentials. The main result is the identification of cases anomalous diffusion depending on the values of the parameters (mostly the Larmor radius). The transitions between diffusive regimes are identified. The presence of *highways* for the super-diffusive trajectories are unveiled. The influence of the charge on these transitions from diffusive to ballistic behaviors is analyzed.

Contents

Chapter 1

Introduction

Nowadays, magnetically confined plasmas can not be controlled and exploited up to the point of properly producing energy with fusion reactors such as tokamaks. Turbulence within the plasma itself can reduce the effectiveness of magnetic confinement, leading to a loss of the conditions that allow fusion reactions to take place.
The motion of a charged particle in strong electromagnetic fields is characterized by a fast cyclotron motion around magnetic field lines and slower drifts associated with the motion of its guiding center, i.e. the center around which the rotational component of motion develops. Given that the fast component of motion is, in practice, not relevant for the analysis of plasma turbulence, reduced kinetic models have been developed in the late 1980s and numerically studied from the early 2000s.
The hottest regions of plasma are weakly collisional and for this reason they are studied with with gyrokinetic codes.
Gyrokinetics is nowadays considered to be the standard way to investigate plasma turbulence. It is associated with distribution functions for the gyrocenters, i.e. modified guiding centers that account for low-frequency electromagnetic fluctuations produced by plasmas themselves.
Multiple gyrokinetic models exist and they depend on the assumptions made for the reduction. The main element behind them is the choice of the ordering between the various small parameters characterizing elements like the inhomogeneities of the background electromagnetic field, the amplitude and the fluctuations (in both space and time) of perturbing electromagnetic fields.
Deriving a gyrokinetic model has to be consistent with the chosen ordering from the particle to the guiding center dynamics and from the guiding center to the gyrocenter one. The order associated with the reduction sets the limits of investigation of a given model. Currently, all gyrokinetic codes are of first or second order in the amplitude of the perturbing electromagnetic fields, but while the first order is well known, there is less information about how the second order term affects the dynamics of gyrocenters.
Gyrokinetics is based on guiding center theory, in which particles are tracked using guid-

ing centers instead of gyrocenters.
The equations for guiding center dynamics can be obtained through changes of coordinates that allow one to move from the point of view of particles to the one of guiding centers. Working with guiding centers instead of particles has a huge advantage when numerical simulations are considered, in particular because a lot of computational time can be saved. In that regard, one of the most important procedures within guiding center theory consists in averaging the angle of rotation of particles around their guiding centers, with the consequence of removing it from the set of variables. In essence, this is possible because the rotation of charged particles around magnetic field lines is much faster than their translation and is related to very small spatial scales. Many phenomena of interest for magnetically confined plasmas, such as turbulence, suit gyrokinetic and guiding center approaches. Following [Lit80], a Hamiltonian framework has been used in this work. The cancellation of the gyroangle is done, in practice, with a Lie transform acting on the Hamiltonian of the system and with integrations over the angle itself (this is the case, for example, when changes of coordinates are being applied to electrostatic potentials).
The equations that are obtained allow one to know the position of guiding centers in the plane perpendicular to the uniform magnetic field lines through spatial derivatives of potentials and can be implemented with more or less efforts, depending on the electromagnetic field, into numerical scripts for fusion-related simulations. For this work, a dimensionless version of the equations, equipped with a simplified version of the electromagnetic fields that can be found in real tokamaks, has been implemented in Python. In particular, the helicoidal magnetic field was turned into a uniform one and measurements of actual electrostatic fields were used to model an analytic electrostatic potential as a superposition of trigonometric functions of space and time, with random phases serving as the tools to reproduce turbulence. The integration of the equations can be a long process, given the nature of the problem itself, but when there are periodic functions, the extremely efficient Fast Fourier Transform algorithm can be used (together with its inverse), making possible to perform derivatives very rapidly. Once the equations and the tools to numerically solve them are available, then what remains to be done is to write efficient scripts, select the kinds of outputs they can produce and/or improving already existing files. The latter is what has been done in this work, in the sense that, starting from the original versions of the Python scripts, they have been extensively developed and changed, simulation after simulation, in order to make them faster and more loyal to the physics involved.
The peculiar feature of the equations for guiding center dynamics associated with this work is the fact that they involve two orders of the potential. Orders beyond the first are usually neglected because they are considered to have very small effects. The core idea for this work was to see if adding the second term, which is more complicated from a mathematical point of view and slower to numerically calculate, produced some relevant

effects that could justify the implementation of second order terms into more sophisticated and developed gyrokinetic codes.
The structure of the manuscript is the following:

- In Chapter 2, the physical context of this work is presented in short, with focuses on nuclear fusion reactions, magnetic confinement of plasmas and motion of charged particles.
- In Chapter 3, the physical problem associated with this work is outlined and the equations for guiding center dynamics are derived from scratch. The potential associated with such equations is characterized to mimic the ones that can be measured in turbulent plasmas inside tokamaks. Then, quantities and equations undergo a nondimensionalization procedure.
- In Chapter 4, final forms of the Python scripts are presented and comparisons with their older versions are made to highlight and justify the modifications that have been introduced. The MATLAB files used for the post-processing are described, too. Some interesting results obtained during the tuning of the scripts are shown.
- In Chapter 5, the outputs of the simulations launched with the final versions of the Python files are extensively analyzed.
- In Chapter 6 a summary of the main results and a general recap is made, while the Appendices contain some information about the main mathematical and numerical tools that have been exploited.

Chapter 2

Controlled thermonuclear fusion and magnetic plasma confinement

This work is related to turbulent plasmas in tokamaks. In this chapter, an overview of nuclear fusion, charged particles interacting with electromagnetic fields and magnetic confinement in experimental fusion reactors will be presented.
This chapter is mostly based on [Mon83], [Cha18], [Fre08], [IRF19], [Vil07] and [HSK00].

2.1 Magnetically confined thermonuclear fusion

It is well known that an atomic nucleus with atomic mass $\mathcal{A}$ is composed of $\mathcal{Z}$ protons and $\mathcal{N}$ neutrons kept together by bonds. The energy Δ required to completely break these bonds is called nuclear binding energy and it corresponds, in different units, to the mass defect between the nucleus and its nucleons considered as free particles: during the formation of a nucleus, a fraction of the mass of neutrons and protons is released as energy. Therefore, in order to get back the starting free particles, the same amount of energy released during the nucleus creation needs be given to it. The higher the average nucleon binding energy $\Delta/\mathcal{A}$, the more stable the corresponding nucleus. Nuclear reactions which proceed towards the formation of more stable nuclei than the starting ones are exothermic and such that the total mass of the reactants is higher than the one of the products. The mass defect associated with the reaction is equal to the energy released during the process.
With nuclear fusion it is possible to obtain energy thanks to the combination of two light nuclei into a new, more stable one. This is the reason why this reaction is of high interest for humans. Fusion is exploited for military applications inside the hydrogen bomb and since last century a lot of research has been conducted in order to use it also as a safe and clean energy source inside power plants. It would represent an important way to both contrast climate change and meet the always increasing energy demand from humans.

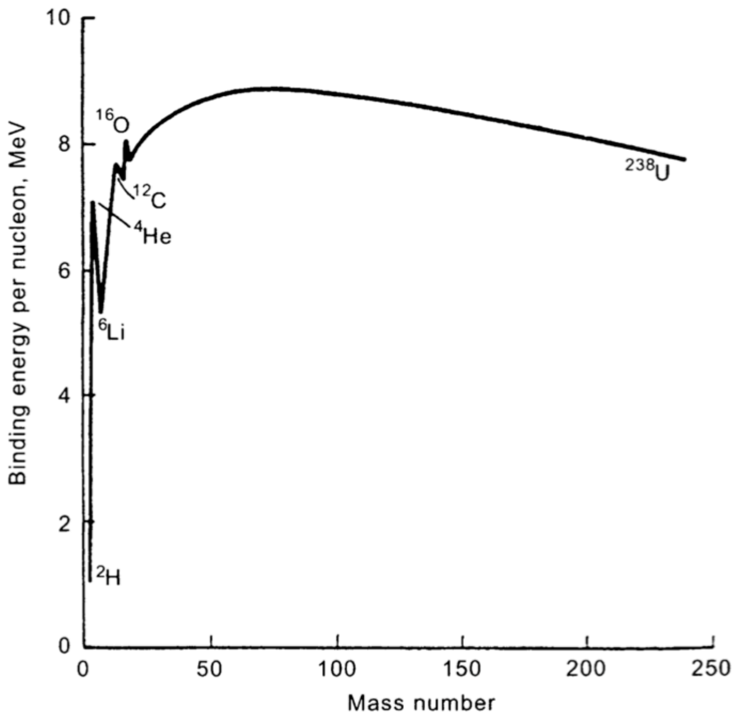


Figure 2.1: Average binding energy per nucleon, expressed in MeV, as a function of atomic mass number. Taken from [Sta07].

One of the main reasons why fusion is already used for non-pacific applications only is the fact that for these, the desired effect is an abrupt release of energy: no particular control over the process is required. On the contrary, fusion would need to be constantly monitored inside nuclear plants, as it is common for every kind of energy source. Being able to properly control fusion reactions is a high demanding task and further research is required.
In order to undergo fusion, nuclei have to be sufficiently close, but them being positively charged means that they have the tendency to repel each other. To overcome this repulsive force, they need to have an energy so high that the system is the state of plasma: atoms are completely ionized, with electrons detached from their nuclei and forming, with them, a globally quasi-neutral fluid able to both conduct electricity and interact with electromagnetic fields.
Inside stars, fusion is a continuous and natural process made possible by the fact that the plasma spontaneous tendency to disperse and cool down is countered by the force of gravity. On Earth, however, gravitational confinement, which constitutes a form of control over the reactions, is not possible and other confinement strategies are needed.
Focusing on thermonuclear controlled fusion, which is of interest for this work, there are

mainly two types of plasma confinement on Earth:

- Inertial confinement: it consists in keeping millimetric pellets of matter at very high pressures ($\sim 10^6$ times the density of air) and temperatures ($\sim 10^8$ T) for a limited amount of time ($\sim 10^{-11}$ s).
- Magnetic confinement: it consists in trapping a considerable volume ($\sim 10^3$ m^3) of plasma with the help of strong magnetic fields (~ 10 T) for a considerable amount of time with respect to inertial confinement (~ 10 s). Magnetically confined plasmas are kept at high temperature ($\sim 10^8$ K) and at low density ($\sim 10^{-5}$ times the density of air).

This work has to do with magnetically confined plasmas. The next sections of this chapter complete the physical introduction for the work being presented in this report.

2.2 Nuclear fusion reactions

The three fusion reactions typically taken into account for applications on Earth involve hydrogen and helium isotopes: deuterium (D), tritium (T) and helium-3 (He3). In particular, they are the D-D, the D-He3 and the D-T reaction. All of them are exothermic and feature subatomic particles among the products:

$$
{}^2_1\mathrm{D} + {}^2_1\mathrm{D} \rightarrow \begin{cases} {}^3_2\mathrm{He} + {}^1_0\mathrm{n} + 3.27\,\mathrm{MeV} \quad (50\%), \\ {}^3_1\mathrm{T}^+ + {}^1_1\mathrm{p} + 4.03\,\mathrm{MeV} \quad (50\%), \end{cases} \tag{2.1}
$$

$$
{}^2_1\mathrm{D} + {}^3_2\mathrm{He} \rightarrow {}^4_2\alpha + {}^1_1\mathrm{p} + 18.3\,\mathrm{MeV}, \tag{2.2}
$$

$$
{}^2_1\mathrm{D} + {}^3_1\mathrm{T} \rightarrow {}^4_2\alpha + {}^1_0\mathrm{n} + 17.6\,\mathrm{MeV}. \tag{2.3}
$$

In each of those, around 80% of the energy released is carried by the subatomic particle in the form of kinetic energy, while the rest is taken by the newly formed nucleus. Assuming that these reactions take place inside a magnetically confined plasma:

- Neutrons, having no charge, can carry their energy outside the plasma. They can be used as the media through which extract energy from the system inside a power plant. However, they could also lead to material activation, meaning that they consititute a radioactivity enabler.
- Protons and nuclei are kept inside the plasma and so does their energy, which can be then transferred to other plasma particles via interactions.

Considering the reactants, deuterium can be easily obtained from seawater, while neither helium-3 nor tritium can be naturally found on Earth. However, it is possible to produce tritium, which is radioactive with a very short half life of approximately 12.3 years, by means of nuclear reactions that involve neutrons and lithium:

$$ {}^{6}_{3}\mathrm{Li} + {}^{1}_{0}\mathrm{n}_{(\mathrm{slow})} \rightarrow {}^{4}_{2}\alpha + {}^{3}_{1}\mathrm{T} + 4.8\,\mathrm{MeV}, \tag{2.4} $$

$$ {}^{7}_{3}\mathrm{Li} + {}^{1}_{0}\mathrm{n}_{(\mathrm{fast})} \rightarrow {}^{4}_{2}\alpha + {}^{3}_{1}\mathrm{T} + {}^{1}_{0}\mathrm{n} - 2.5\,\mathrm{MeV}. \tag{2.5} $$

The process is called tritium breeding and it can take place next to the plasma system, where D-T reactions take place, if lithium is put around it (neutrons are made available from the fusion reactions themselves). Even if it is evident that the D-D reaction is the most appealing in terms of the availability of reactants, the D-T one, thanks to tritium breeding, does not represent a big problem in that regard, too. The most interesting reaction overall is in fact the one involving tritium. The reasons for that are related to two common quantities in nuclear reactor physics: microscopic cross section and reaction rate.

2.2.1 Microscopic fusion cross section

The probability that a certain fusion reaction takes place is closely related to the concept of microscopic fusion cross section σ. It has the unit measure of an area and, given that it is a very small quantity, it is usually expressed in barns:

$$ 1\,\mathrm{b} = 10^{-28}\,\mathrm{m}^2. \tag{2.6} $$

By looking at Fig. 2.2 it can be noticed that, for a fixed energy value, the D-T reaction has almost always the highest cross section. Moreover, it can also be obtained for quite low plasma temperatures compared to the other two, meaning that it is easier to ignite and less energy demanding.

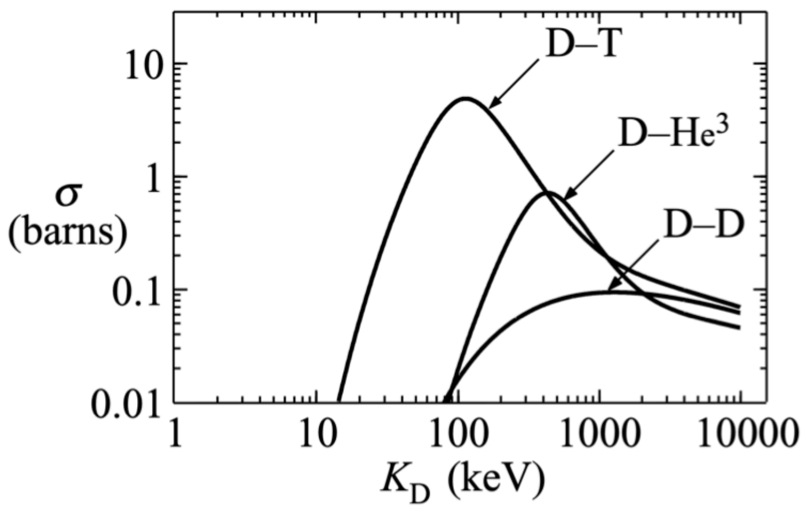


Figure 2.2: Microscopic fusion cross section, expressed in barns, as a function of Deuteron kinetic energy, expressed in keV and that has to be intended as the relative velocity between the nuclei involved in fusion reactions. Taken from [Fre08].

2.2.2 Fusion reaction rate

In a plasma, the number of fusion reactions per unit time and unit volume (i.e. the fusion reaction rate) is directly proportional to $\langle \sigma v \rangle_v$, which is the velocity-averaged product of fusion cross section with the relative velocity between interacting nuclei.
Again, by looking at Fig. 2.3, conclusions similar to the ones for Fig. 2.2 can be derived: the D-T reaction has the highest reaction rate and is also available for lower plasma temperatures than the others.

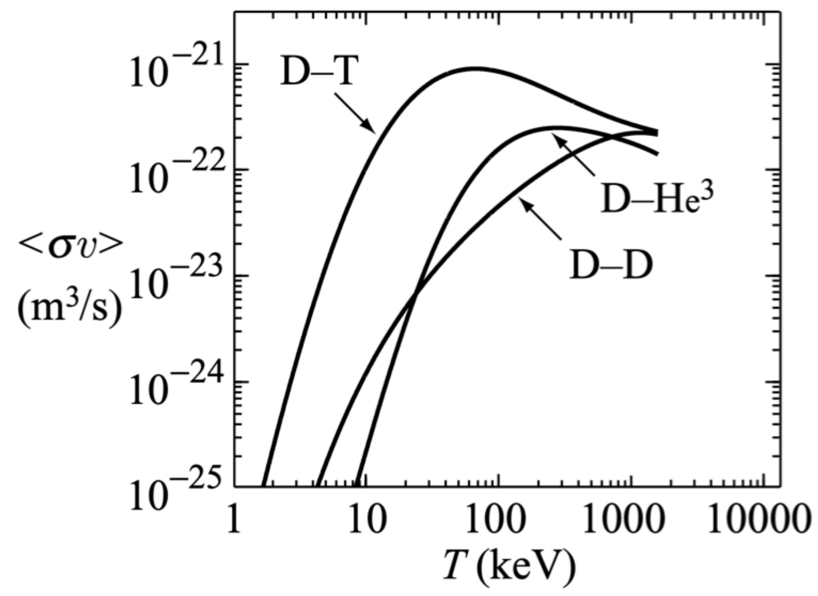


Figure 2.3: Velocity-averaged product between fusion cross section and the relative velocity of the interacting nuclei, expressed in m^3/s, as a function of temperature. A Maxwellian distribution has been assumed for the velocity. Taken from [Fre08].

The plasma in which fusion reactions take place inside experimental nuclear reactors has to be properly controlled and confined, because otherwise fusion conditions are easily lost. In order to understand how this can be achieved, the behavior of charged particles interacting with electromagnetic fields has to be investigated. This is important because it will also lead to the reason why fusion reactors have certain shapes and characteristics.

2.3 Charged particles in electromagnetic fields

As already mentioned, the nature of plasma puts it in the condition of being able to interact with electromagnetic fields. In order to describe how plasma magnetic confinement is obtained in practice, it is convenient to start by analyzing the dynamics of single charged particles. Given a particle of mass m and charge e inside an electric field $\mathbf{E}$ and a magnetic field $\mathbf{B}$, its position $\mathbf{r}$ at any given time t can be obtained, for example and simplicity, by solving this initial value problem:

$$\begin{cases} m\dfrac{d^2\boldsymbol{r}(t)}{dt^2} = e\Bigg(\boldsymbol{E}(\boldsymbol{r},t) + \dfrac{d\boldsymbol{r}(t)}{dt} \times \boldsymbol{B}(\boldsymbol{r},t)\Bigg), \\ \\ \dfrac{d\boldsymbol{r}(t)}{dt}\bigg|_{t=0} = \boldsymbol{v_0}, \\ \\ \boldsymbol{r}(0) = \boldsymbol{r_0}. \end{cases} \tag{2.7}$$

Given that this work focuses on fusion applications on Earth, in problem 2.7 the effect of gravitational force is so small that it is neglected (this would not be the case for stars). In one of the simplest possible cases, when $\mathbf{E} = \mathbf{0}$ and the particle is only subjected to a constant and uniform magnetic field, it exhibits and helicoidal motion. In particular, it translates along magnetic field lines while rotating around them.

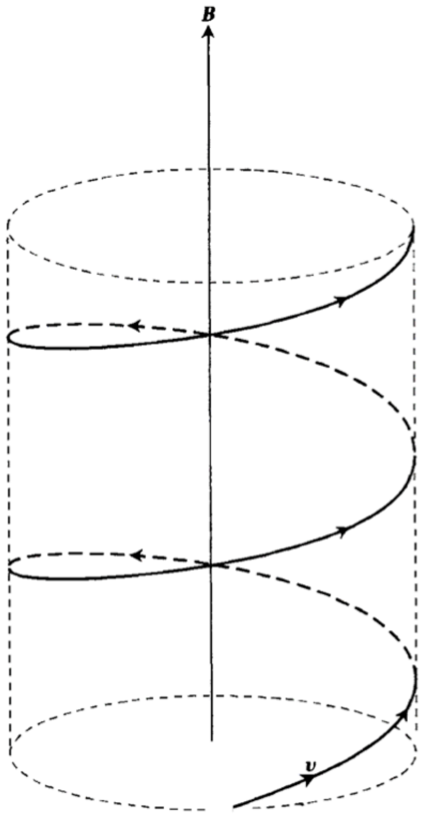


Figure 2.4: Helicoidal motion of a negatively charged particle with initial velocity neither parallel nor perpendicular to the unidirectional magnetic field lines. Taken from [Den90].

The rotational component of motion is associated with a radius, called Larmor radius or gyroradius, which depends on e, m, the magnitude of $\mathbf{B}$ and the component of the velocity of the particle perpendicular to magnetic field lines:

$$\rho = \frac{mv_\perp}{|e|B}. \tag{2.8}$$

If, for example, a Cartesian system of coordinates $S = (x, y, z)$ is taken into account, and magnetic field lines are only along the z-axis, then the perpendicular velocity at time t in the position (x, y) of the particle in the transverse plane is:

$$v_\perp = \sqrt{v_x^2 + v_y^2}. \tag{2.9}$$

The central point around which each particle rotates is called guiding center. The angle of gyration (gyroangle) of the particle around its guiding center is usually labelled as θ. In this particular case, the guiding center translates in the direction of the magnetic field, but in more complex configurations it could move in a different way. The rate at which particles rotate around their guiding centers is the absolute value of a quantity named Larmor frequency and indicated with Ω:

$$\Omega = \frac{eB}{m}. \tag{2.10}$$

The sign of Ω, which depends on e, indicates the direction in which the particle is rotating, i.e. clockwise or counter-clockwise, as it is shown in Fig. 2.5.

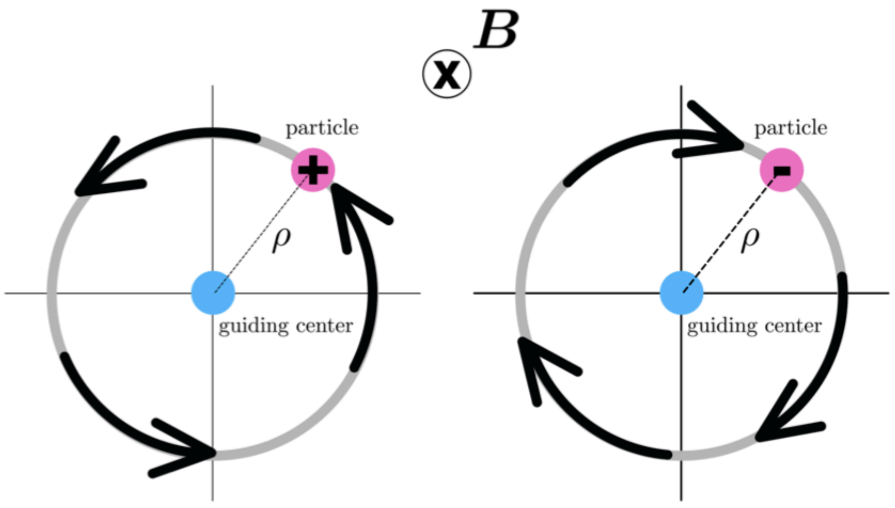


Figure 2.5: Directions of rotation, in the transverse plane, of positively and negatively charged particles when magnetic field lines are unidirectional and ideally entering the page.

2.3.1 Confinement of charged particles

A simple magnetic field, like the one just taken into account, is therefore capable of simultaneously confining multiple charged particles in the transverse plane, but not in the direction of magnetic field lines. In order to achieve total confinement, i.e. including also the third direction, as it is needed for fusion reactors, it could be thought that the solution would be to move from a unidirectional magnetic field to a circular one. Magnetic field lines would then describe a torus in this new congifuration. However, this modification complicates the dynamics of charged particle and leads to a trochoidal motion, which is again not associated with total confinement: a translation across magnetic field lines, called drift, is added to the helicoidal motion. This drift is caused by a spatial gradient in the magnetic field: being ring-shaped means that it is more intense on the inside than on the outside. Because of its nature, it is also more properly known as $\nabla\mathbf{B}$ drift.
Moreover, if multiple particles are considered at once, the drift would result in a separation of charges: positive and negative particles move across magnetic field lines in opposite directions. Such effect would then in turn result in the formation of an electric field which would determine a complete loss of confinement under the effect of a new drift, called $\mathbf{E} \times \mathbf{B}$ drift.
The idea is then to twist the toroidal magnetic field lines so that they become helices. In other words, a rotational transform needs to be applied. Once in this configuration, which is usually known as magnetic bottle, drift of single particles and global charge separation are largely countered. Therefore, in theory, a magnetic bottle is capable of simultaneously confining multiple charged particles. However, in practice, it is not granted that magnetic bottles alone can properly confine plasmas: these systems are made up of so many particles that microscopic electromagnetic interactions can add up to produce global effects that go against magnetic confinement. In fusion reactors, this would lead to a position in which fusion can not take place because temperatures would become too low and plasma state would be lost. In any case, having an helicoidal magnetic field is one of the basic mandatory requirements for magnetically confined and closed fusion machines. In the following, few relevant experimental fusion reactors are briefly described, with a focus on their magnetic fields and how they are generated. Moreover, since in this work the attention is put on tokamaks, a small section is dedicated to its instabilities and turbulent regimes.

2.4 Experimental fusion reactors

2.4.1 Tokamaks

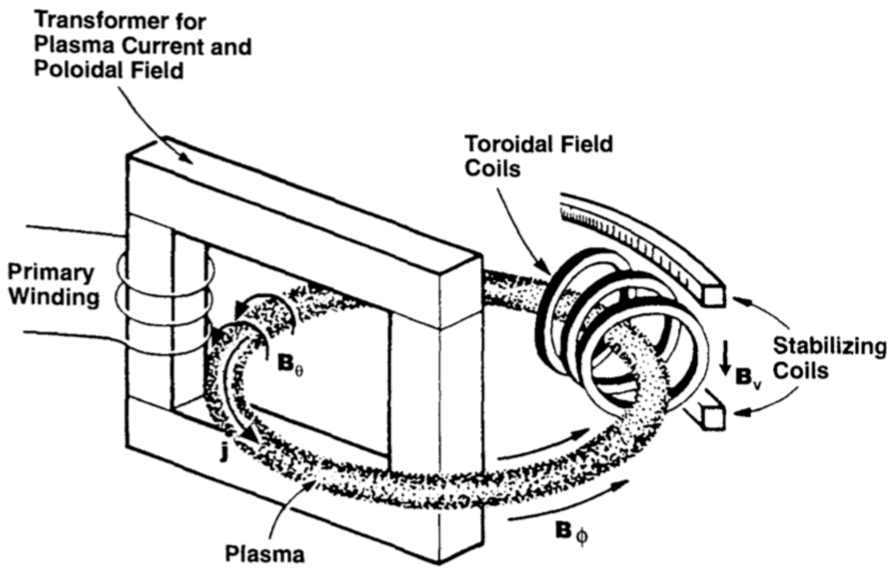


Figure 2.6: Schematic representation of the main components of a tokamak and its magnetic field. Taken from [HSK00].

Nowadays, the tokamak is the most famous device for magnetically confined fusion. It has a toroidal shape, much like its helicoidal magnetic field lines wrap around a torus. Its magnetic field is the result of the superposition of two magnetic fields of different nature:

- A toroidal magnetic field $\mathbf{B}_\phi$, which is generated outside the plasma by ring-shaped vertical magnets called toroidal field coils. Its field lines surround the torus hole.
- A poloidal magnetic field $\mathbf{B}_\theta$, which is obtained with an electromagnetically induced electric current $\boldsymbol{j}$ inside the plasma. In order to create this large axial current, the working principle of a transformer is used: the plasma is the single-winding secondary, while a current flows in the primary winding playing the role of the inductor. In practice, this transformer action can be obtained with or without an iron core. In the latter case, there is a central field coil that works as the primary winding.

On top of these two components, a third magnetic field $\mathbf{B_v}$, generated by horizontal circular magnets called poloidal field coils, is there to provide a control tool for the tokamak. Its presence is vitally important because it can contrast the natural tendency of a torus-shaped plasma to outward expand. In general, this vertical field is associated with the equilibrium, the position and the shape of the plasma, other than being capable of controlling its induced current. These three fields are such that the poloidal one is the stronger, followed by the toroidal and then by the vertical one. Tokamaks with the above characteristics can only work in pulsed mode, and not in steady-state, because the inductive nature of plasma current means it can only be created in presence of time-varying quantities.

2.4.2 Stability of Tokamaks and plasma turbulence

As already mentioned, in practice magnetic confinement is not applied to single particles or to a relatively small group of them, but it is instead associated with a plasma, which contains an enormous amount of interacting charged particles. This complex ionized gas can very easily be associated with instabilities, both on macroscopic and microscopic scales, and become turbulent. In general, instability and turbulence in conductive fluids have fluid and electromagnetic natures. This is not the case with non-conductive fluids like normal gases and liquids.

There is a group of instabilities that can arise when a conductive fluid is moving inside a magnetic field. They are called magnetohydrodynamic instabilities and, if uncontrolled, can cause the plasma to move closer to the machine edges. This can have a huge impact on the quality of plasma confinement and on temperature, but can also determine the complete loss of the plasma state. These macroscopic instabilities happen to be heavily linked with plasma current and pressure, which therefore need to be kept below certain threshold values.

Then, there are instabilities, which are not suppressed when the above measures are taken, that are related to plasma gradients of various nature. In general, when spatial gradients of plasma temperature, plasma pressure or magnetic field become too high, microscopic instabilities of mostly electrostatic nature appear. Even if they have the tendency to not significantly influence magnetic fields, they can however add up and threaten plasma confinement and fusion reactions in the sense that they determine the formation of a turbulent state. Once in this state, particles with different temperatures and coming from different plasma regions are mixed, resulting in transport issues regarding particles and energy in terms of how well the plasma is kept under confinement. In tokamaks, fusion only takes place in the hottest part of the plasma, which by the way occupies only a very limited volume of the plasma chamber. This core region is separated from the solid parts of the fusion device, i.e. the tokamak walls, by colder plasma. Therefore, given that turbulence can facilitate mixing between these two plasma regions, fusion conditions are more difficult to be met or kept for an appropriate amount of time. In this sense microinstabilities, by means of turbulence regimes they create, can lead to thermal isolation problems inside the plasma. Further sources of instability are represented by the magnetic configuration itself, which has a direct influence on how particles interact and, therefore, on their velocity distributions. Being more specific, non-Maxwellian velocity distributions can cause microinstabilities: limiting interactions between particles reflects on the system ability to reach or maintain an equilibrium distribution.

2.4.3 Stellarators

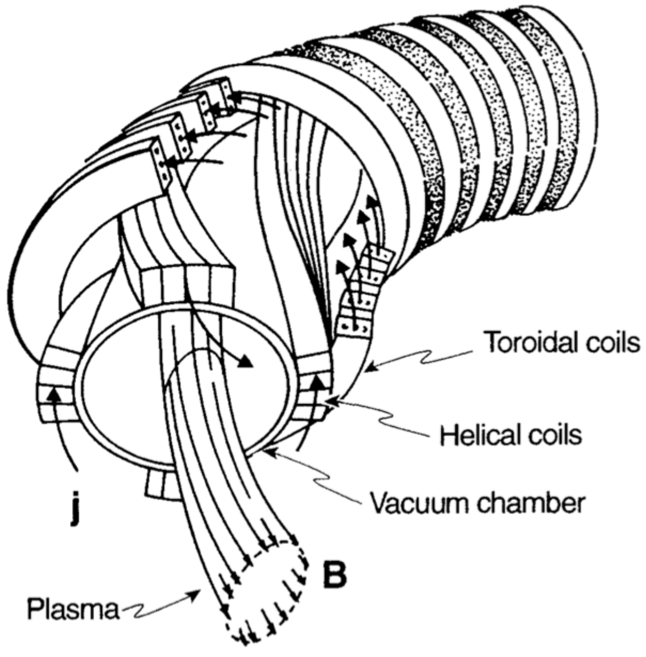


Figure 2.7: Schematic representation of a section of a classic stellarator. Taken from [HSK00].

Another widely known fusion device is the stellarator. Differently from a tokamak, its helical magnetic field is entirely obtained using direct currents flowing in conductors that surround the plasma.
A classic stellarator has the toroidal field component generated by toroidal coils, like in a tokamak. However, while tokamaks use plasma currents to generate the poloidal component, stellarators have coils surrounding the vacuum chamber for that. In theory, such windings would alone provide both components for the helicoidal magnetic field, however they would also produce an undesired vertical field. Therefore, to avoid that, adjacent helical windings require their currents to flow in opposite directions: this eliminates the vertical component, but also the toroidal one. It is then clear that toroidal coils are necessary to keep a toroidal component and, therefore, to ensure the presence

of a global helical magnetic field. More advanced stellarator configurations realize their plasma confinement with modular non-planar twisted coils (see Fig. 2.8). Their bizarre shape makes them able to simultaneously produce both toroidal and poloidal components of a particularly stable helical magnetic field, which however is different from the one of tokamaks. In fact, when the two fields are compared from above, the one of tokamaks resembles a circle, while the one of advanced stellarators is more similar to a polygon. Moreover, not having to depend on induced plasma currents means that stellarators can work in continuous mode, which would be more suitable for pacific energy production.

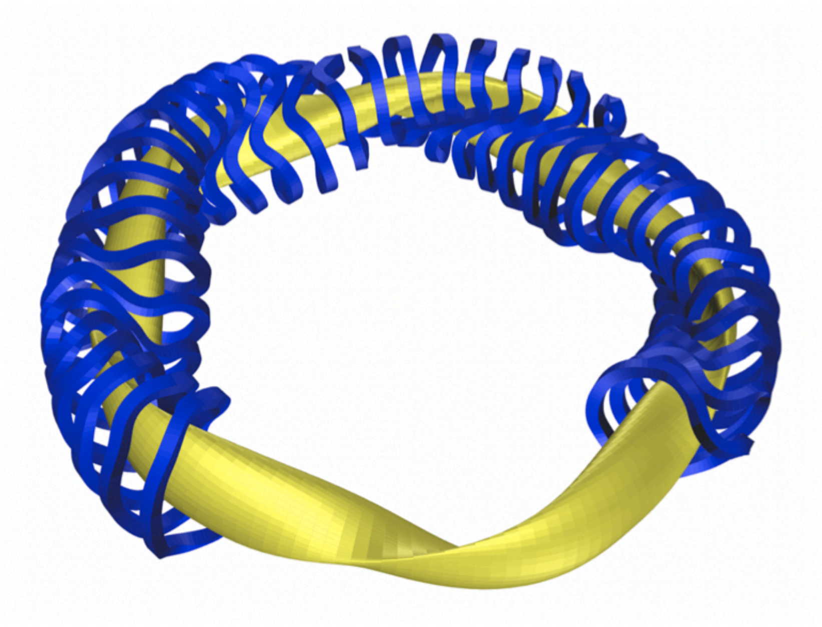


Figure 2.8: Schematic representation of an advanced stellarator. Magnets are in blue, while the magnetic field is in yellow. Taken from [BK12].

2.4.4 Reversed Field Pinches

The reversed field pinch (RFP) is a toroidal configuration similar to tokamaks, since it involves plasma currents to produce its magnetic confinement:

- The toroidal field $\mathbf{B}_\varphi$ is obtained like in tokamaks, but it is weaker and altered by the strong plasma current generated using trasformer action. This alteration, which occurs spontaneously only above certain plasma current intensities, determines a sign change of $\mathbf{B}_\varphi$ known as field reversal: the toroidal magnetic field at the plasma center points in a different direction than at the plasma edge. This condition results in an increased level of plasma stability, other than being associated with a higher plasma temperature.

- The poloidal field $\mathbf{B}_\theta$ is much stronger than in a tokamak and is provided by the same current responsible for field reversal.

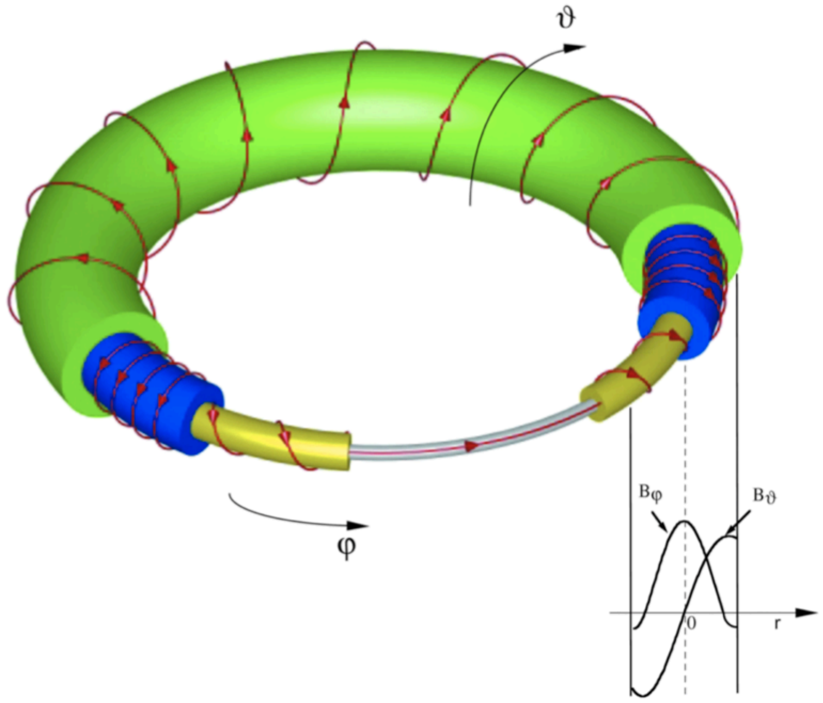


Figure 2.9: Schematic representation of the magnetic field inside a Reversed Field Pinch. Red lines wrapped around the yellow surface are pointing in a different direction than the ones around the green surface (field reversal). Taken from [RFX].

Similarly to tokamaks, a third component of the magnetic field is provided by field shaping coils and is intended to have control over plasma equilibrium. The main advantage of RPFs, compared to tokamaks and stellarators, is that a less intense external magnetic field is required to meet fusion conditions and therefore their coils can be simpler.

2.5 Equations to describe plasma dynamics

Plasma dynamics can be studied with different kinds of equations, which are associated with different levels of description. Depending on various factors, they can be more or less suitable for numerical implementations and for an adequate description of certain phenomena. The two main physical aspects that have to be kept in mind when dealing with plasmas are charged particles and electromagnetic fields. Particles interact with one another inside the plasma, but also with electromagnetic fields, which in practice are the superposition of external fields and fields produced by the plasma itself. Levels of description means that plasma can be investigated by considering every charged particle of it, or using distribution functions, or even by treating it as a fluid that is capable of interacting with electromagnetic fields and that is globally quasi-neutral.

2.5.1 Equations of motion for charged particles

The most immediate way to approach plasma description is to use the equations of motion for every particle, like what has been displayed in the system of equations 2.7. These equations are coupled with Maxwell's equations to account for the feedback that particles have on their superimposed electromagnetic field. Their solution allows one to know the exact position $\boldsymbol{r}(t)$ and velocity $\boldsymbol{v}(t)$ of each particle in the system at any given time t. While equations in this case are very straightforward, even more so if simplified cases with no feedback are considered, there would be an equal number of vectorial equations and particles: this is not something that computers nowadays can handle for plasma systems, given that, for example, they can have particle densities of the order of $10^{20}\ m^{-3}$.

2.5.2 Kinetic approach

A possibility is then to pass to statistical mechanics: starting from the equations of motion of single particles, for example in Hamiltonian form, it is possible to derive the so called kinetic equations. The solution for the most used of these equations is usually represented by the distribution function for one particle, commonly named just distribution function and labelled as $f(\boldsymbol{r}, \boldsymbol{v}, t)$. This quantity gives information about the expected number of particles that have position inside $[\boldsymbol{r} + d\boldsymbol{r}]$ and velocity inside $[\boldsymbol{v} + d\boldsymbol{v}]$ at an instant of time in $[t + dt]$. Kinetic equations, in general, can have much more readable and interpretable results than the previous description. However, in the form presented up to here and in relation to plasmas, they are still quite challenging for computers to deal with, given that $f(\boldsymbol{r}, \boldsymbol{v}, t)$ is a function of seven variables. Starting from distribution functions, it is then possible to obtain a series of useful quantities, such as plasma temperature. Among kinetic equations used for magnetically confined plasmas, some of the most relevant are the Vlasov-Maxwell equations, which in essence consist in the Vlasov equation coupled with Maxwell's equations to account for electromagnetic feedback.

2.5.3 Fluid description

In order to further reduce the required computational effort, losing however on generality and equations simplicity, it is possible to use fluid equations, which can be directly derived from kinetic equations. The simplest and most widely known fluid model for magnetically confined plasmas is called magnetohydrodynamics (MHD). In this description, plasma is globally seen as an electrically conductive fluid. MHD equations are associated with fluid equations and Maxwell's equations and allow one to describe those already mentioned macroscopic instabilities that can lead to a complete loss of confinement. Solving fluid equations allows one to directly know a series of mean quantities that are related to the plasma as a whole. For example, a simple set of MHD equations, when solved, gives information about the plasma density $\varrho(\boldsymbol{r}, t)$, the plasma velocity $\boldsymbol{V}(\boldsymbol{r}, t)$, the pressure $P(\boldsymbol{r}, t)$ and the magnetic field $\mathbf{B}(\boldsymbol{r}, t)$. By looking at these quantities, it is immediately clear that they are simpler than $f(\boldsymbol{r}, \boldsymbol{v}, t)$ to compute and analyze, given that they are functions of just four variables instead of seven.
One of the main restrictions associated with MHD equations is that the plasma needs to be strongly collisional, i.e. the velocity distribution for its particles ideally is a Maxwellian. In tokamaks, this limits their applicability to the plasma edges, since in the hottest areas, where fusion reactions take place, there is no such equilibrium distribution.

2.5.4 Gyrokinetics framework

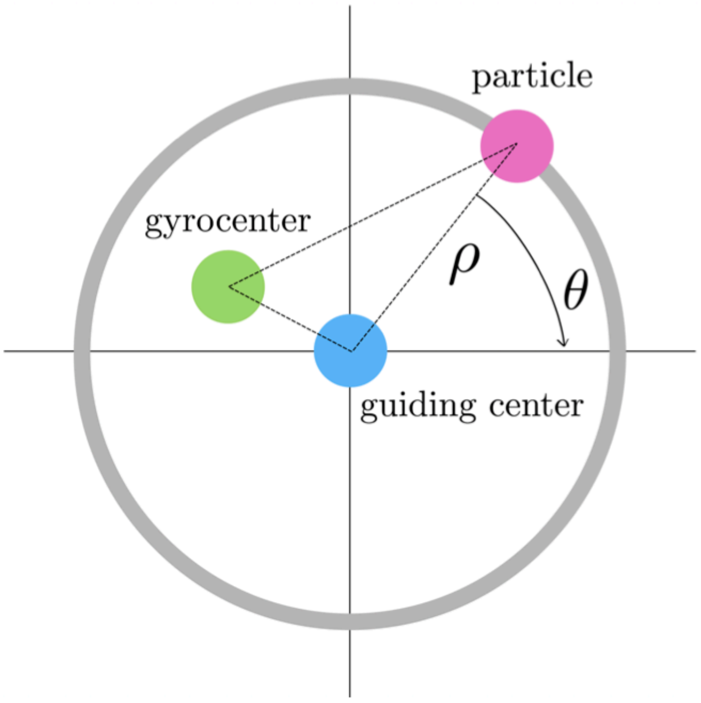


Figure 2.10: Comparison between positions, in the transverse plane, of a particle, its guiding center and the gyrocenter. Inspired by [GL21].

Another way of obtaining more tractable equations without moving away from a kinetic description, is to use a reduced set called gyrokinetic equations. When applying the gyrokinetic description, the area of interest is usually represented by low frequency phenomena in strongly magnetized plasmas, i.e. phenomena with characteristic times that are slower than the period associated with the rotational component of motion of charged particles around magnetic field lines. With respect to kinetic equations, which are related to the particles themselves, gyrokinetic theory and its distribution functions have to do with gyrocenters, which in essence are modified guiding centers that account for low-frequency electromagnetic fluctuations produced by the plasma itself.
Moving from the particles to their guiding centers and then to the fictional gyrocenters is mathematically challeging and strongly relies on multiple changes of coordinates and an averaging procedure over the gyroangle θ. However, once all the steps are performed, there is a concrete gain in the sense that the new distribution function reduces to having just six variables, since the gyroaveraging procedure just mentioned effectively removes one of them:

$$f(\boldsymbol{r}, \boldsymbol{v}, t) \longrightarrow f_{\mathrm{gy}}(\boldsymbol{r}_{\mathbf{gy}}, v_{\parallel \mathrm{gy}}, \mu_{\mathrm{gy}}, t), \tag{2.11}$$

where $\boldsymbol{r}_{\mathbf{gy}}$ is the position of the gyrocenter, $v_{\|\mathrm{gy}}$ is the component of its velocity that parallel to magnetic field lines and μ_{gy} is the magnetic moment of the gyrocenter, which is associated with the one of the particle:

$$\mu = \frac{mv_{\perp}^2}{2B}. \tag{2.12}$$

Not only having reduced the variables simplifies the whole scenario, but it also drastically reduces computational time during simulations with respect to the kinetic approach. The main reason for that is that the fast rotational component of motion is not involved in the computations: in essence, the gyroaveraging procedure has this effect from a numerical point of view. One of the main gyrokinetic equations is derived as a dynamical reduction of Vlasov-Maxwell equations.
Given that gyrokinetics is more closely related to particles than fluid descriptions, it is usually used to study the turbulent dynamics happening in the plasma core of fusion devices.

2.5.5 Guiding center approach

As briefly hinted, an intermediate step in gyrokinetics is associated with guiding centers. This work has guiding center theory (see [Lit80])) as its main component. As it will be later clarified, the intent here is to investigate some test particles in order to obtain information about particle transport in magnetically confined plasmas under an electrostatic turbulent regime. To do that, the equations used to describe plasma are related to single particles, but they are changed through a gyroaveraging procedure so that the point of view is the one of guiding centers. While embracing this particular guiding center approach may seem time consuming since distribution functions are not involved, if its application is properly limited to test particles, which constitute a small fraction of an actual plasma, then that makes it reasonable.
An additional source of time saving and complexity reduction is created when it is possible to consider test particles in just two dimensions, which are usually associated with the plane perpendicular to magnetic field lines. This possibility opens up when the particles move in a trivial way along the third direction or when that particular component is of little to no interest. Even at this simplified stage of following very few sample trajectories, the computational gain offered by the gyroaveraging is well evident.

Chapter 3

Description of the problem and theoretical part

It was decided to do this work because in could have been interesting in relation to the way gyrokinetic theory is built. Gyrokinetic equations are widely used to study fusion plasmas, with a focus on their turbulent regimes and transport processes. As discussed in subsection 2.5.4, they involve a gyroaveraging procedure that eliminates θ from the variables. This particular step is made possible from the use of a canonical change of coordinates represented by the Lie transform, which is outlined in section A.5 of Appendix A. This transformation is associated with expressing coordinates and, more in general, observables, as series. Nowadays, the sets of coordinates that are used in relevant gyrokinetic codes are obtained only by considering the first two terms of these series, i.e. the zeroth and first order ones. This first order gyrokinetic approach is chosen because it seems that higher orders are of really low relevance and are therefore negligible. On top of that, their implementation would complicate the equations and significantly increase the time required to complete simulations.
However, in parallel, it is evident that fusion devices still experiment particular diffusion processes that go against magnetic confinement and therefore the possibility to achieve a proper controlled fusion.
The idea was then to try to create a watered-down physical model of a plasma in a tokamak and see if adding the second order term of these series had some relevant effect, possibly leading to a better understanding of particle transport in fusion, further research and the complete or partial implementation of second-order terms in gyrokinetics codes. In other words, this work just started with the willingness to explore what adding a previously neglected term could do, but, in practice, it also opened up to other interesting aspects which were not expected and that will be extensively presented later.

3.1 Summary of what has been done

This work is based on [Cha21] for the theoretical part and on the original version of three Python scripts that are associated with it (they have been updated and can be found at `github.com/cchandre/Guiding-Center/`). In essence, they implement the equations of guiding center dynamics obtained in [Cha21] (see Eq. 3.93) in dimensionless form and with a specific electrostatic potential (see Eq. 3.152) that mimics the ones associated with direct measures inside tokamaks [Pet+88]. They allow one to perform simulations on test particles inside a superimposed electromagnetic field. They have been written by Cristel Chandre, who is the Research Professor of the Centre National de la Recherche Scientifique (CNRS) at the Institut de Mathématiques de Marseille.
For this work, I have collaborated with Cristel Chandre himself and Professor Guido Ciraolo, who currently leads the Groupe Thèorie et Simulation Numérique (GTSN) at the Institut de Recherche sur la Fusion par Confinement Magnétique (IRFM) of the CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives) at Cadarache. I worked in direct contact with them during my stage in France at the IRFM.
In short, this is what I have done:

- I studied the main features of tokamaks, charged particles in electromagnetic fields and guiding center theory. This part was facilitated by my prior university studies in electromagnetism, particle and radiation transport, plasma physics and, more in general, nuclear engineering.

- I got acquainted with the mathematical tools needed for this work, which essentially consist in Hamiltonian mechanics, changes of coordinates, Lie Transforms and Discrete Fourier Transforms. University studies in mathematical methods for energetics, analytical mechanics and heat transfer were very useful in that regard. In particular, it was very important for me to have prior knowledge on aspects like Hamiltonian functions, nondimensionalization of equations, Fourier Analysis and Bessel functions.

- I learned to read, write and remotely launch Python scripts. I improved my skills on how to read files and analyze data with MATLAB and I learned how to write more efficient lines of code, for example by substituting `for` cycles with specific functions or by avoiding unnecessary repetitions. My studies on numerical methods for energetics and on modelling and simulation techniques for energetics gave me a relevant background on multiple aspects, such as algorithms, discretization and numerical implementation of equations. Moreover, at university, I did multiple projects that allowed me to familiarize more rapidly with Python:

  - I extensively used MATLAB for projects on computational heat transfer, thermofluid dynamics and control theory. In addition to that, I used MATLAB for post-processing of data coming from Monte Carlo simulations associated with radiation protection. The vast majority of what I know about coding from scratch comes from these experiences.

  - I used the particle-in-cell code XOOPIC for a project on helicon plasma thrusters. It was helpful for this work because I learned to work by understanding and modifying already existing scripts, other than consulting manuals.
  - I used the Monte Carlo codes PENELOPE (written in FORTRAN) and OpenMC (written in Python) for radiation protection problems related to X-rays and neutrons. They were particularly helpful because they taught me the relevance of the selection of parameters and trained me to find ideas on how to improve coding in order to save computational time. OpenMC was also my first contact with Python, but it was a really different experience with respect to the work of this report.

While it is true that I exploited my prior knowledge, I still had to make many efforts in order to embrace a huge amount of new physical, mathematical and numerical tools and concepts. From a purely personal point of view, this work allowed me to improve under multiple aspects.

- Moving onto more practical things, I made sure that the derivation in a Hamiltonian framework of the equations for guiding center dynamics reported in [Cha21] did not contain any mistake. In particular, I re-derived them from scratch using a very similar approach to the one that is presented in [TC18] and I mixed it with the short number of steps available in [Cha21]. I produced a detailed and longer description on how to obtain those equations, which is presented in its entirety in this Chapter. It was entirely obtained by hand, but for a part regarding changes of coordinates I used some short MATLAB scripts I wrote.
- The generic electrostatic potential that appears in the equations for guiding center dynamics was more specifically defined on the basis of [Pet+88] and with the help of some handwritten notes made by Cristel Chandre. I checked them, too.
- The last part of the derivation was better characterized by adding an explicit nondimensionalization procedure of quantities and equations. It was partially inspired by [Pet+88] and [Cir+04], but it was almost entirely done from scratch.
- I analyzed and understood the Python scripts associated with the theoretical part. I found a small error in one the files, which is discussed in subsection 4.8.2.
- I launched many different simulations to test the scripts in their original form and I wrote some entirely new scripts in MATLAB to analyze the outputs. MATLAB was

very important as a feedback tool because it allowed me to suggest and implement some improvements in Python to save computational time and, in general, to better match the nature of the physical problem that has been studied. As the scripts evolved in Python, they did the same in MATLAB, up to the point where final forms were reached in both cases. This part took the vast majority of the time I spent on this work and served also to properly choose the values (or the range of values) of the various parameters that characterize the Python scripts.

- With the final versions of the scripts in Python, I launched some simulations whose results have been analyzed in MATLAB from physical and numerical points of view, without the need to use them as feedback or diagnostic tools anymore. Even though most of the results were obtained from these last simulations, some relevant and interesting information about the modelled physical system were found even during the time spent on improving the scripts.

The theoretical aspects are dealt with in this Chapter (and in the Appendices), while the scripts, the way they were improved and the analysis of the outputs are extensively discussed in Chapters 4 and 5.
The choice of using different programming languages has been made because Python is free and relatively fast if used properly, but MATLAB is generally better (and easier for me to use) when dealing with post-processing and data plotting.
The portions of the Python scripts that have been modified are discussed only after the presentation of their final versions in order to better highlight what has been implemented, even at the cost of potential partial overlapping of contents.

3.2 Description of the problem

3.2.1 General outline of the electromagnetic field

This work exploits guiding center theory and revolves around investigating the turbulent motion of charged test particles driven by electromagnetic fields that approximate, to some extent, the ones that can be found in tokamak devices. In particular, this 3D field consists in the the combinations of:

- A constant and uniform magnetic field:

$$\mathbf{B} = B_0\hat{k}. \tag{3.1}$$

- A 2D turbulent electric field, defined through an electrostatic potential which varies with time only in the plane transverse to the magnetic field lines (the transverse plane):

$$\mathbf{E} = -\nabla\Phi(x, y, t). \tag{3.2}$$

Cartesian coordinates have been used here ($\hat{k}$ is the unit vector associated with the z-axis). The magnetic field can be seen as a very basic simplification of the true helicoidal field lines associated with plasma confinement in tokamaks, but can also be interpreted as an approximation of such spiral lines if a limited portion of space is considered, so that curved and straight lines are almost identical. Such an approximation for the magnetic field suits better tokamaks whose vessels have large aspect ratios, and it is also done to decouple the different drift velocities from one another.
This particular choice of the magnetic field helps to keep the equations relatively simple both to derive and code, while also allowing to more effectively see the effect that varying the values of parameters has on the results. However, this is one of the points that could be very well developed in further works so that a more realistic condition is met.
Moving onto the electric field, it is mathematically built in such a way that it resembles what can be directly measured inside tokamaks as effect of plasma microinstabilities (when the magnetic field is helicoidal and there is even the effect of stabilizing coils). As it will be extensively presented later, it is defined via its electrostatic potential, which in turn is represented as a truncated Fourier series. The specification of $\Phi(x, y, t)$ will be done only after having completed the derivation of the equations for guiding center dynamics.

3.2.2 Specifications on the test particles

The test particles that are studied in this electromagnetic environment can be thought of as either samples of the plasma responsible for the electric field, or impurities (which are part of the plasma too, even if in an unwanted manner). In any case, they are associated with some important hypotheses and considerations:

- They no feedback on the electromagnetic field. This is an approximation that has the considerable advantage of not having to consider couplings with Maxwell's equations, and it is also not that far from reality. In fact, given that part of the electromagnetic field is caused by the plasma itself and is modelled on the basis of real measurements, it is as if the feedback is already taken into account with the particular choice of the electric field. In this sense, test particles can be simply seen as specific parts of the plasma that are highlighted during the simulations, emerging from the hidden plasma that would otherwise manifests itself only through its electric field.

- When multiple trajectories are simulated at the same time, the particles do not *see each other*. There is nothing in the equations or in the the associated scripts that considers that. Again, what is discussed in the previous point suggests something that can also serve as a justification for this one.

- Even in the case that, for some reason, either of the previous two points turns out to be too unrealistic, the fact that very few test particles are simulated (compared to the typical particle densities in fusion plasmas) means that the approximations would have limited relevance;

- Test particles are actually studied by simulating the motion of their guiding centers. Almost all the mathematical efforts prior to the coding part has to do with this specific choice, which however turns out to be very powerful.

3.2.3 Reduction from three to two spatial coordinates

As it will become clear in the following parts of this report, another important feature of this problem is that it can be coded and treated in such a way that it is reduced to considering just two spatial dimensions out of three. Guiding centers are then individuated only by two spatial coordinates, i.e. the ones associated with the plane perpendicular to the direction of magnetic field lines. In short, this is possible because of two aspects: the position of guiding centers or particles along the direction of magnetic field lines is both trivial and of little interest. In fact, what matters is how far particles move from their initial position in the transverse plane. If a certain particle remains close to its initial transverse position as time passes, then it means that it is properly confined, otherwise it is possible that, for that particular particle, the turbulent regime has reduced the effectiveness of confinement offered by the magnetic field.
If the system of Eqs. 2.7 is considered only for the component of $\boldsymbol{r}$ that is parallel to the magnetic field lines, i.e. z (in a Cartesian frame of coordinates), and Eqs. 3.1 and 3.2 are used, then it is possible to write:

$$\begin{cases} \dfrac{d^2 z(t)}{dt^2} = 0, \\[2ex] \left.\dfrac{dz(t)}{dt}\right|_{t=0} = w_0, \\[2ex] z(0) = z_0, \end{cases} \tag{3.3}$$

where w_0 is the component of the initial velocity that is parallel to the magnetic field lines. The solution is trivial and is given by:

$$z(t) = w_0 t + z_0. \tag{3.4}$$

In other words, the position of a particle (and, consequently, of its guiding center) along z at time t is always straightforward to determine, provided that z_0 and w_0 are known (they are both needed at least in this particular formulation of the problem, in which a second order derivative of z with respect to time has been considered).

3.3 Obtaining the equations of guiding center dynamics in Hamiltonian formalism

With regard to the theorical portion of this work, it can be ideally split into three parts. In the very first one, the given equations that describe guiding center dynamics have been checked by extensively re-deriving them from a canonical Hamiltonian system associated with a single charged particle inside a generic and externally imposed electromagnetic field. This was long and necessary, but apart from its intrinsic relevance it could also be helpful for similar works in the future: it has allowed to outline, later in this report, a very precise list of mathematical steps that are often not displayed in full length and that could be used by the reader as a tutorial. This same idea of expliciting as many details as possible is also repeated for the remaining two theoretical parts.
Anyway, after all of this, the second step has consisted in validating the correct implementation of the true form of the electrical field, i.e. the truncated Fourier series. In the third part, the procedure to make the equations and the parameters dimensionless has been carried out from scratch (very little information was available on this), but in such a way that everything still worked out properly. In other words, this means that if this last part was removed, the Python scripts and would have required no modifications at all: the results of the simulations would have just needed to be accompanied by units of measurement.

3.3.1 Canonical setting

The starting point to obtain the guiding center equations is the equation of motion for a single charged particle in an electromagnetic field and its associated Hamiltonian in canonical formalism. The relevant aspects of Hamiltonian mechanics for this work are briefly summarized in Appendix A.
The very first step to do is to select a generic 3D orthonormal basis $\mathcal{B}$ that helps to describe the space where the particle moves:

$$\mathcal{B} = (\hat{\boldsymbol{b}}_{\mathbf{1}}, \hat{\boldsymbol{b}}_{\mathbf{2}}, \hat{\boldsymbol{b}}_{\mathbf{3}}). \tag{3.5}$$

Given that there are $\mathscr{N} = 3$ degrees of freedom and that the focus is just on a single particle, a total of $2\mathscr{N} = 6$ canonical variables is needed: three positions and three conjugated momenta. They can be respectively written in this form:

$$\boldsymbol{q}(t) = \big(q_1(t), q_2(t), q_3(t)\big) = q_1(t)\hat{\boldsymbol{b}}_{\mathbf{1}} + q_2(t)\hat{\boldsymbol{b}}_{\mathbf{2}} + q_3(t)\hat{\boldsymbol{b}}_{\mathbf{3}}, \tag{3.6}$$

$$\boldsymbol{p}(t) = \big(p_1(t), p_2(t), p_3(t)\big) = p_1(t)\hat{\boldsymbol{b}}_{\mathbf{1}} + p_2(t)\hat{\boldsymbol{b}}_{\mathbf{2}} + p_3(t)\hat{\boldsymbol{b}}_{\mathbf{3}}. \tag{3.7}$$

In the following, the implicit time dependencies appearing in Eqs. 3.6 and 3.7 will be omitted for simplicity. The canonical variables can be gathered to form a single vector:

$$\boldsymbol{z} = (\boldsymbol{q}, \boldsymbol{p}). \tag{3.8}$$

This will soon become useful. It is now possible to define both a magnetic and an electric field with the help of potential functions (see [Str07]):

$$\boldsymbol{B}(\boldsymbol{q},t) = \nabla \times \boldsymbol{A}(\boldsymbol{q},t), \tag{3.9}$$

$$\boldsymbol{E}(\boldsymbol{q},t) = -\nabla\Phi(\boldsymbol{q},t) - \frac{\partial \boldsymbol{A}(\boldsymbol{q},t)}{\partial t}, \tag{3.10}$$

where $\boldsymbol{A}$ is the vector potential and Φ is the scalar electrostatic potential associated with $\boldsymbol{E}$. However, because of what has been depicted in subsection 3.2.1, the form of the two fields that will be actually used is simpler. In particular, the magnetic field is stationary, uniform and directed along $\boldsymbol{\hat{b}_3}$ with superimposed amplitude B:

$$\boldsymbol{B}(\boldsymbol{q}) = (B_1, B_2, B_3) = (0, 0, B) = B\boldsymbol{\hat{b}_3}. \tag{3.11}$$

By analyzing Eqs. 3.9 and 3.11, it is clear that the vector potential has to be stationary:

$$\boldsymbol{A} = \boldsymbol{A}(\boldsymbol{q}). \tag{3.12}$$

Since $\boldsymbol{A}$ has no time dependence, the electric field is reduced just to:

$$\boldsymbol{E}(\boldsymbol{q},t) = -\nabla\Phi(\boldsymbol{q},t). \tag{3.13}$$

At this point it is possible to write the equation of motion for a single particle of electric charge e and mass m:

$$m\frac{d^2\boldsymbol{q}}{dt^2} = e\Bigg(\boldsymbol{E}(\boldsymbol{q},t) + \frac{d\boldsymbol{q}}{dt} \times \boldsymbol{B}(\boldsymbol{q})\Bigg). \tag{3.14}$$

Similarly to Eq. 2.7, the effect of gravitational force is neglected since the focus is always on magnetically confined plasma. For now, no further specifications on Φ are made, but anyway it will not involve any feedback from the charged particle. The Hamiltonian for this mechanical system is well known and is given by:

$$H(\boldsymbol{q},\boldsymbol{p},t) = \frac{1}{2m}\big(\boldsymbol{p} - e\boldsymbol{A}(\boldsymbol{q})\big)^2 + e\Phi(\boldsymbol{q},t). \tag{3.15}$$

Eq. 3.14 can be obtained from $\dot{\boldsymbol{q}} = \{q, H\}$ and $\dot{\boldsymbol{p}} = \{p, H\}$, in which Poisson brackets act on canonical variables and on the Hamiltonian. They have the canonical variables themselves as their variables (see Appendix A).
It is now that two subsequent and invertible changes of coordinates are performed. The first keeps the attention towards the single charged particle, but takes the derivation into the more familiar Cartesian setting. Instead, the second one shifts the focus to the guiding center. An additional transformation will be performed later, but it will only have a concrete effect on the Hamiltonian for the purposes of this work.

The exploitation of changes of coordinates is associated with the nature of charged particles inside uniform magnetic fields. Such particles move in a helicoidal way around magnetic field lines. However, rotation happens on much faster timescales than translation. As a consequence, in order to study phenomena such as turbulent dynamics, it is possible to neglect the rotational component of motion by performing an averaging of fast timescales, which means focusing just on guiding center dynamics. *Focusing on guiding center dynamics* means *needing a series of invertible changes of coordinates to move from the positions of particles to the positions of their guiding centers*, while *averaging of fast timescales* means *integrating over the gyroangle θ*, which is a variable that appears because of changes of coordinates.

3.3.2 First change of coordinates

The first transformation maps $\boldsymbol{z}$ to a new vector $\bar{\boldsymbol{z}}$ composed of three positions (exactly like in $\boldsymbol{z}$) and three velocities (instead of momenta):

$$\boldsymbol{z} \longleftrightarrow \bar{\boldsymbol{z}} = (\boldsymbol{x}, \boldsymbol{v}), \tag{3.16}$$

$$\begin{cases} \boldsymbol{q} = (q_1, q_2, q_3), \\ \\ \boldsymbol{p} = (p_1, p_2, p_3), \end{cases} \longleftrightarrow \begin{cases} \boldsymbol{x} = \boldsymbol{q} = (x_1, x_2, x_3), \\ \\ \boldsymbol{v} = \dfrac{1}{m}(\boldsymbol{p} - e\boldsymbol{A}(\boldsymbol{q}, t)) = (v_1, v_2, v_3). \end{cases} \tag{3.17}$$

Again, there is an implicit time dependence for both $\boldsymbol{x}$ and $\boldsymbol{v}$, but it is omitted. The choice of using numeric subscripts even in a Cartesian environment has been done to facilitate part of the following derivation of the wanted equations, but after a while $(\square_1, \square_2, \square_3)$ will be relabelled. Moreover, this transformation effectively characterizes $\boldsymbol{\mathcal{B}}$ by showing that it can be seen as an orthonormal Cartesian basis.
The Hamiltonian then becomes:

$$H(\boldsymbol{x}, \boldsymbol{v}, t) = \frac{1}{2}m\boldsymbol{v}^2 + e\Phi(\boldsymbol{x}, t). \tag{3.18}$$

3.3.3 Second change of coordinates

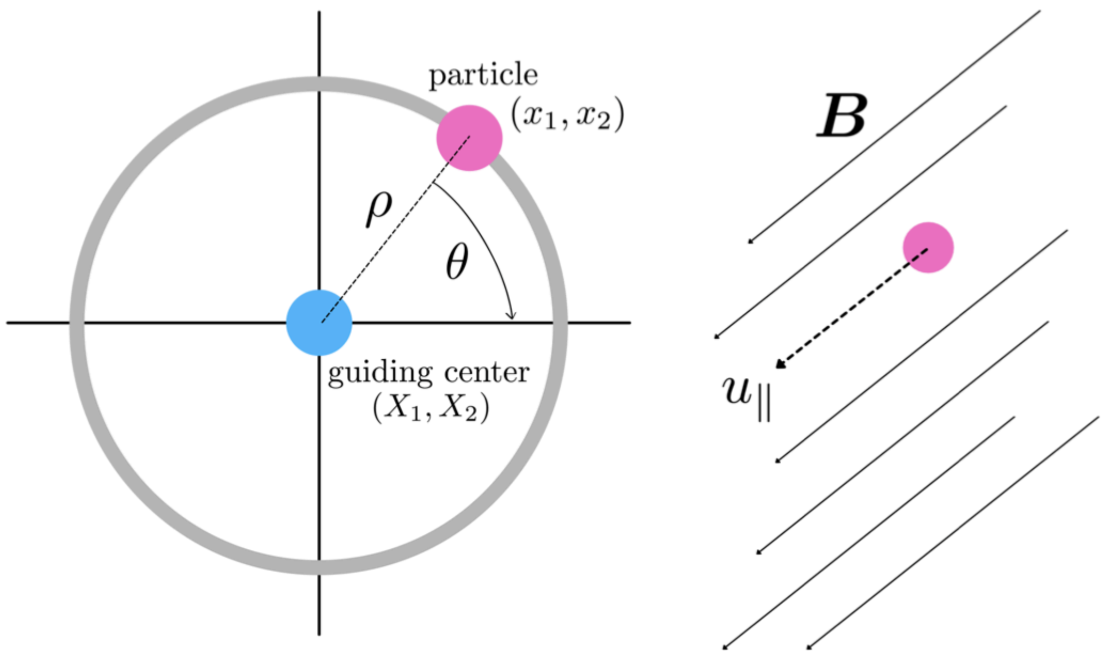


Figure 3.1: Visual representation of the second change of coordinates, from the particle framework to the guiding center one.

While the first change of coordinates simplifies the Hamiltonian and turn momenta into velocities, the next transformation seems to complicate them again. This is however a necessary step to get to the guiding center point of view and is vital to have acceptable computation times for the simulations. In specific, the three components of the particle position are shifted to the guiding center ones and then there are three additional quantities: the Larmor radius (which is a length like positions), the gyroangle and the component of the particle velocity that is parallel to the magnetic field lines. In can be noticed that this last variable and its correspondent for the guiding center are almost the same: the guiding center and the particle move along the direction of the magnetic field lines with the exact same velocity (the only difference is that they are in different positions in the transverse plane). In symbols, the new change of coordinates maps $\bar{\boldsymbol{z}}$ to $\bar{\bar{\boldsymbol{z}}}$:

$$\bar{\boldsymbol{z}} = (\boldsymbol{x}, \boldsymbol{v}) \longleftrightarrow \bar{\bar{\boldsymbol{z}}} = (\boldsymbol{X}, \rho, \theta, u_{\|}), \tag{3.19}$$

$$\begin{cases} \boldsymbol{x} = (x_1, x_2, x_3), \\ \\ \boldsymbol{v} = (v_1, v_2, v_3), \end{cases} \longleftrightarrow \begin{cases} \boldsymbol{X} = \boldsymbol{x} - \dfrac{m}{eB}\hat{\boldsymbol{b}}_{\boldsymbol{3}} \times \boldsymbol{v} = (X_1, X_2, X_3), \\ \\ \rho = \dfrac{m}{|e|B}\sqrt{v_1^2 + v_2^2}, \\ \\ \theta = -\tan^{-1}\left(\dfrac{v_2}{v_1}\right), \\ \\ u_\parallel = v_3. \end{cases} \tag{3.20}$$

It is straightforward to verify by substitution that the Hamiltonian written with these new variables is:

$$H(\boldsymbol{X}, \rho, \theta, u_\parallel, t) = \frac{e^2B^2}{2m}\rho^2 + \frac{1}{2}mu_\parallel^2 + e\Phi(\boldsymbol{X}, \rho, \theta, t). \tag{3.21}$$

Simple and useful relations that connect the position of the particle and the one of its guiding center are obtainable by exploiting the Larmor vector $\boldsymbol{\rho}$:

$$\boldsymbol{\rho} = (\rho\cos\theta, -\rho\sin\theta, 0), \tag{3.22}$$

$$\begin{cases} x_1 = X_1 + \rho\cos\theta, \\ x_2 = X_2 - \rho\sin\theta, \\ x_3 = X_3. \end{cases} \tag{3.23}$$

It is important to notice that the different signs appearing in the components of the Larmor vector have to do with the particular definition of the gyroangle, which ideally starts from the actual Larmor radius pointing outwards, and not viceversa. It is actually straightforward to verify these signs by trying to decompose the particle distance to the guiding center using the angle α displayed in Fig. 3.2, which is the opposite of the gyroangle:

$$x_1 - X1 = \rho\cos\alpha = \rho\cos(-\theta) = \rho\cos\theta, \tag{3.24}$$

$$x_2 - X2 = \rho\sin\alpha = \rho\sin(-\theta) = -\rho\sin\theta. \tag{3.25}$$

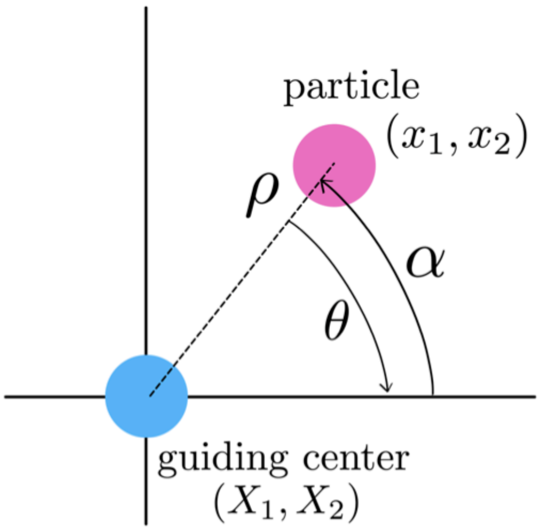


Figure 3.2: Schematic representation to explain the expression of the Larmor vector based on the particular choice for the gyroangle.

3.3.4 Determining the Poisson bracket for the guiding center configuration

After having completed the changes of coordinates, the following important step is to determine the form of the Poission brackets for the system described through $\bar{\bar{\boldsymbol{z}}}_\tau$. This has to do with the fact that the temporal evolution of any observable, such as the coordinates themselves, can be determined once the Hamiltonian and the Poisson bracket are known. The idea is that this can be exploited to formulate the equations that describe the guiding center dynamics. All the useful information regarding Poisson brackets and how to calculate them are displayed in sections A.2-A.4 of Appendix A.
Before going into the mathematical details, it is useful to explicitly show and stress the other label that has been given to the various coordinates:

$$\boldsymbol{z} = (\boldsymbol{q}, \boldsymbol{p}) = (z_1, ..., z_6), \tag{3.26}$$

$$\bar{\boldsymbol{z}} = (\boldsymbol{x}, \boldsymbol{v}) = (\bar{z}_1, ..., \bar{z}_6), \tag{3.27}$$

$$\bar{\bar{\boldsymbol{z}}} = (\boldsymbol{X}, \rho, \theta, u_\parallel) = (\bar{\bar{z}}_1, ..., \bar{\bar{z}}_6). \tag{3.28}$$

This is helpful because there will be a lot of indices in the formulae involved in the following steps, which by the way are a direct application of what can be found in the above-mentioned Appendix. It is important to underline even here that the procedure revolves around evaluating the matrices that account for the changes of coordinates. The last one of those matrices is then used to express the Poisson bracket in the guiding

center setting.
Proceding step by step, the starting point is to determine the matrix $\mathcal{J}(\boldsymbol{z})$ associated with the original set of canonical coordinates. Its explicit form is:

$$\mathcal{J}(\boldsymbol{z}) = \begin{pmatrix} \{q_1,q_1\} & \{q_1,q_2\} & \{q_1,q_3\} & \{q_1,p_1\} & \{q_1,p_2\} & \{q_1,p_3\} \\ \{q_2,q_1\} & \{q_2,q_2\} & \{q_2,q_3\} & \{q_2,p_1\} & \{q_2,p_2\} & \{q_2,p_3\} \\ \{q_3,q_1\} & \{q_3,q_2\} & \{q_3,q_3\} & \{q_3,p_1\} & \{q_3,p_2\} & \{q_3,p_3\} \\ \{p_1,q_1\} & \{p_1,q_2\} & \{p_1,q_3\} & \{p_1,p_1\} & \{p_1,p_2\} & \{p_1,p_3\} \\ \{p_2,q_1\} & \{p_2,q_2\} & \{p_2,q_3\} & \{p_2,p_1\} & \{p_2,p_2\} & \{p_2,p_3\} \\ \{p_3,q_1\} & \{p_3,q_2\} & \{p_3,q_3\} & \{p_3,p_1\} & \{p_3,p_2\} & \{p_3,p_3\} \end{pmatrix}. \tag{3.29}$$

Each element of this matrix represents a Poisson bracket that acts on two out of the six coordinates of $\boldsymbol{z}$ and that has all these six coordinates as its variables. The generic element positioned in the k-th row and l-th column can therefore be evaluated with this formula:

$$k,l = 1,...,6 : \mathcal{J}_{kl}(\boldsymbol{z}) = \{z_k, z_l\} = \frac{\partial z_k}{\partial \boldsymbol{q}} \cdot \frac{\partial z_l}{\partial \boldsymbol{p}} - \frac{\partial z_k}{\partial \boldsymbol{p}} \cdot \frac{\partial z_l}{\partial \boldsymbol{q}}. \tag{3.30}$$

Eq. 3.30 is then used to build the second transformation matrix, i.e. the one associated with the second set of coordinates:

$$\mathcal{J}(\bar{\boldsymbol{z}}) = \begin{pmatrix} \{x_1,x_1\} & \{x_1,x_2\} & \{x_1,x_3\} & \{x_1,v_1\} & \{x_1,v_2\} & \{x_1,v_3\} \\ \{x_2,x_1\} & \{x_2,x_2\} & \{x_2,x_3\} & \{x_2,v_1\} & \{x_2,v_2\} & \{x_2,v_3\} \\ \{x_3,x_1\} & \{x_3,x_2\} & \{x_3,x_3\} & \{x_3,v_1\} & \{x_3,v_2\} & \{x_3,v_3\} \\ \{v_1,x_1\} & \{v_1,x_2\} & \{v_1,x_3\} & \{v_1,v_1\} & \{v_1,v_2\} & \{v_1,v_3\} \\ \{v_2,x_1\} & \{v_2,x_2\} & \{v_2,x_3\} & \{v_2,v_1\} & \{v_2,v_2\} & \{v_2,v_3\} \\ \{v_3,x_1\} & \{v_3,x_2\} & \{v_3,x_3\} & \{v_3,v_1\} & \{v_3,v_2\} & \{v_3,v_3\} \end{pmatrix}. \tag{3.31}$$

This time, each Poisson bracket acts on two out the six coordinates of $\bar{\boldsymbol{z}}$, but the variables are still the ones contained in $\boldsymbol{z}$. This is caused by the fact that the new set depends on the old one through the relations 3.17. The expanded version of the generic element actually shows the presence of $\mathcal{J}(\boldsymbol{z})$ and is given by:

$$k,l = 1,...,6 : \mathcal{J}_{kl}(\bar{\boldsymbol{z}}) = \{\bar{z}_k, \bar{z}_l\} = \sum_{m,n=1}^{6} \frac{\partial \bar{z}_k}{\partial z_m} \mathcal{J}_{mn}(\boldsymbol{z}) \frac{\partial \bar{z}_l}{\partial z_n} =$$

$$= \sum_{m,n=1}^{6} \frac{\partial \bar{z}_k}{\partial z_m} \{z_m, z_n\} \frac{\partial \bar{z}_l}{\partial z_n}. \tag{3.32}$$

The third step is exactly the same as the second: Eq. 3.32 serves as the main tool to create the transformation matrix for the guiding center set of coordinates:

$$\mathcal{J}(\bar{\bar{\boldsymbol{z}}}) = \begin{pmatrix} \{X_1, X_1\} & \{X_1, X_2\} & \{X_1, X_3\} & \{X_1, \rho\} & \{X_1, \theta\} & \{X_1, u_\parallel\} \\ \{X_2, X_1\} & \{X_2, X_2\} & \{X_2, X_3\} & \{X_2, \rho\} & \{X_2, \theta\} & \{X_2, u_\parallel\} \\ \{X_3, X_1\} & \{X_3, X_2\} & \{X_3, X_3\} & \{X_3, \rho\} & \{X_3, \theta\} & \{X_3, u_\parallel\} \\ \{\rho, X_1\} & \{\rho, X_2\} & \{\rho, X_3\} & \{\rho, \rho\} & \{\rho, \theta\} & \{\rho, u_\parallel\} \\ \{\theta, X_1\} & \{\theta, X_2\} & \{\theta, X_3\} & \{\theta, \rho\} & \{\theta, \theta\} & \{\theta, u_\parallel\} \\ \{u_\parallel, X_1\} & \{u_\parallel, X_2\} & \{u_\parallel, X_3\} & \{u_\parallel, \rho\} & \{u_\parallel, \theta\} & \{u_\parallel, u_\parallel\} \end{pmatrix}. \tag{3.33}$$

The recurring pattern now should be more evident: each Poisson bracket acts on two out of the six elements of $\bar{\bar{\boldsymbol{z}}}$, but the variables are still taken from the previous set $\bar{\boldsymbol{z}}$:

$$k, l = 1, .., 6 : \mathcal{J}_{kl}(\bar{\bar{\boldsymbol{z}}}) = \{\bar{\bar{z}}_k, \bar{\bar{z}}_l\} = \sum_{m,n=1}^{6} \frac{\partial \bar{z}_k}{\partial z_m} \mathcal{J}_{mn}(\bar{\boldsymbol{z}}) \frac{\partial \bar{\bar{z}}_l}{\partial \bar{z}_n} =$$

$$= \sum_{m,n=1}^{6} \frac{\partial \bar{\bar{z}}_k}{\partial \bar{z}_m} \{\bar{z}_m, \bar{z}_n\} \frac{\partial \bar{\bar{z}}_l}{\partial \bar{z}_n}. \tag{3.34}$$

At this point it is possible to write down the form of the desired Poisson bracket:

$$\{F, G\} = \sum_{m,n=1}^{6} \frac{\partial F}{\bar{\bar{z}}_m} \mathcal{J}_{mn}(\bar{\bar{\boldsymbol{z}}}) \frac{\partial G}{\partial \bar{\bar{z}}_n} = \frac{\partial F}{\partial \bar{\bar{\boldsymbol{z}}}} \cdot \mathcal{J}(\bar{\bar{\boldsymbol{z}}}) \left(\frac{\partial G}{\partial \bar{\bar{\boldsymbol{z}}}} \right)^{\mathrm{T}}, \tag{3.35}$$

where $F = F(\bar{\bar{\boldsymbol{z}}})$ and $G = G(\bar{\bar{\boldsymbol{z}}})$ are any two observables, i.e. scalar functions of the chosen set of coordinates. It can be noticed that, in order to properly perform the scalar product, the first vector on the right-hand side is a row vector, while the second is a column vector obtained by transposition.
Arrived at this point, the formula for the Poisson bracket is there but actually has not been evaluated yet. The procedure to obtain the true form could be done manually with pen and paper. However, it is tricky because it tends to be repetitive and it is easy to make confusion given the considerable amount of indices and similar symbols. Therefore, a clever idea is to write small MATLAB scripts that exploit the Symbolic Math Toolbox. Other than saving time, this has also other advantages:

- The scripts can be easily recycled and adapted to calculate other Poisson brackets.
- The method ensures that the calculations are correct (provided that the scripts have no errors).

The explicit forms of the three matrices of transformation are:

$$\mathcal{J}(\boldsymbol{z}) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}, \tag{3.36}$$

$$\mathcal{J}(\bar{\boldsymbol{z}}) = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{m} \\ -\frac{1}{m} & 0 & 0 & 0 & \frac{Be}{m^2} & 0 \\ 0 & -\frac{1}{m} & 0 & -\frac{Be}{m^2} & 0 & 0 \\ 0 & 0 & -\frac{1}{m} & 0 & 0 & 0 \end{pmatrix}, \tag{3.37}$$

$$\mathcal{J}(\bar{\bar{\boldsymbol{z}}}) = \begin{pmatrix} 0 & -\dfrac{1}{Be} & 0 & 0 & 0 & 0 \\ \dfrac{1}{Be} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dfrac{1}{m} \\ 0 & 0 & 0 & 0 & -\dfrac{1}{e\rho B} & 0 \\ 0 & 0 & 0 & \dfrac{1}{e\rho B} & 0 & 0 \\ 0 & 0 & -\dfrac{1}{m} & 0 & 0 & 0 \end{pmatrix}. \tag{3.38}$$

Thanks to the explicit form of $\mathcal{J}(\bar{\bar{\boldsymbol{z}}})$, the actual expression for the Poisson bracket of Eq. A.21 is:

$$\{F, G\} = \frac{1}{eB\rho}\left(\frac{\partial F}{\partial \theta}\frac{\partial G}{\partial \rho} - \frac{\partial F}{\partial \rho}\frac{\partial G}{\partial \theta}\right) - \frac{1}{eB}\left(\frac{\partial F}{\partial X_1}\frac{\partial G}{\partial X_2} - \frac{\partial F}{\partial X_2}\frac{\partial G}{\partial X_1}\right) + \\ + \frac{1}{m}\left(\frac{\partial F}{\partial X_3}\frac{\partial G}{\partial u_\parallel} - \frac{\partial F}{\partial u_\parallel}\frac{\partial G}{\partial X_3}\right). \tag{3.39}$$

```
syms P O real
syms v_x v_y v_z real
syms e B m real
syms r_x r_y r_z real
b = [0 0 1];
r = [r_x r_y r_z];
v = [v_x v_y v_z];
P = (m/(abs(e)*B))*sqrt(v(1)^2+v(2)^2);
O = -atan(v(2)/v(1));
z = [r, v];
j1 = [ 0,    0,    0,              1/m,              0,              0;...
       0,    0,    0,                0,            1/m,              0;...
       0,    0,    0,                0,              0,            1/m;...
    -1/m,    0,    0,                0,   (B*b(3)*e)/m^2, -(B*b(2)*e)/m^2;...
       0, -1/m,    0, -(B*b(3)*e)/m^2,              0,   (B*b(1)*e)/m^2;...
       0,    0, -1/m,  (B*b(2)*e)/m^2, -(B*b(1)*e)/m^2,              0];
J1 = j1(1:6,1:6);
dP = simplify(gradient(P, z));
dO = simplify(gradient(O, z));
Di = dP;
Dj = dO;
for i = 1:6
    for j = 1:6
        a(i, j) = Di(i)*J1(i, j)*Dj(j);
    end
end
P_O = simplify(sum(a, 'all'));
O_P = -P_O;
```

Listing 1: Selected portion of one of the MATLAB scripts used to obtain transformation matrices and the Poisson bracket displayed in Eq. 3.39. First, the MATLAB Symbolic Toolbox is exploited to define ρ, θ and $\bar{z}$ (i.e. `P`, `O` and `z`, respectively). Then, the transformation matrix $\mathrm{J}(\bar{z})$ (previously obtained with another MATLAB script) is used together with the derivatives of ρ and θ to compute $\{\rho, \theta\}$ and $\{\theta, \rho\}$ (i.e. `P_O` and `O_P`), which are part of $\mathrm{J}(\bar{\bar{z}})$.

3.3.5 Autonomization of the system and extension of the Poisson bracket

It can be noticed that the Hamiltonian in Eq. 3.21 has an explicit time dependence through $\Phi(\boldsymbol{X}, \rho, \theta, t)$. This has to do with the presence of external forces acting on the charged particle. In these cases, it is convenient to autonomize the entire system by introducing a new variable, h, which is canonically conjugated with time t. Therefore, H is transformed into a new version labelled as $\mathcal{H}$:

$$\mathcal{H}(\boldsymbol{X}, \rho, \theta, u_\parallel, t, h) = \frac{e^2B^2}{2m}\rho^2 + \frac{1}{2}mu_\parallel^2 + e\Phi(\boldsymbol{X}, \rho, \theta, t) + h. \tag{3.40}$$

Moreover, an extended set of coordinates, denoted by the subscript τ, can be outlined:

$$\bar{\bar{\boldsymbol{z}}}_\tau = (\bar{\bar{\boldsymbol{z}}}, t, h). \tag{3.41}$$

Because of the autonomization, it is important at this point to also extend the Poisson bracket that has been obtained in the previous section. Information about this procedure, which is actually straightforward, can be found in section A.3 of Appendix A together with a closer look on autonomization. The end result, which is one of those important formulae that needed a validation, is:

$$\begin{aligned}\{\mathcal{F}, \mathcal{G}\}_\tau = {} & \frac{1}{eB\rho}\left(\frac{\partial \mathcal{F}}{\partial \theta}\frac{\partial \mathcal{G}}{\partial \rho} - \frac{\partial \mathcal{F}}{\partial \rho}\frac{\partial \mathcal{G}}{\partial \theta}\right) - \frac{1}{eB}\left(\frac{\partial \mathcal{F}}{\partial X_1}\frac{\partial \mathcal{G}}{\partial X_2} - \frac{\partial \mathcal{F}}{\partial X_2}\frac{\partial \mathcal{G}}{\partial X_1}\right) + \\ & + \frac{1}{m}\left(\frac{\partial \mathcal{F}}{\partial X_3}\frac{\partial \mathcal{G}}{\partial u_\parallel} - \frac{\partial \mathcal{F}}{\partial u_\parallel}\frac{\partial \mathcal{G}}{\partial X_3}\right) + \left(\frac{\partial \mathcal{F}}{\partial t}\frac{\partial \mathcal{G}}{\partial h} - \frac{\partial \mathcal{F}}{\partial h}\frac{\partial \mathcal{G}}{\partial t}\right),\end{aligned} \tag{3.42}$$

where τ denotes the above-mentioned extension, while $\mathcal{F}$ and $\mathcal{G}$ are two generic observables of the extended set of coordinates $\bar{\bar{\boldsymbol{z}}}_\tau$. The same result could have been obtained by slightly modifying the MATLAB script used for the various matrices. However, since the extension always modifies the Poisson bracket in the same way, regardless of the set of coordinates, it is far more convenient to manually perform it afterwards.

3.4 Elimination of the gyroangle from the autonomized Hamiltonian using the Lie Transform

In the previous section, the Hamiltonian has been changed by autonomizing the system. However, an additional modification is required in the framework of guiding center theory and that has to do with the gyroangle θ. In fact, it is true that a single guiding center is associated with a single particle, but it also true that the particle rotates, while the center does not. It is this rotation that can arise problems when simulations are performed:

not only it increases computation time by a huge amount, but it is also an unnecessary phenomenon to process and analyze, given that many relevant plasma phenomena are much slower than the rotation period. What guiding center theory wants to achieve could be alternatively explained by saying that it is trying to replicate the screen of a computer as it is seen by human eyes when it is switched on. The eye only cares about what is being displayed, but it is not capable of detecting the constant and super fast refreshing of the screen itself (actually, it is also not interested in that aspect at all). In this metaphor, the relevant plasma phenomena are what is displayed on screen, while the screen refreshing is the rotational component of motion.
In this context, the idea is then to eliminate the gyroangle from the Hamiltonian so that the guiding center dynamics in the transverse plane can be independent of it when it is evaluated with Poisson brackets that act on $\mathcal{H}$ together with either X_1 or X_2. In order to achieve this, the Lie Transform has to be taken into account. It can act on any observable in such a way that the form of Poisson brackets remains unchanged. This very relevant fact helps to avoid an otherwise possible inconvenience, which is making the Poisson bracket explicitly dependent of the gyroangle. More details on Lie transforms can be found in section A.5 of Appendix A.

3.4.1 Introduction of small dimensionless parameters to further characterize the problem

This part of the derivation opens up with the introduction of a parameter, ε, that measures how fast the particle rotates around its guiding center. In particular, it is a dimensionless real number of the order of $1/(\Omega\zeta)$, where Ω is the Larmor frequency defined in Eq. 2.10 and ζ is the typical timescale of interest. The Poisson bracket is affected by ε because the first term on the right-hand side of Eq. 3.42 is associated with the fast Larmor dynamics of rotation around the guiding center:

$$\frac{1}{eB\rho}\left(\frac{\partial\mathcal{F}}{\partial\theta}\frac{\partial\mathcal{G}}{\partial\rho}-\frac{\partial\mathcal{F}}{\partial\rho}\frac{\partial\mathcal{G}}{\partial\theta}\right)\longrightarrow\frac{1}{\varepsilon eB\rho}\left(\frac{\partial\mathcal{F}}{\partial\theta}\frac{\partial\mathcal{G}}{\partial\rho}-\frac{\partial\mathcal{F}}{\partial\rho}\frac{\partial\mathcal{G}}{\partial\theta}\right), \tag{3.43}$$

$$\begin{aligned}\{\mathcal{F},\mathcal{G}\}_\tau = {}&\frac{1}{\varepsilon eB\rho}\left(\frac{\partial\mathcal{F}}{\partial\theta}\frac{\partial\mathcal{G}}{\partial\rho}-\frac{\partial\mathcal{F}}{\partial\rho}\frac{\partial\mathcal{G}}{\partial\theta}\right)-\frac{1}{eB}\left(\frac{\partial\mathcal{F}}{\partial X_1}\frac{\partial\mathcal{G}}{\partial X_2}-\frac{\partial\mathcal{F}}{\partial X_2}\frac{\partial\mathcal{G}}{\partial X_1}\right)+\\ &+\frac{1}{m}\left(\frac{\partial\mathcal{F}}{\partial X_3}\frac{\partial\mathcal{G}}{\partial u_\parallel}-\frac{\partial\mathcal{F}}{\partial u_\parallel}\frac{\partial\mathcal{G}}{\partial X_3}\right)+\left(\frac{\partial\mathcal{F}}{\partial t}\frac{\partial\mathcal{G}}{\partial h}-\frac{\partial\mathcal{F}}{\partial h}\frac{\partial\mathcal{G}}{\partial t}\right).\end{aligned} \tag{3.44}$$

In the following, the subscript τ will not be used for simplicity. Other than ε, another dimensionless parameter that accounts for the scales of variation of the electrostatic potential Φ is from now on exploited: ε_δ. The double assumption that is made at this point consists in having Φ small and slowly varying with both time and the direction of the magnetic field lines. Effectively, the old potential is turned into $\varepsilon_\delta\Phi$ with the first

hypothesis, and then is characterized in its variables as $\varepsilon_\delta \Phi(X_1, X_2, \varepsilon_\delta X_3, \rho, \theta, \varepsilon_\delta t)$ with the second one.
It can be noticed that the latter assumption is somewhat related to what has been said in section 3.2.1. Actually, the hypothesis on the electric field in that section is slightly stricter than what is being considered here. Again, this has to do with the will to keep the derivation more general than what is really needed for the coding section of this work. This allows us to possibly recycle what is presented here for other scenarios.
With this second introduction, the autonomized Hamiltonian is evidently changed into:

$$\mathcal{H}(\boldsymbol{X}, \rho, \theta, u_\parallel, t, h) = \frac{e^2 B^2}{2m}\rho^2 + \frac{1}{2}mu_\parallel^2 + e\varepsilon_\delta \Phi(X_1, X_2, \varepsilon_\delta X_3, \rho, \theta, \varepsilon_\delta t) + h. \tag{3.45}$$

It is now convenient, for later clarity, to label the various terms of $\mathcal{H}$ in this way:

$$\begin{cases} \mathcal{H}_0 = \mathcal{H}_0(\rho, u_\parallel, h) = \dfrac{e^2 B^2}{2m}\rho^2 + \dfrac{1}{2}mu_\parallel^2 + h, \\ \\ \varepsilon_\delta \mathcal{H}_1 = \varepsilon_\delta \mathcal{H}_1(\boldsymbol{X}, \rho, \theta, t) = e\varepsilon_\delta \Phi(\boldsymbol{X}, \rho, \theta, t). \end{cases} \tag{3.46}$$

This allows one to write a very compact Hamiltonian:

$$\mathcal{H} = \mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1, \tag{3.47}$$

in which a further inspection reveals that $\mathcal{H}_0$ could be seen as $\varepsilon_\delta^0 \mathcal{H}_0$ and therefore suggests that the names given to the Hamiltonian terms are based on the instances of ε_δ and its exponents in Eq. 3.45.
This last observation is very helpful, since it shows what could be said to be a more mathematical approach to this portion of the derivation. In fact, the new Hamiltonian can be seen as a series expansion with all the terms beyond the second equal to zero:

$$\mathcal{H} = \sum_{n=0}^{\infty} \varepsilon_\delta^n \mathcal{H}_n. \tag{3.48}$$

Given that this way of proceeding is more general (even if it contains less physics) and does not overcomplicate anything, the following steps will be made in an even broader setting than what has been done up to here. In essence, this simply means that while

$$\mathcal{H} = \sum_{n=0}^{1} \varepsilon_\delta^n \mathcal{H}_n \tag{3.49}$$

or even

$$\mathcal{H} = \sum_{n=0}^{2} \varepsilon_\delta^n \mathcal{H}_n \quad (\text{with } \mathcal{H}_2 = 0) \tag{3.50}$$

would be both sufficient for this work, the Hamiltonian will be treated as:

$$\mathcal{H} = \sum_{n=0}^{2} \varepsilon_\delta^n \mathcal{H}_n + O(\varepsilon_\delta^3). \tag{3.51}$$

3.4.2 Application of the Lie transform in a more general setting

In this subsection, the Lie transform will be actively used. The fundamental elements are the Hamiltonian function of Eq. 3.51, the small parameter ε and a generating function S, which is expressed through a series like $\mathcal{H}$:

$$S(\bar{\bar{\boldsymbol{z}}}_\tau) = \sum_{n=0}^{+\infty} \varepsilon_\delta^n S_n(\bar{\bar{\boldsymbol{z}}}_\tau) = \sum_{n=0}^{2} \varepsilon_\delta^n S_n(\bar{\bar{\boldsymbol{z}}}_\tau) + O(\varepsilon_\delta^3), \tag{3.52}$$

which can also be written as:

$$S = S_0 + \varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2 + O(\varepsilon_\delta^3). \tag{3.53}$$

At this point, a Lie transform equipped with ε and S is immediately applied to $\mathcal{H}$ to get $\bar{\mathcal{H}}$. However, the effect of this operation, i.e. eliminating θ from the Hamiltonian, will take several steps to become evident:

$$\bar{\mathcal{H}} = \mathrm{e}^{-\varepsilon \mathcal{L}_S} \mathcal{H} = \sum_{n=0}^{2} \frac{(-1)^n \varepsilon^n}{n!} \mathcal{L}_S^n \mathcal{H} + O(S^3). \tag{3.54}$$

Expliciting the expression and recalling the definition of the Liouville operator $\mathcal{L}_S$ lead to:

$$\bar{\mathcal{H}} = \frac{(-1)^0 \varepsilon^0}{0!} \mathcal{L}_S^0 \mathcal{H} + \frac{(-1)^1 \varepsilon^1}{1!} \mathcal{L}_S^1 \mathcal{H} + \frac{(-1)^2 \varepsilon^2}{2!} \mathcal{L}_S^2 \mathcal{H} + O(S^3) =$$
$$= \mathcal{H} - \varepsilon\{S, \mathcal{H}\} + \frac{1}{2}\varepsilon^2 \{S, \{S, \mathcal{H}\}\} + O(S^3), \tag{3.55}$$

where the Poisson brackets are all in the form of Eq. 3.44. This expression can be developed with the help of Eqs. 3.51 and 3.53 and then by assigning new symbols to the various relevant terms of $\bar{\mathcal{H}}$:

$$\bar{\mathcal{H}}_0 = \mathcal{H} = \mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1 + \varepsilon_\delta^2 \mathcal{H}_2 + O(\varepsilon_\delta^3), \tag{3.56}$$

$$\bar{\mathcal{H}}_1 = -\varepsilon\{S, \mathcal{H}\} = -\varepsilon\{(S_0 + \varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1 + \varepsilon_\delta^2 \mathcal{H}_2)\} + O(\varepsilon_\delta^3), \tag{3.57}$$

$$\bar{\mathcal{H}}_2 = \frac{1}{2}\varepsilon^2 \{S, \{S, \mathcal{H}\}\} =$$
$$= \frac{1}{2}\varepsilon^2 \{(S_0 + \varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), \{(S_0 + \varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1 + \varepsilon_\delta^2 \mathcal{H}_2)\}\} + O(\varepsilon_\delta^3). \tag{3.58}$$

The compact version of $\bar{\mathcal{H}}$ is therefore given by:

$$\bar{\mathcal{H}} = \bar{\mathcal{H}}_0 + \bar{\mathcal{H}}_1 + \bar{\mathcal{H}}_2 + O(S^3). \tag{3.59}$$

It is important to point out here that the $O(...)$ notation does not imply that all the terms that should go in it are already there. Some parts could still be hidden inside other

non-explicit or non-developed terms. This is true for what has already been written, but also for what comes after.
Now that Eq. 3.59 is in place, the next step to do is the elimination of its low-order fluctuating parts by pushing them to higher orders.
Their removal can be done by choosing the value of the generating function S or, more precisely, by properly setting its terms: S_0, S_1 and so on. In order to better characterize what has just been said, it has to be pointed out that each observable can be seen as the superposition of a fluctuating and a mean part. In this context, *fluctuating* means θ-*depending* and it is indicated with $\tilde{\Box}$, while *mean* stands for θ-*depending* and its symbol is $\langle\Box\rangle_\theta$ or, equivalently, $\langle\Box\rangle$. Considering a generic observable $\mathcal{F}$, it can be rewritten as:

$$\mathcal{F} = \langle\mathcal{F}\rangle + \tilde{\mathcal{F}}. \tag{3.60}$$

The averaging has a specific mathematical expression:

$$\langle\mathcal{F}\rangle = \frac{1}{2\pi}\int_0^{2\pi} \mathcal{F} d\theta. \tag{3.61}$$

In the case that the observable is expressed through a Poisson bracket like $\{\mathcal{F},\mathcal{G}\}$, then the adopted notation is slightly different:

$$\{\mathcal{F},\mathcal{G}\} = \{\mathcal{F},\mathcal{G}\}^{(<>)} + \{\mathcal{F},\mathcal{G}\}^{(\sim)}, \tag{3.62}$$

where, as usual, $\mathcal{F}$ and $\mathcal{G}$ are any two observables themselves. It is easier to see now that the attention has to be put towards $\mathcal{H}_1$, because all the fluctuating terms can only be there.

Determination of S_0

If Eq. 3.57 is partially developed, the first two Poisson brackets that appear feature S_0, with the second of them being θ-dependent.

$$\bar{\mathcal{H}}_1 = -\varepsilon\{S_0,\mathcal{H}_0\} - \varepsilon\{S_0,\varepsilon_\delta\mathcal{H}_1\} - \varepsilon\{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2),(\mathcal{H}_0 + \varepsilon_\delta\mathcal{H}_1)\} + O(\varepsilon_\delta^3). \tag{3.63}$$

By looking at that bracket, the conclusion is that eliminating its fluctuating component requires to choose $S_0 = 0$:

$$-\varepsilon\{S_0,\varepsilon_\delta\mathcal{H}_1\} = -\varepsilon\varepsilon_\delta\{S_0,\langle\mathcal{H}_1\rangle\} \underbrace{-\varepsilon\varepsilon_\delta\{S_0,\tilde{\mathcal{H}}_1\}}_{\text{actual fluctuating term}} \longrightarrow$$

$$\longrightarrow \{S_0,\tilde{\mathcal{H}}_1\} = 0 \longrightarrow S_0 = 0. \tag{3.64}$$

In this way, both the two above-mentioned brackets are cancelled. Moreover, together with the fact that $\mathcal{H}_2$ is null, also other terms are removed from $\bar{\mathcal{H}}$:

$$\begin{cases} \bar{\mathcal{H}}_0 = \mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1, \\ \\ \bar{\mathcal{H}}_1 = -\varepsilon\{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\} + O(\varepsilon_\delta^3), \\ \\ \bar{\mathcal{H}}_2 = \dfrac{1}{2}\varepsilon^2\{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), \{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\}\} + O(\varepsilon_\delta^3). \end{cases} \tag{3.65}$$

It can be noticed that non-fluctuating parts are removed with procedure, too. However, defining S_1 and S_2 is not as easy. It is trickier and requires some further calculations.

Determination of S_1

Both $\bar{\mathcal{H}}_1$ and $\bar{\mathcal{H}}_2$ can be expanded in such a way that more higher-order terms become explicit. They are then gathered into $O(...)$ notations. Moreover, this allows one to individuate all the terms containing S_1 and therefore creates the condition to properly set its value as discussed before. $\bar{\mathcal{H}}_1$ is now given by:

$$\bar{\mathcal{H}}_1 = -\varepsilon\{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\} + O(\varepsilon_\delta^3) =$$

$$= -\varepsilon\{\varepsilon_\delta S_1, \mathcal{H}_0\} - \varepsilon\{\varepsilon_\delta S_1, \varepsilon_\delta \mathcal{H}_1\} - \varepsilon\{\varepsilon_\delta^2 S_2, \mathcal{H}_0\} - \varepsilon\{\varepsilon_\delta^2 S_2, \varepsilon_\delta \mathcal{H}_1\} + O(\varepsilon_\delta^3) =$$
$$= -\varepsilon\varepsilon_\delta\{S_1, \mathcal{H}_0\} - \varepsilon\varepsilon_\delta^2\{S_1, \mathcal{H}_1\} - \varepsilon\varepsilon_\delta^2\{S_2, \mathcal{H}_0\} - \varepsilon\varepsilon_\delta^3\{S_2, \mathcal{H}_1\} + O(\varepsilon_\delta^3). \tag{3.66}$$

$\bar{\mathcal{H}}_2$ requires more rearrangments:

$$\bar{\mathcal{H}}_2 = \frac{1}{2}\varepsilon^2\{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), \{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\}\} + O(\varepsilon_\delta^3) =$$

$$= \frac{1}{2}\varepsilon^2\{\varepsilon_\delta S_1, \{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\}\} +$$

$$+ \frac{1}{2}\varepsilon^2\{\varepsilon_\delta^2 S_2, \{(\varepsilon_\delta S_1 + \varepsilon_\delta^2 S_2), (\mathcal{H}_0 + \varepsilon_\delta \mathcal{H}_1)\}\} + O(\varepsilon_\delta^3) =$$

$$= \frac{1}{2}\varepsilon^2\varepsilon_\delta\Big[\{S_1, \{\varepsilon_\delta S_1, \mathcal{H}_0\}\} + \{S_1, \{\varepsilon_\delta S_1, \varepsilon_\delta \mathcal{H}_1\}\} +$$

$$+ \{S_1, \{\varepsilon_\delta^2 S_2, \mathcal{H}_0\}\} + \{S_1, \{\varepsilon_\delta^2 S_2, \varepsilon_\delta \mathcal{H}_1\}\}\Big] +$$

$$+ \frac{1}{2}\varepsilon^2\varepsilon_\delta^2\Big[\{S_2, \{\varepsilon_\delta S_1, \mathcal{H}_0\}\} + \{S_2, \{\varepsilon_\delta S_1, \varepsilon_\delta \mathcal{H}_1\}\} +$$

$$+\{S_2,\{\varepsilon_\delta^2 S_2,\mathcal{H}_0\}\}+\{S_2,\{\varepsilon_\delta^2 S_2,\varepsilon_\delta\mathcal{H}_1\}\}\Big]+O(\varepsilon_\delta^3)=$$

$$=\frac{1}{2}\varepsilon^2\varepsilon_\delta^2\{S_1,\{S_1,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_1,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_2,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_1,\{S_2,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_2,\{S_1,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_1,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_2,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^5\{S_2,\{S_2,\mathcal{H}_1\}\}+O(\varepsilon_\delta^3)=$$
$$=\frac{1}{2}\varepsilon^2\varepsilon_\delta^2\{S_1,\{S_1,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_1,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_2,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_1,\{S_2,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^3\{S_2,\{S_1,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_1,\mathcal{H}_1\}\}+$$
$$+\frac{1}{2}\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_2,\mathcal{H}_0\}\}+\frac{1}{2}\varepsilon^2\varepsilon_\delta^5\{S_2,\{S_2,\mathcal{H}_1\}\}+O(\varepsilon_\delta^3). \tag{3.67}$$

To sum up, the three main terms of $\bar{\mathcal{H}}$ become:

$$\begin{cases}\bar{\mathcal{H}}_0=\mathcal{H}_0+\varepsilon_\delta\mathcal{H}_1,\\[2ex] \bar{\mathcal{H}}_1=-\varepsilon\varepsilon_\delta\{S_1,\mathcal{H}_0\}-\varepsilon\varepsilon_\delta^2\{S_1,\mathcal{H}_1\}-\varepsilon\varepsilon_\delta^2\{S_2,\mathcal{H}_0\}-\varepsilon\varepsilon_\delta^3\{S_2,\mathcal{H}_1\}+O(\varepsilon_\delta^3),\\[2ex] \bar{\mathcal{H}}_2=\frac{1}{2}\Big(\varepsilon^2\varepsilon_\delta^2\{S_1,\{S_1,\mathcal{H}_0\}\}+\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_1,\mathcal{H}_1\}\}+\\ +\varepsilon^2\varepsilon_\delta^3\{S_1,\{S_2,\mathcal{H}_0\}\}+\varepsilon^2\varepsilon_\delta^4\{S_1,\{S_2,\mathcal{H}_1\}\}+\\ +\varepsilon^2\varepsilon_\delta^3\{S_2,\{S_1,\mathcal{H}_0\}\}+\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_1,\mathcal{H}_1\}\}+\\ +\varepsilon^2\varepsilon_\delta^4\{S_2,\{S_2,\mathcal{H}_0\}\}+\varepsilon^2\varepsilon_\delta^5\{S_2,\{S_2,\mathcal{H}_1\}\}\Big)+O(\varepsilon_\delta^3).\end{cases} \tag{3.68}$$

These equations can and must be simplified by removing all the terms of order ε_δ^3 or higher:

$$\begin{cases}\bar{\mathcal{H}}_0=\mathcal{H}_0+\varepsilon_\delta\mathcal{H}_1,\\[2ex] \bar{\mathcal{H}}_1=-\varepsilon\varepsilon_\delta\{S_1,\mathcal{H}_0\}-\varepsilon\varepsilon_\delta^2\{S_1,\mathcal{H}_1\}-\varepsilon\varepsilon_\delta^2\{S_2,\mathcal{H}_0\}+O(\varepsilon_\delta^3),\\[2ex] \bar{\mathcal{H}}_2=\frac{1}{2}\varepsilon^2\varepsilon_\delta^2\{S_1,\{S_1,\mathcal{H}_0\}\}+O(\varepsilon_\delta^3).\end{cases} \tag{3.69}$$

By gathering and ordering the terms using the powers of ε_δ, $\bar{\mathcal{H}}$ assumes this form:

$$\bar{\mathcal{H}} = \mathcal{H}_0 + \varepsilon_\delta\big(\mathcal{H}_1 - \varepsilon\{S_1, \mathcal{H}_0\}\big) + \\ -\varepsilon_\delta^2\bigg(\varepsilon\{S_1, \mathcal{H}_1\} + \varepsilon\{S_2, \mathcal{H}_0\} - \frac{1}{2}\varepsilon^2\{S_1, \{S_1, \mathcal{H}_0\}\}\bigg) + O(\varepsilon_\delta^3). \tag{3.70}$$

S_1 will be chosen to eliminate the fluctuating parts at order ε_δ in Eq. 3.70.
Before doing that, a particular notation is conveniently introduced here. It can be recalled in fact that the extended Poisson bracket of Eq. 3.44 has the first term which contains ε. It can be relabelled in this compact way:

$$\varepsilon^{-1}\{\mathcal{F}, \mathcal{G}\}_{(\theta,\rho)} = \frac{1}{\varepsilon eB\rho}\bigg(\frac{\partial\mathcal{F}}{\partial\theta}\frac{\partial\mathcal{G}}{\partial\rho} - \frac{\partial\mathcal{F}}{\partial\rho}\frac{\partial\mathcal{G}}{\partial\theta}\bigg), \tag{3.71}$$

where $\{\mathcal{F}, \mathcal{G}\}_{(\theta,\rho)}$ is called gyrobracket.
With regard to this, a useful consideration is that $\mathcal{H}_0$ actually depends on a limited amount of coordinates:

$$\begin{cases} \dfrac{\partial\mathcal{H}_0}{\partial X} = 0, \\ \dfrac{\partial\mathcal{H}_0}{\partial Y} = 0, \\ \dfrac{\partial\mathcal{H}_0}{\partial Z} = 0, \\ \dfrac{\partial\mathcal{H}_0}{\partial\theta} = 0, \\ \dfrac{\partial\mathcal{H}_0}{\partial t} = 0, \\ \dfrac{\partial\mathcal{H}_0}{\partial\rho} = \dfrac{e^2B^2}{m}\rho. \end{cases} \tag{3.72}$$

In fact, all these null derivatives suggest that any Poisson bracket acting on $\mathcal{H}_0$ (plus a generic observable $\mathcal{F}$) can be essentially reduced to the gyrobracket:

$$\{\mathcal{F}, \mathcal{H}_0\} = \varepsilon^{-1}\{\mathcal{F}, \mathcal{G}\}_{(\theta,\rho)}. \tag{3.73}$$

The term to look for in Eq. 3.70 is the second on the right-hand side, whose fluctuating part has to be cancelled through S_1. Using Eq. 3.73 and the definition of $\mathcal{H}_1$:

$$\mathcal{H}_1 - \{S_1, \mathcal{H}_0\}_{(\theta,\rho)} \longrightarrow \langle\mathcal{H}_1\rangle + \tilde{\mathcal{H}}_1 - \{S_1, \mathcal{H}_0\}_{(\theta,\rho)} \longrightarrow \tilde{\mathcal{H}}_1 - \{S_1, \mathcal{H}_0\}_{(\theta,\rho)} = 0, \tag{3.74}$$

$$\{S_1, \mathcal{H}_0\}_{(\theta,\rho)} = \tilde{\mathcal{H}}_1 \longrightarrow \frac{\partial S_1}{\partial \theta} = \frac{m}{Be}\tilde{\mathcal{H}}_1 \longrightarrow S_1 = \frac{m}{Be}\int \tilde{\mathcal{H}}_1 \, d\theta. \tag{3.75}$$

Given the choice of S_1, it has to have the same two slow dependencies on t and X_3 as the electrostatic potential contained in $\mathcal{H}_1$. This allows one to turn the Poisson brackets acting on it into gyrobrackets (still multiplied for ε^{-1}). With a bit of substitutions and semplifictations, the Hamiltonian of Eq. 3.70 is then turned into:

$$\bar{\mathcal{H}} = \mathcal{H}_0 + \varepsilon_\delta \langle \mathcal{H}_1 \rangle - \varepsilon_\delta^2 \bigg(\{S_1, \mathcal{H}_1\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} - \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}_{(\theta,\rho)} \bigg) + O(\varepsilon_\delta^3). \tag{3.76}$$

Determination of S_2

Now the terms at order ε_δ^2 are developed so that S_2 can be properly chosen. The first and the third of those terms can be rewritten in such a way that they can be partially combined. The main steps are shown below:

$$\begin{aligned}
&\{S_1, \mathcal{H}_1\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} - \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}_{(\theta,\rho)} = \\
&= \{S_1, \langle \mathcal{H}_1 \rangle\}_{(\theta,\rho)} + \{S_1, \tilde{\mathcal{H}}_1\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} - \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}_{(\theta,\rho)} = \\
&= \{S_1, \langle \mathcal{H}_1 \rangle\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} + \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}_{(\theta,\rho)} = \\
&= \{S_1, \langle \mathcal{H}_1 \rangle\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} + \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}^{(<>)}_{(\theta,\rho)} + \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}^{(\sim)}_{(\theta,\rho)}.
\end{aligned} \tag{3.77}$$

S_2 is then indirectly defined so that the following is true:

$$\{S_1, \langle \mathcal{H}_1 \rangle\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} + \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}^{(\sim)}_{(\theta,\rho)} = 0. \tag{3.78}$$

In short, this is the effect sorted by this choice:

$$\{S_1, \mathcal{H}_1\}_{(\theta,\rho)} + \{S_2, \mathcal{H}_0\}_{(\theta,\rho)} - \frac{1}{2}\{S_1, \{S_1, \mathcal{H}_0\}_{(\theta,\rho)}\}_{(\theta,\rho)} \longrightarrow \frac{1}{2}\{S_1, \tilde{\mathcal{H}}_1\}^{(<>)}_{(\theta,\rho)}. \tag{3.79}$$

Final form of the autonomized Hamiltonian

The surviving term at order ε_δ^2 should now be investigated. To do this, it is helpful to rearrange the derivatives associated with its Poisson bracket in this way:

$$\frac{\partial \mathcal{A}}{\partial \alpha}\frac{\partial \mathcal{B}}{\partial \beta} - \frac{\partial \mathcal{A}}{\partial \beta}\frac{\partial \mathcal{B}}{\partial \alpha} = \frac{\partial}{\partial \alpha}\bigg(\mathcal{A}\frac{\partial \mathcal{B}}{\partial \beta}\bigg) - \frac{\partial}{\partial \beta}\bigg(\mathcal{A}\frac{\partial \mathcal{B}}{\partial \alpha}\bigg), \tag{3.80}$$

where $\mathcal{A}$ and $\mathcal{B}$ are generic scalar functions, while α and β are generic variables. Making the Poisson bracket explicit and using Eq. 3.80 lead to:

$$\{S_1, \tilde{\mathcal{H}}_1\}^{(<>)}_{(\theta,\rho)} = -\{\tilde{\mathcal{H}}_1, S_1\}^{(<>)}_{(\theta,\rho)} = -\frac{1}{eB\rho}\frac{1}{2\pi}\int_0^{2\pi}\left(\frac{\partial\tilde{\mathcal{H}}_1}{\partial\theta}\frac{\partial S_1}{\partial\rho} - \frac{\partial\tilde{\mathcal{H}}_1}{\partial\rho}\frac{\partial S_1}{\partial\theta}\right)d\theta =$$

$$= \frac{1}{eB\rho}\frac{1}{2\pi}\int_0^{2\pi}\left[\frac{\partial}{\partial\rho}\left(\tilde{\mathcal{H}}_1\frac{\partial S_1}{\partial\theta}\right) - \frac{\partial}{\partial\theta}\left(\tilde{\mathcal{H}}_1\frac{\partial S_1}{\partial\rho}\right)\right]d\theta. \tag{3.81}$$

Given that θ is an angle, it comes with periodic boundary conditions, which means that the integral over the gyroangle of a derivative done with respect to the gyroangle itself is null. In particular:

$$\int_0^{2\pi}\frac{\partial}{\partial\theta}\left(\tilde{\mathcal{H}}_1\frac{\partial S_1}{\partial\rho}\right)d\theta = 0. \tag{3.82}$$

This allows one to write:

$$-\frac{1}{2}\varepsilon_\delta^2\{S_1, \tilde{\mathcal{H}}_1\}^{(<>)}_{(\theta,\rho)} = -\frac{\varepsilon_\delta^2}{2eB\rho}\frac{\partial}{\partial\rho}\left(\left\langle\tilde{\mathcal{H}}_1\frac{\partial S_1}{\partial\theta}\right\rangle\right) = -\frac{\varepsilon_\delta^2}{2eB\rho}\frac{m}{Be}\frac{\partial}{\partial\rho}\langle\tilde{\mathcal{H}}_1^2\rangle, \tag{3.83}$$

where use of Eq. 3.75 has been made. The final form of $\bar{\mathcal{H}}$, with no fluctuating terms below order ε_δ^3, can be outlined:

$$\bar{\mathcal{H}} = \mathcal{H}_0 + \varepsilon_\delta\langle\mathcal{H}_1\rangle - \frac{\varepsilon_\delta^2}{2eB\rho}\frac{m}{Be}\frac{\partial}{\partial\rho}\langle\tilde{\mathcal{H}}_1^2\rangle + O(\varepsilon_\delta^3) =$$

$$= \mathcal{H}_0 + e\varepsilon_\delta\langle\Phi\rangle - \frac{\varepsilon_\delta^2}{2eB\rho}\frac{m}{Be}e^2\frac{\partial}{\partial\rho}\langle\tilde{\Phi}^2\rangle + O(\varepsilon_\delta^3).$$

Here, $\mathcal{H}_1$ has been made explicit using its definition from Eqs. 3.46. If now two quantities are introduced, namely the guiding center electrostatic potential Ψ and the real parameter η, then:

$$\varepsilon_\delta\Psi = \varepsilon_\delta\langle\Phi\rangle - \frac{\eta\varepsilon_\delta^2}{B}\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\tilde{\Phi}^2\rangle, \tag{3.84}$$

$$\eta = \frac{m}{2eB} = \frac{1}{2\Omega}, \tag{3.85}$$

$$\bar{\mathcal{H}} = \mathcal{H}_0 + e\varepsilon_\delta\Psi + O(\varepsilon_\delta^3). \tag{3.86}$$

One of the assumptions was to have Φ small and therefore it was changed into $\varepsilon_\delta\Phi$: it is evident that the same is true for Ψ, which in fact appears multiplied by ε_δ.
The parameter η has the dimension of time and it is close to being the inverse of the Lamor frequency. In this sense, its absolute value can be seen as double of the time that a particle needs to completely rotate around its guiding center once. In the Hamiltonian, it measures the relevance of the second order term. In particular, when $\eta = 0$ the second

order is neglected. It is important to notice that, while for example ρ does not depend on the charge of the particle, η does. In this sense, it can assume both positive and negative values and this can lead to the birth of otherwise invisible asymmetries in the simulations.
A new parameter, which will be extensively used later, is called A and is obtained by dividing $\varepsilon_\delta \Psi$ for the amplitude of the magnetic field:

$$\frac{\varepsilon_\delta}{B}\Psi = \frac{\varepsilon_\delta}{B}\langle\Phi\rangle - \eta\frac{\varepsilon_\delta^2}{B^2}\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\tilde{\Phi}^2\rangle, \tag{3.87}$$

$$\langle\Phi\rangle = \frac{1}{2\pi}\int_0^{2\pi}\Phi d\theta, \tag{3.88}$$

$$A = \frac{\varepsilon_\delta}{B}. \tag{3.89}$$

The terms of Eq. 3.87 are not electrostatic potentials. In fact, they have the dimension of an electric potential divided by a magnetic field. They are rewritten with the introduction of A:

$$A\Psi = \langle A\Phi\rangle - \eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle(A\tilde{\Phi})^2\rangle \tag{3.90}$$

It is convenient to relabel both potentials to further simplify the notation:

$$\begin{cases} \text{potential: } \phi = \dfrac{\varepsilon_\delta}{B}\Phi = A\Phi, \\ \\ \text{guiding center potential: } \psi = \dfrac{\varepsilon_\delta}{B}\Psi = A\Psi. \end{cases} \tag{3.91}$$

As it will be later shown in this report, A is the amplitude of the potentials. Using these last notations, Eq. 3.90 becomes:

$$\psi = \langle\phi\rangle - \eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\tilde{\phi}^2\rangle. \tag{3.92}$$

Everything is now in place to finally determine the equations that govern the guiding center dynamics. This is achieved by combining the Poisson bracket of Eq. 3.44 and the Hamiltonian of Eq. 3.85:

$$\begin{cases} \dot{X}_1 = \{X_1, \bar{\mathcal{H}}\} = \{X_1, \bar{\mathcal{H}}\}_{(X_1,X_2)} = -\dfrac{\varepsilon_\delta}{B}\dfrac{\partial\Psi}{\partial X_2} = -\dfrac{\partial\psi}{\partial X_2}, \\ \\ \dot{X}_2 = \{X_2, \bar{\mathcal{H}}\} = \{X_2, \bar{\mathcal{H}}\}_{(X_1,X_2)} = \dfrac{\varepsilon_\delta}{B}\dfrac{\partial\Psi}{\partial X_1} = \dfrac{\partial\psi}{\partial X_1}, \end{cases} \tag{3.93}$$

where $\dot{\Box}$ stands for $d\Box/dt$, the definition of ψ has been used and $\{\Box, \bar{\mathcal{H}}\}_{(X_1,X_2)}$ is associated with the second term on the right-hand side of Eq. 3.44.

It is important to remind here that the quantities of Eqs. 3.93 are associated with the set of coordinates $\bar{\bar{\boldsymbol{z}}}_\tau$.
ψ can be made explicit to make ϕ appear:

$$\begin{cases} \dot{X}_1 = -\dfrac{\partial \psi}{\partial X_2} = -\dfrac{\partial}{\partial X_2}\left[\langle \phi \rangle - \eta \dfrac{1}{\rho}\dfrac{\partial}{\partial \rho}\langle \tilde{\phi}^2 \rangle\right], \\ \\ \dot{X}_2 = \dfrac{\partial \psi}{\partial X_1} = \dfrac{\partial}{\partial X_1}\left[\langle \phi \rangle - \eta \dfrac{1}{\rho}\dfrac{\partial}{\partial \rho}\langle \tilde{\phi}^2 \rangle\right]. \end{cases} \tag{3.94}$$

An additional notation, which will be extensively used, can be taken into consideration:

$$\begin{cases} \psi_{\mathrm{GC}_1} = \langle \phi \rangle, \\ \\ \psi_{\mathrm{GC}_2} = -\eta \dfrac{1}{\rho}\dfrac{\partial}{\partial \rho}\langle \tilde{\phi}^2 \rangle, \end{cases} \tag{3.95}$$

where the subscript GC stands for *Guiding Center* and allows one to visually split ψ into its first and second order terms (with respect to the powers of ϕ):

$$\psi = \psi_{\mathrm{GC}_1} + \psi_{\mathrm{GC}_2}. \tag{3.96}$$

This same idea about the representation of ψ is used in the Python scripts, too. Therefore, Eqs. 3.94 are turned into:

$$\begin{cases} \dot{X}_1 = -\dfrac{\partial \psi_{\mathrm{GC}_1}}{\partial X_2} - \dfrac{\partial \psi_{\mathrm{GC}_2}}{\partial X_2}, \\ \\ \dot{X}_2 = +\dfrac{\partial \psi_{\mathrm{GC}_1}}{\partial X_1} + \dfrac{\partial \psi_{\mathrm{GC}_2}}{\partial X_1}. \end{cases} \tag{3.97}$$

The set of equations for the guiding center dynamics that has been obtained validates the equations that can be found in [Cha21]. The other aspect that required a check is the actual form assumed by ψ for this particular work.

3.5 Characterization of the guiding center potential

It is evident from the result of the previous section that guiding center dynamics rely on ψ or, equivalently, on ϕ. From now on, some coordinates are relabelled:

$$\begin{cases} (x_1, x_2, x_3) = (x, y, z), \\ \\ (X_1, X_2, X_3) = (X, Y, Z). \end{cases} \tag{3.98}$$

3.5.1 Nondimensionalization of quantities and equations

?? The step associated with the implementation of the theoretical part into the Python scripts was originally direct, i.e. without an explicit section dedicated to obtaining dimensionless equations. It was implicit, closely following what can be found in many related works, where this portion is only briefly mentioned (see [Cir+04] and [Pet+88]). Here, this procedure is more extensively presented instead.
The primary aspect that drives this section is the mathematical model of the turbulent electrostatic potential Φ: a truncated series based on the shapes of the electrostatic potential of real tokamaks. However, it has to be intended as a toy model:

$$\Phi(x,y,t) = \sum_{\substack{n,m=1 \\ n^2+m^2\leq M^2}}^{M} \frac{1}{(n^2+m^2)^{3/2}} \sin\left(\frac{2\pi}{L}(nx+my) - \varphi_{nm} - \frac{2\pi}{T}t\right). \tag{3.99}$$

In this series:

- The generic modes are represented by the couple (n, m).
- L and T are the spatial and temporal periods of Φ.
- φ_{nm} is a phase whose value is randomly picked between 0 and 2π. Random phases are used to emphasize the turbulent nature of the potential.
- There are two limitations on the modes, with the first being $(1,1) \leq (n,m) \leq (M,M)$ and second being $n^2+m^2 \leq M^2$.

Φ is time-dependent and constructed in the transverse plane. In fact, it does not vary along the direction z of magnetic field lines. This represents something stricter than what has been said in the previous section, but it still fits that assumption as a particular case.
The following *conversion* formulae are mostly based on Eq. 3.99 and allow one to get dimensionless quantities in such a way that the equations for guiding center dynamics keep their original form:

$$\hat{x} = 2\pi\frac{x}{L}, \tag{3.100}$$

$$\hat{y} = 2\pi\frac{y}{L}, \tag{3.101}$$

$$\hat{X} = 2\pi\frac{X}{L}, \tag{3.102}$$

$$\hat{Y} = 2\pi\frac{Y}{L}, \tag{3.103}$$

$$\hat{\rho} = 2\pi\frac{\rho}{L}, \tag{3.104}$$

$$\hat{t} = 2\pi \frac{t}{T}, \tag{3.105}$$

$$\hat{\eta} = 2\pi \frac{\eta}{T}, \tag{3.106}$$

$$\hat{B} = \frac{B}{B_0}, \tag{3.107}$$

$$\hat{A} = \frac{\varepsilon_\delta}{\hat{B}} = B_0 A, \tag{3.108}$$

$$\hat{\Phi} = \frac{\Phi}{\Phi_0}, \quad \text{with } \Phi_0 = \frac{L^2}{T}\frac{B_0}{2\pi}, \tag{3.109}$$

$$\hat{\Psi} = \frac{\Psi}{\Phi_0}. \tag{3.110}$$

In general, $\hat{\square}$ is the dimensionless version of $\square$, while B_0 is a rescaling value of the magnetic field amplitude that can be chosen in accordance with real physical data. By using these conversions, it is possible to obtain the dimensionless versions of some relevant equations:

$$\hat{\Phi} = \sum_{\substack{n,m=1 \\ n^2+m^2 \leq M^2}}^{M} \frac{1}{(n^2+m^2)^{3/2}} \sin\left(n\hat{x} + m\hat{y} - \varphi_{nm} - \hat{t}\right), \tag{3.111}$$

$$\begin{cases} \dot{\hat{X}} = -\dfrac{\partial \hat{\psi}}{\partial \hat{Y}}, \\ \\ \dot{\hat{Y}} = \dfrac{\partial \hat{\psi}}{\partial \hat{X}}. \end{cases} \tag{3.112}$$

The dimensionless potentials are given by:

$$\hat{\phi} = \frac{\varepsilon_\delta}{\hat{B}} \hat{\Phi}, \tag{3.113}$$

$$\hat{\psi} = \frac{\varepsilon_\delta}{\hat{B}} \hat{\Psi}. \tag{3.114}$$

For example, this is how the equation for $\dot{\hat{X}}$ is obtained:

$$\dot{X}_1 = -\frac{\varepsilon_\delta}{B}\frac{\partial \Psi}{\partial X_2} \longrightarrow \dot{\hat{X}} = -\frac{T}{L}\frac{\varepsilon_\delta}{B_0 \hat{B}}\frac{2\pi}{L}\Phi_0 \frac{\partial \hat{\Psi}}{\partial \hat{Y}} =$$

$$= -\frac{T}{L}\frac{\varepsilon_\delta}{B_0 \hat{B}}\frac{2\pi}{L}\Phi_0 \frac{\partial \hat{\Psi}}{\partial \hat{Y}} = -\left(\frac{T}{L}\frac{\varepsilon_\delta}{B_0 \hat{B}}\frac{2\pi}{L}\;\frac{L^2}{T}\frac{B_0}{2\pi}\right)\frac{\partial \hat{\Psi}}{\partial \hat{Y}} = -\frac{\varepsilon_\delta}{\hat{B}}\frac{\partial \hat{\Psi}}{\partial \hat{Y}} = -\frac{\partial \hat{\psi}}{\partial \hat{Y}}. \tag{3.115}$$

As already hinted, the equations of motion before and after the change of units have the exact same form. Therefore, starting from the next section of this report:

- All the new quantities that will be introduced will be considered dimensionless.
- All dimensionless quantities, old and new, will be labelled without the $\hat{\Box}$ notation.

This simplification has been made possible by the specific definition of Φ_0. Once the dimensionless results of a simulation are available, it is possible to recover the dimensional quantities provided that the followings are known, chosen, provided or used:

- The *conversion* formulae above, i.e. from Eq. 3.100 to Eq. 3.110.
- The values of L, T, B_0 and ε_δ.
- The actual perpendicular plasma temperature (needed for the square root of velocities that features in the definition of the Larmor radius).
- The nature of test particles, i.e. their mass m and charge e.

3.5.2 Model for the dimensionless potential

Based on Eqs. 3.111 and 3.113, the mathematical model for ϕ, inspired by the one used in [Pet+88], is given by:

$$\phi(x, y, t) = \sum_{\substack{n,m=1 \\ n^2+m^2 \leq M^2}}^{M} \frac{A}{(n^2+m^2)^{3/2}} \sin(nx + my + \varphi_{nm} - t). \tag{3.116}$$

It is a turbulent potential, in the sense that it is characterized by eddies that cyclically arise at certain spots in the transverse plane, then move in space over time and eventually disappear. The presence of random phases ensures some degree of unpredictability that is responsible for the presence of eddies, while the nondimensionalization procedure makes it a potential with periods of 2π both in time and space. In particular, a window on the (x, y) plane that goes from 0 to 2π in each of the two axial directions, which can be called *fundamental square*, is sufficient to represent the entire potential. This fact also has positive implications on the way scripts are written, but this will be discussed later. It seems appropriate here (and only here) to show what the guidelines of subsection **??** refer to. The rigorous way of writing the *dimensionless* potential would be:

$$\hat{\phi}(\hat{x}, \hat{y}, \hat{t}) = \sum_{\substack{n,m=1 \\ n^2+m^2 \leq M^2}}^{M} \frac{\hat{A}}{(n^2+m^2)^{3/2}} \sin(n\hat{x} + m\hat{y} + \varphi_{nm} - \hat{t}), \tag{3.117}$$

but both the notation and the full name of the potential would prove to be tedious, considering the amount of steps to go through.

Moving away from this brief consideration, the idea is to write ϕ in terms of a complex potential ϕ_c. This will allow, after some steps, to have an expression for ψ that can be efficiently introduced in the Python scripts. The starting point for this derivation is the following formula, which can be be easily verified:

$$\phi(x,y,t) = \Im\big[\phi_\mathrm{c}(x,y)\mathrm{e}^{-\mathrm{j}t}\big] = \Im[\phi_\mathrm{c}(x,y)]\cos(t) - \Re[\phi_\mathrm{c}(x,y)]\sin(t), \tag{3.118}$$

where real and imaginary parts are indicated with $\Re[\square]$ and $\Im[\square]$, respectively. Moreover, the complex potential is defined as:

$$\phi_\mathrm{c}(x,y) = \sum_{\substack{n,m=1\\ n^2+m^2\leq M^2}}^{M} K_{nm}\mathrm{e}^{\mathrm{j}(nx+my)}, \tag{3.119}$$

with

$$K_{nm} = \frac{A\mathrm{e}^{\varphi_{nm}}}{(n^2+m^2)^{3/2}}. \tag{3.120}$$

Using the Larmor vector of Eq. 3.23, it is possible to rewrite Eqs. 3.117, 3.118 and 3.119 in this way:

$$\phi(x,y,t) = \phi(X+\rho\cos\theta, Y-\rho\sin\theta, t) =$$
$$= \sum_{\substack{n,m=1\\ n^2+m^2\leq M^2}}^{M} \frac{A}{(n^2+m^2)^{3/2}}\sin\big[n(X+\rho\cos\theta)+m(Y-\rho\sin\theta)+\varphi_{nm}-t\big], \tag{3.121}$$

$$\phi(x,y,t) = \phi(X+\rho\cos\theta, Y-\rho\sin\theta, t) =$$
$$= \Im[\phi_\mathrm{c}(X+\rho\cos\theta, Y-\rho\sin\theta)]\cos(t) - \Re[\phi_\mathrm{c}(X+\rho\cos\theta, Y-\rho\sin\theta)]\sin(t), \tag{3.122}$$

$$\phi_\mathrm{c}(x,y) = \phi_\mathrm{c}(X+\rho\cos\theta, Y-\rho\sin\theta) = \sum_{\substack{n,m=1\\ n^2+m^2\leq M^2}}^{M} K_{nm}\mathrm{e}^{\mathrm{j}(nX+mY)}\mathrm{e}^{\mathrm{j}(n\rho\cos\theta-m\rho\sin\theta)}. \tag{3.123}$$

3.5.3 First order of the guiding center potential

Because of the relations ϕ, ψ can be expressed in terms of complex potentials too. First, ψ_{GC_1} is considered:

$$\psi_{\mathrm{GC}_1} = \psi_{\mathrm{GC}_{1,1}} = \Im[\psi_{\mathrm{GC}_{1,c}}\mathrm{e}^{-\mathrm{j}t}], \tag{3.124}$$

with

$$\psi_{\mathrm{GC}_{1,c}} = \big\langle\phi_\mathrm{c}(X+\rho\cos\theta, Y-\rho\sin\theta)\big\rangle =$$
$$= \sum_{\substack{n,m=1\\ n^2+m^2\leq M^2}}^{M} K_{nm}\mathrm{e}^{\mathrm{j}(nX+mY)}\Big\langle\mathrm{e}^{\mathrm{j}(n\rho\cos\theta-m\rho\sin\theta)}\Big\rangle. \tag{3.125}$$

At this point, the gyroaveraging has to be actively performed. This will make Bessel functions appear. More informations about them can be found in [Bar05]. In order to see how this happens, it is important to recall two possible definitions for the Bessel function of the first kind J_ν:

$$J_\nu(s) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{j}(s\sin\beta - \nu\beta)} d\beta, \tag{3.126}$$

$$J_\nu(s) = \sum_{k=0}^{\infty} (-1)^k \frac{(s/2)^{2k+\nu}}{k!\,\Gamma(k+\nu+1)}. \tag{3.127}$$

In these definitions:

- s is a generic variable.
- ν is an integer number that constitutes the order of the Bessel function.
- Γ is the Gamma function.

A useful relation for Γ is the following:

$$\Gamma(a+1) = a!, \quad \forall a \in \mathbb{I}. \tag{3.128}$$

In the particular case of $\nu = 0$, then the two definitions become:

$$J_0(s) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{j}(s\sin\beta)} d\beta = \frac{1}{2\pi}\int_{0}^{2\pi} \mathrm{e}^{\mathrm{j}(s\sin\beta)} d\beta, \quad \sin\beta = \cos\left(\beta + \pi/2\right), \tag{3.129}$$

$$J_0(s) = \sum_{k=0}^{\infty} (-1)^k \frac{(s/2)^{2k}}{(k!)^2}. \tag{3.130}$$

At this point, the gyroaveraging is developed as presented below:

$$\left\langle \mathrm{e}^{\mathrm{j}(n\rho\cos\theta - m\rho\sin\theta)} \right\rangle = \left\langle \mathrm{e}^{\mathrm{j}\frac{\sqrt{n^2+m^2}}{\sqrt{n^2+m^2}}(n\rho\cos\theta - m\rho\sin\theta)} \right\rangle =$$

$$= \left\langle \exp\left[\mathrm{j}\rho\sqrt{n^2+m^2}\left(\frac{n}{\sqrt{n^2+m^2}}\cos\theta - \frac{m}{\sqrt{n^2+m^2}}\sin\theta \right) \right] \right\rangle. \tag{3.131}$$

The quantities $n/\sqrt{n^2+m^2}$ and $m/\sqrt{n^2+m^2}$ can be expressed in terms of trigonometric functions:

$$\begin{cases} \cos\theta_{nm} = \dfrac{n}{\sqrt{n^2+m^2}}, \\ \\ \sin\theta_{nm} = \dfrac{m}{\sqrt{n^2+m^2}}. \end{cases} \tag{3.132}$$

By exploiting Eq. 3.129 and the well known formula for the cosine of a sum

$$\cos(\theta + \theta_{nm}) = \cos\theta\cos\theta_{nm} - \sin\theta\sin\theta_{nm}, \tag{3.133}$$

it is possible to reformulate Eq. 3.131 as:

$$\left\langle \mathrm{e}^{\mathrm{j}(n\rho\cos\theta - m\rho\sin\theta)} \right\rangle = \left\langle \mathrm{e}^{\mathrm{j}\rho\sqrt{n^2+m^2}\cos(\theta+\theta_{nm})} \right\rangle =$$

$$= \frac{1}{2\pi}\int_0^{2\pi} \mathrm{e}^{\mathrm{j}\rho\sqrt{n^2+m^2}\cos(\theta+\theta_{nm})} d\theta = J_0\big(\rho\sqrt{n^2+m^2}\big). \tag{3.134}$$

In can also be verified that the way in which the integral definition of J_0 has been exploited relies on the value assumed by β:

$$\begin{cases} \beta = \theta + \theta_{nm} - \pi/2, \\ \\ d\beta = d\theta, \\ \\ \sin\beta = \cos(\beta + \pi/2) = \cos(\theta + \theta_{nm}). \end{cases} \tag{3.135}$$

To further reduce complexity, the notation

$$s_{nm} = \sqrt{n^2 + m^2} \tag{3.136}$$

is introduced. Here, the symbol s helps reminding the presence of a square root. This allows one to write:

$$J_0\big(\rho\sqrt{n^2+m^2}\big) = J_0(\rho s_{nm}). \tag{3.137}$$

Putting everything together gives:

$$\psi_{\mathrm{GC}_{1,c}}(X,Y) = \sum_{\substack{n,m=1 \\ n^2+m^2 \leq M^2}}^{M} K_{nm} J_0(\rho s_{nm}) \mathrm{e}^{\mathrm{j}(nX+mY)}. \tag{3.138}$$

It is important to note here that, since θ has been averaged out, ρ is now considered a parameter (like A, for example) and not a variable (like X or Y). Obviously, this is true even for the following subsection, in which the second order term of the potential is considered.

3.5.4 Second order of the guiding center potential

The second term associated with ψ is:

$$\psi_{\mathrm{GC}_2} = -\eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\tilde{\phi}^2\rangle, \tag{3.139}$$

in which $\tilde{\phi}$ can be written in this form:

$$\langle\tilde{\phi}^2\rangle = \langle\phi^2\rangle - \langle\phi\rangle^2. \tag{3.140}$$

Having to work with two terms instead of just one may seem more difficult, but the absence of $\tilde{\Box}$ is really helpful. This substitution modifies Eq. 3.139:

$$\psi_{\mathrm{GC}_2} = -\eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\big(\langle\phi^2\rangle - \langle\phi\rangle^2\big) = -\eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\phi^2\rangle + \eta\frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\phi\rangle^2. \tag{3.141}$$

New symbols can be assigned to the two terms on the right-hand side:

$$\begin{cases} \psi^{(\mathrm{I})}_{\mathrm{GC}_2} = +\eta\dfrac{1}{\rho}\dfrac{\partial}{\partial\rho}\langle\phi^2\rangle, \\ \\ \psi^{(\mathrm{II})}_{\mathrm{GC}_2} = +\eta\dfrac{1}{\rho}\dfrac{\partial}{\partial\rho}\langle\phi\rangle^2. \end{cases} \tag{3.142}$$

This allows one to write:

$$\psi_{\mathrm{GC}_2} = -\psi^{(\mathrm{I})}_{\mathrm{GC}_2} + \psi^{(\mathrm{II})}_{\mathrm{GC}_2}, \tag{3.143}$$

with some attention that has to be put on the different signs.
It is also useful to give a name to the product between the complex potential ϕ_c and $\mathrm{e}^{-\mathrm{j}t}$:

$$\varphi_{\mathrm{c},\tau}(X, Y, \rho, \theta, t) = \phi_{\mathrm{c}}(X, Y, \rho, \theta)\mathrm{e}^{-\mathrm{j}t}, \tag{3.144}$$

where the subscript τ accounts for the presence of time. This opens up to these relations, whose validity can be easily verified:

$$\begin{cases} \phi = \Im\big[\varphi_{\mathrm{c},\tau}\big], \\ \\ \phi = \dfrac{\varphi_{\mathrm{c},\tau} - \varphi^*_{\mathrm{c},\tau}}{2\mathrm{j}}, \\ \\ \phi^2 = \dfrac{\varphi^2_{\mathrm{c},\tau} + \varphi^{*2}_{\mathrm{c},\tau} - 2\varphi_{\mathrm{c},\tau}\varphi^*_{\mathrm{c},\tau}}{-4} = \dfrac{\varphi^2_{\mathrm{c},\tau} + \varphi^{*2}_{\mathrm{c},\tau} - 2|\varphi_{\mathrm{c},\tau}|^2}{-4} = \dfrac{1}{2}|\varphi_{\mathrm{c},\tau}|^2 - \dfrac{1}{2}\Re[\varphi^2_{\mathrm{c},\tau}], \\ \\ \langle\phi\rangle^2 = \dfrac{1}{2}|\langle\varphi_{\mathrm{c},\tau}\rangle|^2 - \dfrac{1}{2}\Re[\langle\varphi_{\mathrm{c},\tau}\rangle^2], \\ \\ \langle\phi^2\rangle = \dfrac{1}{2}\langle|\varphi_{\mathrm{c},\tau}|^2\rangle - \dfrac{1}{2}\Re[\langle\varphi^2_{\mathrm{c},\tau}\rangle], \end{cases} \tag{3.145}$$

where $\Box^*$ is the complex conjugated of $\Box$. The last two of these can be implemented into Eqs. 3.142 so that the following is obtained:

$$\begin{cases} \psi^{(\mathrm{I})}_{\mathrm{GC}_2} = +\dfrac{\eta}{2\rho}\dfrac{\partial}{\partial\rho}\big(\langle|\varphi_{\mathrm{c},\tau}|^2\rangle - \Re[\langle\varphi^2_{\mathrm{c},\tau}\rangle]\big), \\ \\ \psi^{(\mathrm{II})}_{\mathrm{GC}_2} = +\dfrac{\eta}{2\rho}\dfrac{\partial}{\partial\rho}\big(|\langle\varphi_{\mathrm{c},\tau}\rangle|^2 - \Re[\langle\varphi_{\mathrm{c},\tau}\rangle^2]\big). \end{cases} \tag{3.146}$$

At this point, if the notation

$$\mathbb{J}(\Box) = \frac{1}{\rho}\frac{\partial}{\partial\rho}\langle\Box\rangle \tag{3.147}$$

is introduced, Eqs. 3.146 become:

$$\begin{cases} \psi^{(\mathrm{I})}_{\mathrm{GC}_2} = +\dfrac{\eta}{2}\Big(\mathbb{J}\big(|\varphi_{\mathrm{c},\tau}|^2\big) - \Re[\mathbb{J}(\varphi^2_{\mathrm{c},\tau})]\Big), \\ \\ \psi^{(\mathrm{II})}_{\mathrm{GC}_2} = +\dfrac{\eta}{2}\dfrac{\partial}{\partial\rho}\big(\langle\varphi_{\mathrm{c},\tau}\rangle\langle\varphi^*_{\mathrm{c},\tau}\rangle - \Re[\langle\varphi_{\mathrm{c},\tau}\rangle^2]\big) = \\ = +\dfrac{\eta}{2}\Big(\langle\varphi^*_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau}) + \langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi^*_{\mathrm{c},\tau}) - \Re[2\langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau})]\Big). \end{cases} \tag{3.148}$$

If both terms are put back into Eq. 3.143, then the following expression for the second order of the potential is obtained:

$$\begin{aligned} \psi_{\mathrm{GC}_2} = \frac{\eta}{2}\Big(&- \mathbb{J}\big(|\varphi_{\mathrm{c},\tau}|^2\big) + \Re[\mathbb{J}(\varphi^2_{\mathrm{c},\tau})]\langle\varphi^*_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau}) + \\ &+\langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi^*_{\mathrm{c},\tau}) - \Re[2\langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau})]\Big) = \\ = \frac{\eta}{2}\Big(&- \mathbb{J}\big(|\varphi_{\mathrm{c},\tau}|^2\big) + \langle\varphi^*_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau}) + \\ &\langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi^*_{\mathrm{c},\tau}) + \Re\big[\mathbb{J}(\varphi^2_{\mathrm{c},\tau}) - 2\langle\varphi_{\mathrm{c},\tau}\rangle\mathbb{J}(\varphi_{\mathrm{c},\tau})\big]\Big). \end{aligned} \tag{3.149}$$

3.5.5 Relabelling of some terms to account for their numerical implementation

The guiding center potential can be written in this form:

$$\psi = \psi_{\mathrm{GC}_{1,1}} + \psi_{\mathrm{GC}_{2,0}} + \psi_{\mathrm{GC}_{2,2}}, \tag{3.150}$$

where

$$\begin{cases} \psi_{\mathrm{GC}_{1,1}} = +\Im[\langle\phi_\mathrm{c}\rangle \mathrm{e}^{-1\mathrm{j}t}], \\[2ex] \psi_{\mathrm{GC}_{2,0}} = -\dfrac{\eta}{2}\big(\mathbb{J}\big(|\phi_\mathrm{c}|^2\big) - \langle\phi_\mathrm{c}^*\rangle\mathbb{J}(\phi_\mathrm{c}) - \langle\phi_\mathrm{c}\rangle\mathbb{J}(\phi_\mathrm{c}^*)\big)\mathrm{e}^{-0\mathrm{j}t}, \\[2ex] \psi_{\mathrm{GC}_{2,2}} = +\dfrac{\eta}{2}\Re\big[\big(\mathbb{J}(\phi_\mathrm{c}^2) - 2\langle\phi_\mathrm{c}\rangle\mathbb{J}(\phi_\mathrm{c})\big)\mathrm{e}^{-2\mathrm{j}t}\big]. \end{cases} \tag{3.151}$$

The form used to represent the time-dependent exponential parts has been chosen to highlight the meaning of subscripts. For example, $\psi_{\mathrm{GC}_{2,0}}$ means *portion of second order (2) of the guiding center (GC) potential that has time, in the exponential part, that is multiplied by zero (0).*

$$\begin{cases} \dot{X} = -\dfrac{\partial\psi_{\mathrm{GC}_{1,1}}}{\partial Y} - \dfrac{\partial\psi_{\mathrm{GC}_{2,0}}}{\partial Y} - \dfrac{\partial\psi_{\mathrm{GC}_{2,2}}}{\partial Y}, \\[2ex] \dot{Y} = +\dfrac{\partial\psi_{\mathrm{GC}_{1,1}}}{\partial X} + \dfrac{\partial\psi_{\mathrm{GC}_{2,0}}}{\partial X} + \dfrac{\partial\psi_{\mathrm{GC}_{2,2}}}{\partial X}. \end{cases} \tag{3.152}$$

The general idea of the quantities hidden in the system of Eqs. 3.152 is the following:

$$\langle\square\rangle = \sum_{\substack{n,m=1 \\ n^2+m^2\leq M^2}}^{M} \hat{\square}_{nm} J_0(\rho s_{nm})\mathrm{e}^{+j(nX+mY)}, \tag{3.153}$$

$$\mathbb{J}(\square) = -\sum_{\substack{n,m=1 \\ n^2+m^2\leq M^2}}^{M} \hat{\square}_{nm} \frac{s_{nm}}{\rho} J_1(\rho s_{nm})\mathrm{e}^{+j(nX+mY)}, \tag{3.154}$$

where $\square$ is a generic quantity (e.g. ϕ_c) and $\hat{\square}_{nm}$ is a generic coefficient (e.g. K_{nm}). Moreover, if $\rho = 0$, the numerical implementation of $\mathbb{J}(\square)$ can exploit this limit:

$$\lim_{\rho\to 0} \frac{s_{nm}}{\rho} J_1(\rho s_{nm}) = \frac{s_{nm}^2}{2}. \tag{3.155}$$

Numerically, left-hand sides of Eqs. 3.153 and 3.154 can be related to their associated coefficients $\hat{\square}_{nm}$ with the help of the Fast Fourier Transform algorithm and its inverse in two dimensions (see Appendix B and section 4.3).

Chapter 4

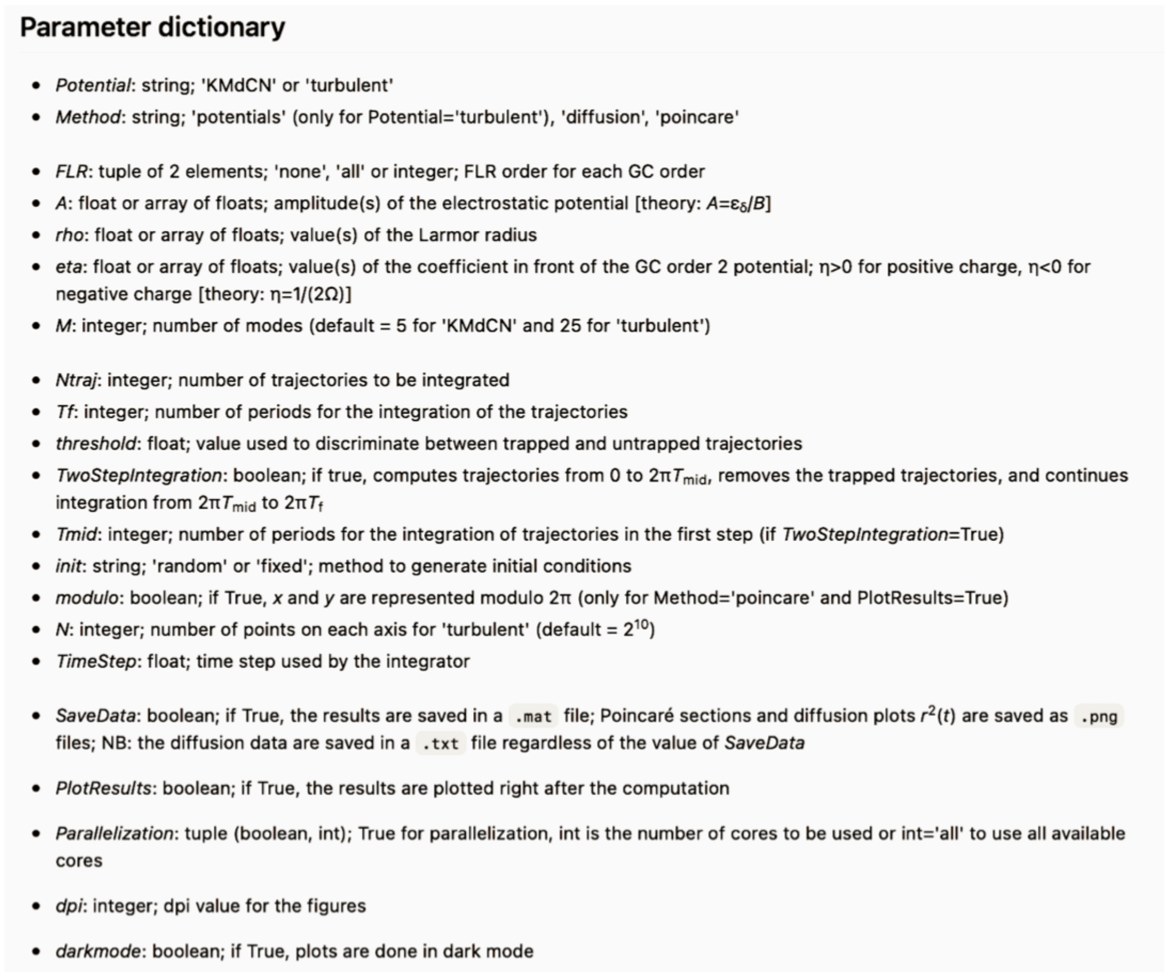
Numerical part

4.1 General overview

This chapter presents the numerical scripts used to carry out the simulations and to analyze the outputs. The first are associated with three scripts in Python, while the analysis was done with two scripts in MATLAB in order to produce results and provide feedback to improve the Python files.
Unless otherwise noted, this chapter is associated with the final versions of the scripts. However, from time to time, references or comparisons will be made with older versions of them. In general, the simulations can be divided into three categories: those that allow one to plot the potentials (`potentials` mode), those that allow one to plot the temporal evolution of guiding centers in the transverse plane (`poincare` mode) and those that provide information about the way in which these guiding centers diffuse (`diffusion` mode).
Some fundamental aspects of this section are the Fast Fourier Transform algorithm in two dimensions and the anomalous diffusion of particles. They are presented in Appendices B and C.
As for the three scripts in Python:

- `gc2d_dict.py` constitutes the interface with the user, where it is possible to set the values of the parameters for the simulations.
- `gc2d.py` contains the physics of the problem, in the sense that it has the main functions to define the guiding center dynamics. In essence, it revolves around the system of Eqs. 3.152.
- `gc2d_modules.py` is the one that collects the various steps that the computer must perform during the simulations. In particular, it involves the methods to integrate the guiding center dynamics and the instructions to provide the outputs.

Moving now to the MATLAB scripts, `reader_diffusion.m` deals with analyzing the outputs associated with `diffusion` simulations, while `reader_poincare.m` does the same for `poincare` simulations. What has just been presented here as an introductory overview will be fully described in the following.



Parameter dictionary

- *Potential*: string; 'KMdCN' or 'turbulent'
- *Method*: string; 'potentials' (only for Potential='turbulent'), 'diffusion', 'poincare'

- *FLR*: tuple of 2 elements; 'none', 'all' or integer; FLR order for each GC order
- *A*: float or array of floats; amplitude(s) of the electrostatic potential [theory: $A=\varepsilon_\delta/B$]
- *rho*: float or array of floats; value(s) of the Larmor radius
- *eta*: float or array of floats; value(s) of the coefficient in front of the GC order 2 potential; $\eta>0$ for positive charge, $\eta<0$ for negative charge [theory: $\eta=1/(2\Omega)$]
- *M*: integer; number of modes (default = 5 for 'KMdCN' and 25 for 'turbulent')

- *Ntraj*: integer; number of trajectories to be integrated
- *Tf*: integer; number of periods for the integration of the trajectories
- *threshold*: float; value used to discriminate between trapped and untrapped trajectories
- *TwoStepIntegration*: boolean; if true, computes trajectories from 0 to $2\pi T_{\mathrm{mid}}$, removes the trapped trajectories, and continues integration from $2\pi T_{\mathrm{mid}}$ to $2\pi T_{\mathrm{f}}$
- *Tmid*: integer; number of periods for the integration of trajectories in the first step (if *TwoStepIntegration*=True)
- *init*: string; 'random' or 'fixed'; method to generate initial conditions
- *modulo*: boolean; if True, *x* and *y* are represented modulo 2π (only for Method='poincare' and PlotResults=True)
- *N*: integer; number of points on each axis for 'turbulent' (default = 2^{10})
- *TimeStep*: float; time step used by the integrator

- *SaveData*: boolean; if True, the results are saved in a `.mat` file; Poincaré sections and diffusion plots $r^2(t)$ are saved as `.png` files; NB: the diffusion data are saved in a `.txt` file regardless of the value of *SaveData*

- *PlotResults*: boolean; if True, the results are plotted right after the computation

- *Parallelization*: tuple (boolean, int); True for parallelization, int is the number of cores to be used or int='all' to use all available cores

- *dpi*: integer; dpi value for the figures

- *darkmode*: boolean; if True, plots are done in dark mode

Figure 4.1: Parameter dictionary as it appears in the `README.md` file that accompanies the Python scripts.

4.2 Description of `gc2d_dict.py`

The `gc2d_dict.py` file constitutes the parameter dictionary: it allows the user to select the values of some fundamental parameters. Fig. 4.1 shows a list of parameters with a short comment, as they are reported in the `README.md` that can be found in the same folder as the Python files. In the following list, the same parameters are described more in detail:

- `Potential`: it allows one to select the type of potential. With reference to this work, this parameter has never been modified and has always been left set as `turbulent` so as to implement in Python the turbulent potential equations developed in Chapter 3. Everything concerning the other mode has not been considered, neither theoretically nor numerically.

- `Method`: with this parameter it is possible to choose the type of simulation by assigning it one of the strings among `potentials`, `diffusion` and `poincare`. The first works only if `Potential = 'turbulent'` and causes the simulation output to be some plots associated with ϕ and ψ. In the cases of `Method = 'diffusion'` and `Method = 'poincare'`, the potentials are used to calculate the evolution over time of guiding centers. However, in the `diffusion` case the main output is a `.txt` file that contains some quantities of interest associated with transport of particles, while in the `poincare` case the output is a graph showing the trajectories of guiding centers in the transverse plane. The three modes and their outputs will be better characterized in section 4.4.

- `FLR`: this parameter is associated with a pair of values (one for the first and one for the second order of ψ), each of which can be a positive integer, `'all'` or `'none'`. `FLR` stands for *Finite Larmor Radius* and it allows one to take into account the approximation that consists in confusing the position of guiding centers with those of the associated particles. To do this, it is necessary to set `FLR = ('none', 'none')`, which means that ρ is zero in all of ψ (in reality ρ can never be zero when $e, m, B > 0$). The result of this is that the first order of ψ has $J_0(0) = 1$ and the second order has $s_{nm}J_1(_{nm})/\rho = s_{nm}^2/2$, as it can be easily checked by looking at section 3.5.5. In the case of `eta = 0`, then having no FLR effect leads to $\psi = \phi$ (with the only theoretical difference that one has (X, Y) and the other (x, y) as spatial coordinates). If, on the other hand, other values are used (i.e. a positive and *finite* Larmor radius is used), then interesting effects of the Bessel functions are taken into account. In particular, if `FLR = ('all', 'all')`, then the Bessel functions are implemented in their entirety in Python. If a pair of numerical values is used, then the Bessel functions are approximated with a finite number of terms of the series with which they can be defined (see section 4.3) In general, more complicated pairs can be chosen, such as `FLR = ('all', 'none')` or even `FLR = ('none', 5)`.

- `A`: it is the dimensionless amplitude of the turbulent potential and obviously corresponds to A from the theory. Single numeric values or multiple values can be assigned to it. This second case is normally implemented through the `numpy.linspace` function, which allows one to create a vector of equally spaced elements. If `A` is a vector, launching a simulation means launching a series of sub-simulations, one for each value of `A`. The same is also true when other parameters are defined as vectors rather than single values. It is important to notice here that even if A is described as an amplitude, the potentials in which it features could have smaller actual amplitudes because of other terms, e.g. $e^{j\varphi_{nm}}$ and $(n^2+m^2)^{-3/2}$.

- `rho`: it is the Larmor radius associated with the test particles and its values can be assigned in the same way as for `A`. It corresponds to ρ. When `FLR` is not equal to `(0,0)`, it is possible to confuse the particle with its guiding center by setting `rho = 0` or, more generally, by including the value `0` inside a vector obtained with `numpy.linspace`. This last possibility is particularly useful when you want to compare the effect of a non-zero Larmor radius with a zero one. However, using `rho` as a vector does not mean that the test particles have different radii during the same simulation: each `rho` value corresponds to a single sub-simulation in which all test particles have the same Larmor radius.

- `eta`: it is the coefficient that takes into account the weight of the second order of ψ in the equations of the guiding centers and its values can be assigned in the same way as for `A`. It corresponds to η. Unlike `A` and `rho`, `eta` can also assume negative values since it depends on the charge of test particles. When `eta = 0`, then simulations do not take into account the second order of the potential.

- `M`: it is the number of modes (along each direction of the transverse plane) with which the truncated series of the turbulent potential is constructed. It corresponds to M from the theory. `M` is one of those parameters that has a default value even if the user does not directly assign it, and it is `25` in the case of `Potential = 'turbulent'`.

- `Ntraj`: it is the number of trajectories of the guiding centers that are simulated when `Method = 'potentials'` or `Method = 'poincare'`.

- `Tf`: it is the number of cycles for the integration of the `Ntraj` trajectories, that is, it is the number of temporal periods of the potential that are considered for the simulations. Each of the periods, based on how the nondimensionalization was carried out in the theory, is worth 2π. At the end of the k-th cycle (or period), time is equal to $2\pi k$.

- `threshold`: it is the `threshold` value used to distinguish between two kinds of trajectories: trapped, which do not contribute to diffusion, and untrapped. From now on, trajectory will specifically mean *the collection of positions, of a single guiding center, registered only at* $t = 0$ *and and the end of each temporal cycle.* The criterion by which this parameter is used is closely related to `gc2d_modules.py`, while its value can be fixed based on some analysis with MATLAB (see subsection 4.8.4). In any case, it can be thought of as a dimensionless distance in the transverse plane.

- `TwoStepIntegration`: it is the parameter that indicates the way in which the simulation of guiding centers is conducted. If it is set as `False`, all `Ntraj` guiding centers are followed for all `Tf` cycles. If instead it is set as `True`, then `Ntraj` guiding centers are followed up to an intermediate number of cycles <`Tf` and then only the untrapped ones are followed for the remaining cycles. This parameter is particularly important because it allows one to considerably reduce the time and computational efforts and therefore will be discussed further on.

- `Tmid`: it is the intermediate number of cycles (it has to be less than `Tf`) which is used if `TwoStepIntegration = True`. This parameter, like `threshold` and others, can be chosen more efficiently after some post-processing with MATLAB (see subsection 4.8.5).

- `init`: with this parameter it is possible to choose the initial conditions to be associated with the equations for the guiding centers. It fixes the positions from which the guiding centers start in the transverse plane. If `init = 'random'`, the initial positions are chosen at random within the fundamental square $[0, 2\pi] \times [0, 2\pi]$, while, if `init = 'fixed'`, the positions are preset and to form a sort of structured grid. In the latter case, the number of trajectories that are actually followed may be different from the one manually assigned to `Ntraj`. In particular, the new value of `Ntraj` becomes equal to the integer part of the square root of `Ntraj`.

- `modulo`: when this parameter is set to `True` and in case of `poincare` simulations with, stroboscopic plots are referred to the fundamental square and the modulo function is used. Standard plots are produced instead if `modulo = False`.

- `N`: it is the number of points on the axes of the transverse spatial plane where the guiding centers are studied. It is also the number of Fourier coefficients associated with the numerical implementation of the potential. Its default value is `2**10`.

- `TimeStep`: it is the time step used by the Python integrator to solve the equations for the guiding center position. The smaller this number is, the more accurate and slower the solution is to obtain.

- `SaveData`: when this parameter is set to `True`, then the results of simulations are saved in a `.mat` file, which for example can be easily imported in MATLAB as a workspace. Even in the case of `SaveData = False`, some data obtained in `diffusion` mode are still saved in a `.txt` file.

- `PlotResults`: if `True`, the results are plotted right after the computation. In particular, this allows one to plot stroboscopic plots in `poincare` mode, potentials in `potentials` mode and the mean square displacement of guiding centers as a function of time in `diffusion` mode.

- `Parallelization`: this setting requires two assignments and allows one to activate the parallelization of sub-simulations associated with a single simulations. If the first value given to is `True`, then the second value is the number of processes to run in parallel. Its value has to be a positive integer or `'all'`. If the chosen number is higher than the number of available cores of the computer, then it is reduced accordingly. Instead, if `'all'` is used, then the number of parallel processes is exactly equal to the number of available cores.

- `dpi`: it has to be an integer and it is associated with the resolution of images obtained with the simulations. The name `dpi` stands for *dots per inch.*

- `darkmode`: if `True`, all plots have a black background (colors of text and plotted quantities are automatically set to be readable). Otherwise, if `False`, plots have a white background (see Fig. 4.10 as an example).

4.3 Description of `gc2d.py`

This file contains the definitions useful for calculating the potentials and, consequently, the evolution over time of guiding centers in the transverse plane. This file is used to carry out a series of instructions that appear in the `gc2d_modules.py` file, which is presented in the 4.4. In addition to physical and modelling aspects, a part of this file is dedicated to the parallelization of the sub-simulations.

4.3.1 Evaluation of the potentials alternating between physical and Fourier spaces

The numerical implementation of potentials is done by exploiting the physical space (the transverse plane) and the Fourier space (the frequency domain). This is done due to the nature of the turbulent potential ϕ, which is periodic. The definition of ϕ_c, which is part of ϕ, can be traced back to a truncated Fourier series and therefore its numerical version can be associated with Discrete Fourier Transform. ϕ_c, and all its related potentials, can therefore be very well approximated using the 2D Fast Fourier Transform algorithm (and its inverse).

Physical and Fourier spaces

The number of points with which the transverse plane is discretized is equal to $\mathtt{N} \times \mathtt{N}$ and only refers to the fundamental square. This creates a squared structured mesh with an equal number of points along each coordinate. This mesh refers to both the (x, y) and the (X, Y) coordinates of the theory, but in the file the same symbols `(x,y)` are used interchangeably. This is possible since the two orders of ψ_c contain, in their series, the same exponential part of ϕ_c if, in the latter, the instances (x, y) are replaced with (X, Y). The equality of exponential parts derives from the averaging procedure which simultaneously eliminates the angle of rotation from the equations and makes the Larmor radius a parameter (from being a variable). It should be emphasized that it is possible to limit the physical mesh just to the fundamental square due to the periodicity of the potential, which can be greatly exploited by the use of the modulo function in Python, which is indicated as %.
The physical space defined for this problem is naturally associated with a Fourier space involving a number of wave numbers and Fourier coefficients equal to the number of nodes: $\mathtt{N} \times \mathtt{N}$. Fourier space is used to evaluate potentials and their spatial derivatives, in a very accurate way, with the help of the Fast Fourier Transform algorithm in two dimensions (`numpy.fft.fft2`) and its inverse (`numpy.fft.ifft2`). It is intuitive that, in order to have a good degree of approximation of the potential and therefore of the temporal evolution of guiding centers, it is important that `N` is very high. When applying `numpy.fft.fft2` to an $\mathtt{N} \times \mathtt{N}$ matrix that approximates a function in physical space, a matrix of Fourier coefficients of equal size is then obtained in Fourier space.

From ϕ_c to an inverse Discrete Fourier Transform in two dimensions

Given what has been just said, it is clear that discretized versions of any potential of the theory:

- Need to be defined in the $\mathtt{N} \times \mathtt{N}$ points of the fundamental square.
- Need to have $\mathtt{N} \times \mathtt{N}$ coefficients in Fourier space.

The building block of any potential of the theory is ϕ_c, which is defined through Eq. 3.119. From that equation, it is evident that:

- The numerical version of ϕ_c can be defined, without problems, in the $\mathtt{N} \times \mathtt{N}$ points of the fundamental square. This means that applying a direct Fourier transform to it would lead to $\mathtt{N} \times \mathtt{N}$ coefficients in Fourier space.
- The series of ϕ_c has no more than M^2 coefficients.
- The series of ϕ_c is very similar to an inverse Discrete Fourier Transform in two dimensions.

Therefore, in order to obtain the matrix that approximates 3.119, what must be solved is the relation between K_{nm} (i.e. the coefficients of ϕ_c) and the Fourier coefficients of the numerical version of ϕ_c. By looking at the expressions for ϕ_c and a generic inverse Discrete Fourier Transform in two dimensions, the solution to the problem is quite easy to obtain:

$$\phi_c(x, y) = \sum_{\substack{n,m=1 \\ n^2+m^2 \leq M^2}}^{M} K_{nm} \mathrm{e}^{j(nx+my)}, \tag{4.1}$$

$$a_{nm} = \frac{1}{N^2} \sum_{k,l=0}^{N-1} A_{kl} \mathrm{e}^{\mathrm{j}(kx_n+ly_m)}, \tag{4.2}$$

where $N = \texttt{N}$, $M = \texttt{M}$ and, therefore, $N > M$.
The idea is that the series of ϕ_c has to be extended without actually changing it:

$$\phi_c(x, y) = \frac{1}{N^2} \sum_{n,m=0}^{N-1} c_{nm} \mathrm{e}^{j(nx+my)}, \tag{4.3}$$

where

- $c_{nm} = 0$ if $(n, m) < (1, 1)$ or $(n, m) > (M, M)$ or $n^2 + m^2 \geq M^2$.
- $c_{nm} = N^2 K_{nm}$ otherwise.

After that, by regularly discretizing (x, y) in the fundamental square with

$$\boldsymbol{x} = (x_0, ..., x_{N-1}) \tag{4.4}$$

and

$$\boldsymbol{y} = (y_0, ..., y_{N-1}), \tag{4.5}$$

it is possible to reach the formulation of an inverse Discrete Fourier Transform in two dimensions for the complex potential. In `gc2d.py`:

- The numerical matrix that accounts for ϕ_c is called `phi`. It is an inverse Discrete Fourier Transform in two dimensions.
- The numerical matrix that accounts for c_{nm}/N^2 is called `fft_phi`. It represents the Fourier coefficients of `phi` its non-zero terms are associated with $K_n m$.

Put simply, `phi` is defined in Python by applying the inverse Fast Fourier Transform algorithm in two dimensions (`numpy.fft.ifft2`) to `(N**2)*fft_phi`, i.e. to c_{nm}.

Definition of the Fourier coefficients

In light of what has been explained, the matrix of Fourier coefficients is defined with a series of steps, which are presented in Listing 2.

```python
import numpy as xp
from numpy.fft import fftfreq
from gc2d_modules import run_method
from gc2d_dict import dict_list
xp.random.seed(27)
phases = 2 * xp.pi * xp.random.random((self.M, self.M))
n = xp.meshgrid(xp.arange(1, self.M+1), xp.arange(1, self.M+1),
↪  indexing='ij')
self.xy_ = 2 * (xp.linspace(0, 2 * xp.pi, self.N+1, dtype=xp.float64),)
nm = xp.meshgrid( fftfreq (self.N, d=1/self.N), fftfreq (self.N, d=1/self.N),
↪  indexing='ij')
sqrt_nm = xp.sqrt(nm[0]**2 + nm[1]**2)
fft_phi = xp.zeros((self.N, self.N), dtype=xp.complex128)
fft_phi[1:self.M+1, 1:self.M+1] = (self.A / (n[0]**2 +
↪  n[1]**2)**1.5).astype(xp.complex128) * xp.exp(1j * phases)
fft_phi[sqrt_nm > self.M] = 0
```

Listing 2: Portion of `gc2d.py`. The $\mathtt{N} \times \mathtt{N}$ matrix of Fourier coefficients `fft_phi` is defined. The use of `numpy.fft.fftfreq` is described in Appendix B.

Evaluation of all potentials and their spatial derivatives starting from Fourier coefficients

By exploiting the $\mathtt{N}^2$ Fourier coefficients, the potentials are defined in rapid sequence. Then, they can be graphically represented or exploited to determine the temporal evolution of trajectories. It is possible to reconstruct the three matrices that are used to approximate the terms of ψ_c as shown in Listing 3.

```python
import numpy as xp
from numpy.fft import ifft2
from gc2d_modules import run_method
from gc2d_dict import dict_list
if (self.FLR[0] == 'none') or (self.FLR[0] in range(2)):
        flr1_coeff = 1
elif self.FLR[0] == 'all':
        flr1_coeff = jv(0, self.rho * sqrt_nm)
elif isinstance(self.FLR[0], int):
        x = sp.Symbol('x')
        flr_expansion = sp.besselj(0, x).series(x, 0,
        ↪  self.FLR[0]+1).removeO()
        flr_func = sp.lambdify(x, flr_expansion)
        flr1_coeff = flr_func(self.rho * sqrt_nm)
fft_phi_gc1_1 = flr1_coeff * fft_phi
self.phi = ifft2(fft_phi) * (self.N**2)
self.phi_gc1_1 = ifft2(fft_phi_gc1_1) * (self.N**2)
```

Listing 3: Portion of `gc2d.py` that comes immediately after Listing 2. Fourier coefficients are exploited to calculate the complex potential ϕ_c and $\psi_{GC_{1,c}}$ (i.e. `phi` and `phi_gc1_1`). The effect of considering guiding centers is taken into account with `flr1_coeff`, which in its full version is associated with $J_0(\rho s_{nm})$.

```
import numpy as xp
from numpy.fft import fft2, ifft2
from gc2d_modules import run_method
from gc2d_dict import dict_list
if (self.FLR[1] == \texttt{none}) or (self.FLR[1] in range(3)) or (self.rho
↪   == 0):
        flr2_coeff = - sqrt_nm**2 / 2
else:
        if self.FLR[1] == \texttt{all}:
        flr2_coeff = - sqrt_nm * jv(1, self.rho * sqrt_nm) / self.rho
        elif isinstance(self.FLR[1], int):
                x = sp.Symbol('x')
                flr_exp = sp.besselj(1, x).series(x, 0,
                ↪   self.FLR[1]+1).removeO()
                flr2_coeff = - sqrt_nm * sp.lambdify(x, flr_exp)(self.rho *
                ↪   sqrt_nm) / self.rho
self.flr2 = lambda psi: ifft2(fft2(psi) * flr2_coeff)
self.phi_gc2_0 = - self.eta * (self.flr2(xp.abs(self.phi)**2) -
↪   self.phi_gc1_1 * self.flr2(self.phi.conjugate()) -
↪   self.phi_gc1_1.conjugate() * self.flr2(self.phi)).real / 2
self.phi_gc2_2 = - self.eta * (self.flr2(self.phi**2) - 2 * self.phi_gc1_1 *
↪   self.flr2(self.phi)) / 2
derivs = lambda psi: [xp.pad(ifft2(1j * nm[_] * fft2(psi)), ((0, 1),),
↪   mode='wrap') for _ in range(2)]
self.dphidx_gc1_1, self.dphidy_gc1_1 = derivs(self.phi_gc1_1)
self.dphidx_gc2_0, self.dphidy_gc2_0 = derivs(self.phi_gc2_0)
self.dphidx_gc2_2, self.dphidy_gc2_2 = derivs(self.phi_gc2_2)
```

Listing 4: Portion of `gc2d.py` that comes immediately after Listing 3. Fourier coefficients are exploited to calculate the time-independent factors of the second order terms of ψ (i.e. `phi_gc2_0` and `phi_gc2_2`). The effect of considering guiding centers is taken into account with `flr2_coeff`, which in its full version is associated with $-(s_{nm}/\rho)J_1(\rho s_{nm})$. Complex potentials are then used to calculate their spatial derivatives at mesh nodes with the definition of `derivs`.

4.3.2 Updating the positions of guiding centers with interpolations in the fundamental square

The three terms of the complex potential ψ_c can be used to quickly and accurately calculate, through fast transforms, spatial derivatives of *psi* at any given point of the transverse plane (these points, that correspond to the positions of guiding centers at any given time, are folded into the fundamental square through the modulo function). This is carried out by exploiting the `scipy.interpolate.interpn` function, which acts on the spatial derivatives that are available at the `N` $\times$ `N` nodes.

```
import numpy as xp
from scipy.interpolate import interpn
from gc2d_modules import run_method
from gc2d_dict import dict_list
def eqn_phi(self, t, y):
        yr = xp.array(xp.split(y, 2)).transpose() % (2 * xp.pi)
        dphidx = interpn(self.xy_, self.dphidx_gc1_1, yr)
        dphidy = interpn(self.xy_, self.dphidy_gc1_1, yr)
        dy_gc1 = xp.concatenate((-(dphidy * xp.exp(-1j * t)).imag, (dphidx *
        ↪  xp.exp(-1j * t)).imag), axis=None)
        if self.GCorder == 1:
                return dy_gc1
        elif self.GCorder == 2:
                dphidx_0 = interpn(self.xy_, self.dphidx_gc2_0, yr)
                dphidy_0 = interpn(self.xy_, self.dphidy_gc2_0, yr)
                dphidx_2 = interpn(self.xy_, self.dphidx_gc2_2, yr)
                dphidy_2 = interpn(self.xy_, self.dphidy_gc2_2, yr)
                dy_gc2 = xp.concatenate((-dphidy_0.real + (dphidy_2 *
                ↪  xp.exp(-2j * t)).real, dphidx_0.real - (dphidx_2 *
                ↪  xp.exp(-2j * t)).real), axis=None)
                        return dy_gc1 + dy_gc2
```

Listing 5: Portion of `gc2d.py` that defines the equations of guiding center dynamics that are integrated in `gc2d_modules.py` with `scipy.integrate.solve_ivp`. The use of the modulo function % folds actual positions of guiding centers inside the fundamental square. There, modular spatial derivatives (needed to update the positions of guiding centers) are approximated by interpolating the values of derivatives of time-independent complex potentials known at mesh nodes. Derivatives of first and second orders of ψ are then obtained by taking time into account.

4.4 Description of `gc2d_modules.py`

This file contains the instructions to integrate the equations of motion and to produce the different kinds of outputs, based on the parameter dictionary. This is the file that has undergone the greatest amount of changes compared to its original form.

4.4.1 Creating animations of the potentials

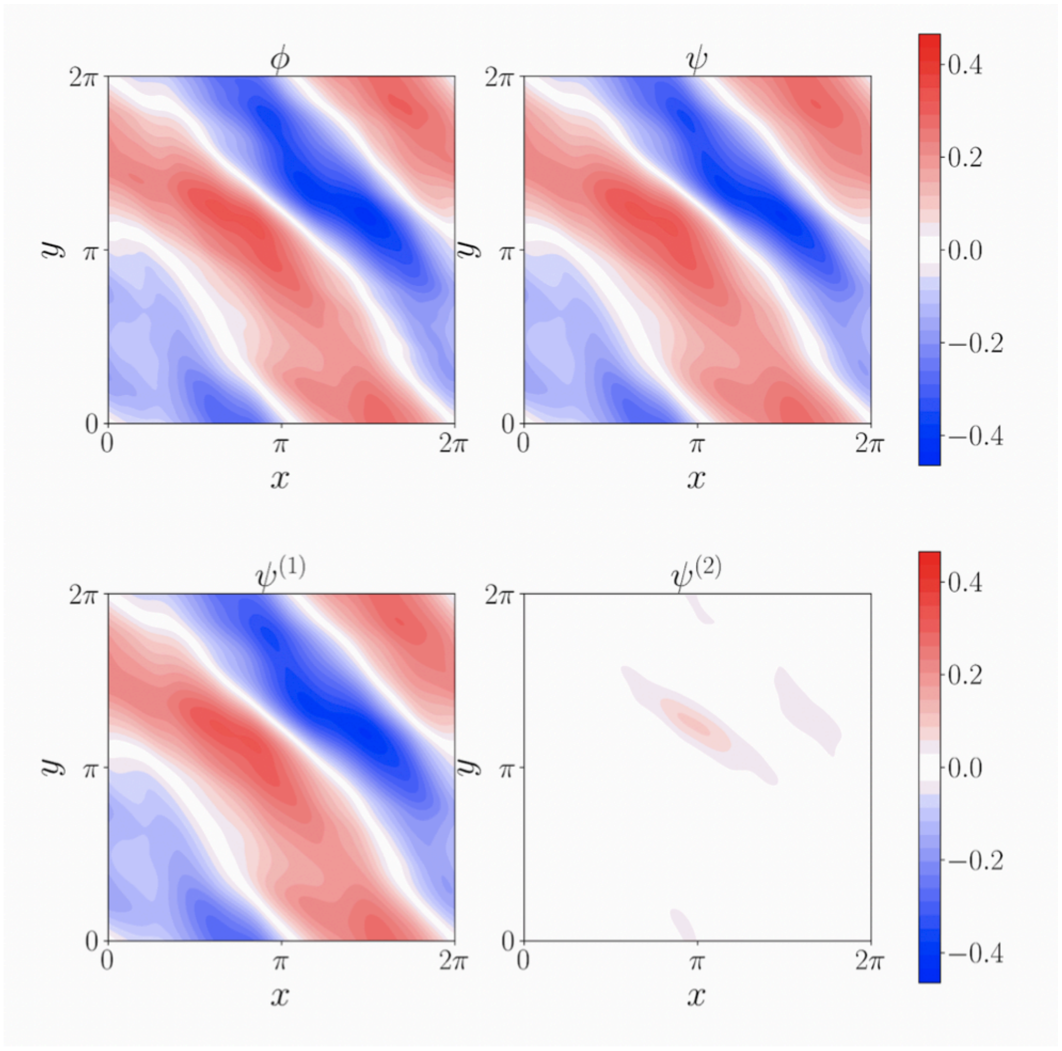


Figure 4.2: Potentials for a system with $A = 0.700$, $\rho = 0.115$ and $\eta = -0.100$. $\psi^{(1)}$ and $\psi^{(2)}$ correspond to ψ_{GC_1} and ψ_{GC_2}, respectively.

In the section that is activated only if `Method = 'potentials'` and `Potential = 'turbulent'`, up to four animated plots are produced: one for ϕ, one for ψ, one for the first order of ψ and one for the second order of ψ. All the graphs are produced within the fundamental square and refer to a single termporal cycle starting at $t = 0$. This representation is sufficient to characterize the potentials given their periodicity in both space and time. A single frame of the animations is displayed in Fig. 4.2 A graphical extension in space of any of the potentials would result in something similar to what is shown in Fig. 4.3. The extension is highlighted by the use of gray scales. The animations are saved as `.gif` files. The effects on the potentials of the three fundamental parameters `A`, `rho` and `eta`, as well as of other quantities such as the phases, are discussed in section 4.7.

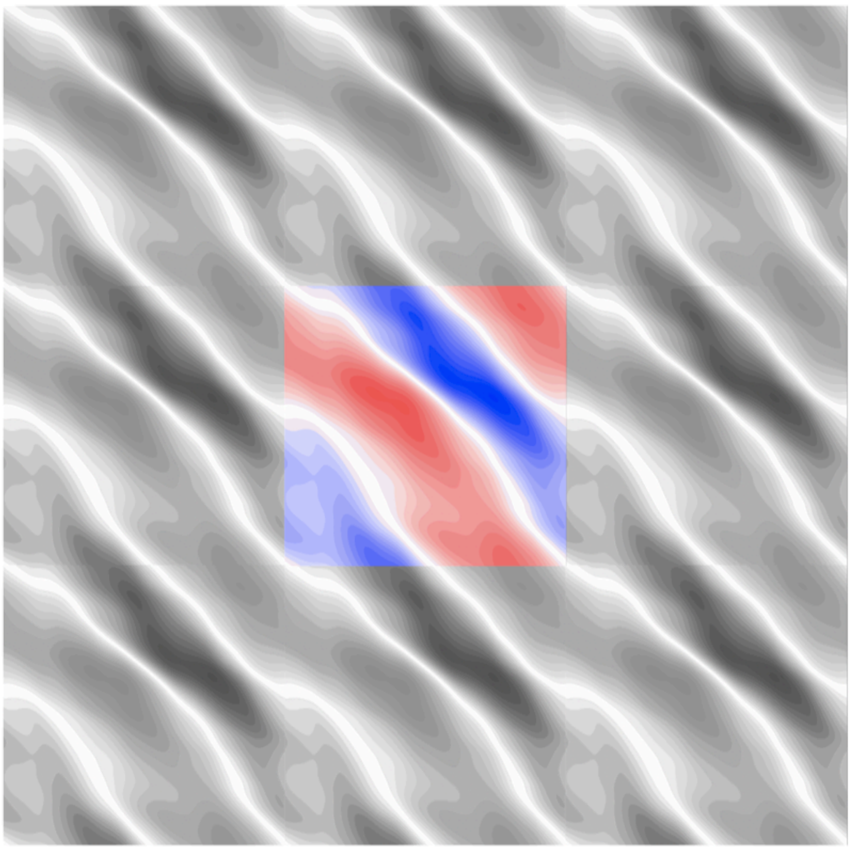


Figure 4.3: Partial extension of ψ from Fig. 4.2 outside the fundamental square. This allows one to visually show the spatial periodicity of potentials. In this case, the potential covers a square with sides equal to 6π.

4.4.2 Multicolored stroboscopic plots to highlight different behaviors

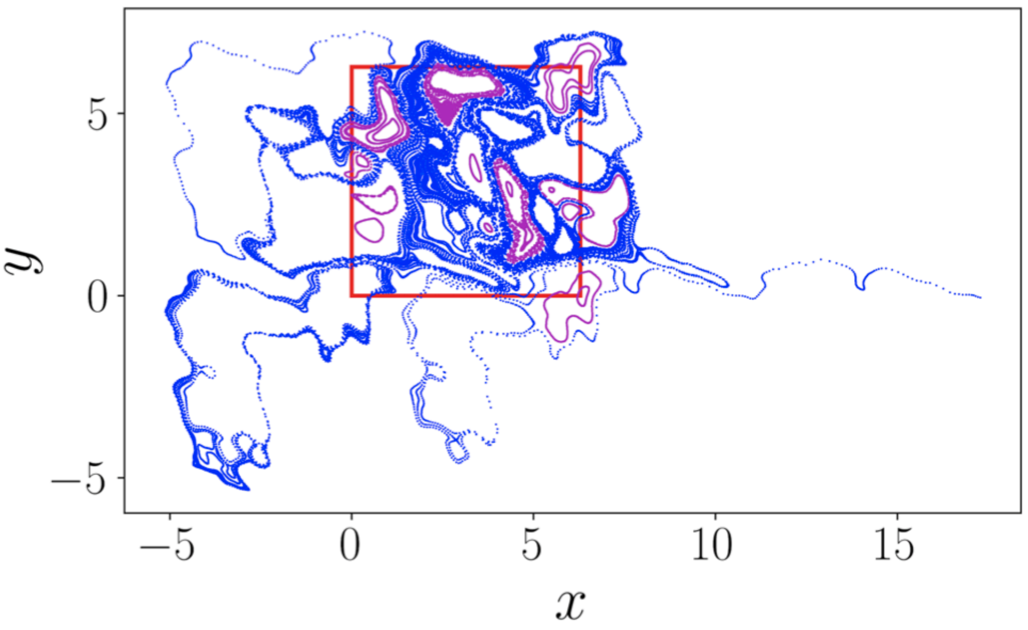


Figure 4.4: Multicolored non-modular stroboscopic plot obtained with Python using `A = 0.600`, `rho = 0.015` and `eta = 0.000`. Trapped trajectories (`threshold = 4`) are in magenta, while the others are in blue. The fundamental square is highlighted in red (this is a minor addition with respect to the original version of Python scripts, but it is helpful to get an idea of where guiding centers are spawned at $t = 0$). For better clarity, only 64 test particles and 500 cycles have been used (`TimeStep = 0.05`).

When `Method = poincare`, the aim is to produce stroboscopic plots like the one of Fig. 4.4. A single one of them represents, in the transverse plane, the positions of all guiding centers at $t = 0$ and then at the end of each cycle. It should be remembered here that, in reality, particles do not move only in the transverse plane, but also along the direction of the magnetic field. However, in addition to the fact that the position along z at time t is always known in a trivial way, what is of interest is precisely the movement in the transverse plane, because the more the particles move radially away from their initial position, the more the effectiveness of the magnetic confinement is reduced.
The word *stroboscopic* refers to the fact that the positions of guiding centers are recorded in the graph in synchronism with the period of the potential (i.e. the promoter of the motion through the system of Eqs. 3.152). Instead, the `poincare` instance that appears in the file, which actually stands for *Poincaré*, refers to the fact that the stroboscopic plots

are linked to Poincaré sections, in the sense that they are particular cases of Poincaré sections in which time is considered as the dynamical variable and whose associated equation is $t = 0$ (modulo 2π). In the following, the terms *stroboscopic plot* and *Poincaré section* will be used interchangeably. In this work I decided to use multicolored plots with two colors: one of them is used to highlight the trajectories of trapped particles, while the other is associated with the untrapped ones. This allows one to better analyze the system and its behavior.
A more complete investigation of this system is obtained by combining the analysis of curve fitting plots with the investigation of stroboscopic plots. The former tries to unravel the general characteristics of the system, while the latter allows one to more specifically look for what may be the causes of potentially anomalous diffusion.
An aspect that differentiates the needs related to `diffusion` and `poincare` methods is number of trajectories to simulate: to have a curve fitting as accurate as possible, it is important to have both `Ntraj` and `Tf` sufficiently large, while to graphically analyze trajectories, it is possible to use a smaller number of test particles (`Tf` still needs to be high in most cases).
Another relevant consideration is that, having to deal with less trajectories, it could be a good idea to use single step integrations for stroboscopic plots, so that the true evolutions of all trajectories can be followed. For example, this could be used to detect those rare particles that suddenly go from having a non-diffusive behavior to being untrapped.
Last but not least, the tendency is to launch simulations with a high number of sub-simulations in `diffusion` mode, while stroboscopic plots are usually produced with more focused simulations. This helps to make `poincare` an overall lighter feature to use.

4.4.3 Standard and modular stroboscopic plots

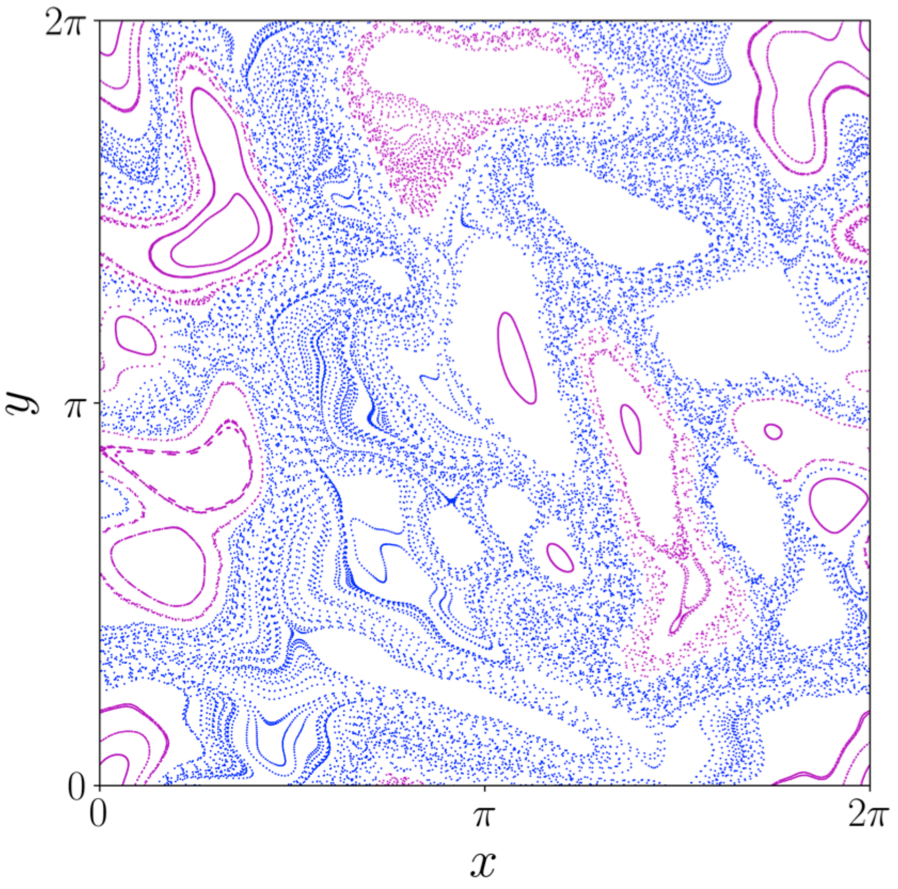


Figure 4.5: Modular version of the stroboscopic plot of Fig. 4.4.

The periodic nature of the potential makes it possible to create modular stroboscopic plots in addition to the normal ones. Modular graphs are limited to the fundamental square and are obtained through the aforementioned modulo function acting on the true positions of guiding centers. Non-modular figures allow one to have a more immediate idea of the type of diffusion regime and of the actual distance traveled by the particles, while the modular ones allow one to focus more on the visual differences between trapped and untrapped trajectories. In addition, by looking at them, it is possible to qualitatively identify the degree of chaos of the system with little effort. Although Poincaré sections can be produced with Python, notably with the package `matplotlib`, they are better analyzed through MATLAB for a variety of reasons, such as the ability to interact more easily with figures and the possibility to highlight trajectories with more colors, even according to criteria different from the trapped-untrapped one used in Python.

```python
import numpy as xp
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
cs = ['w', 'k', 'b', 'm']
if case.Method == 'poincare':
        if case.PlotResults:
                fig, ax = plt.subplots(1, 1)
                ax.set_xlabel('$x$')
                ax.set_ylabel('$y$')
                if case.modulo:
                        ax.plot(x_un % (2 * xp.pi), y_un % (2 * xp.pi),
                        ↪  '.', color=cs[2], markersize=2,
                        ↪  markeredgecolor='none')
                        ax.plot(x_tr % (2 * xp.pi), y_tr % (2 * xp.pi),
                        ↪  '.', color=cs[3], markersize=2,
                        ↪  markeredgecolor='none')
                        ax.set_xlim(0, 2 * xp.pi)
                        ax.set_ylim(0, 2 * xp.pi)
                if not case.modulo:
                        ax.plot(x_un, y_un, '.', color=cs[2], markersize=2,
                        ↪  markeredgecolor='none')
                        ax.plot(x_tr, y_tr, '.', color=cs[3], markersize=2,
                        ↪  markeredgecolor='none')
                        ax.add_patch(Rectangle((0, 0), 2 * xp.pi, 2 * xp.pi,
                        ↪  facecolor='None', edgecolor='r', lw=2))
```

Listing 6: Selected portion of `gc2d_modules.py` that is used to produce modular and non-modular stroboscopic plots. If `darkmode = False`, trapped trajectories (i.e. `x_tr` and `y_tr` are in magenta, while the untrapped ones (i.e. `x_un` and `y_un`) are in blue. If `modulo = True`, then the modulo function % is used to fold actual trajectories into the fundamental square. If `modulo = False`, the edges of the fundamental square are highlighted in red.

4.4.4 Integration of the equations of motion in one or two steps

The second part of the file involves the integration of the equations for the guiding centers. It is associated with both `Method = 'diffusion'` and `Method = 'poincare'`. The user, through `gc2d_dict.py`, can choose whether to carry out the integration in one or two steps. In the first case, the idea is that `Ntraj` guiding centers are followed during all the `Tf` cycles. Instead, in the second case:

1. `Ntraj` guiding centers are followed for `Tmid` cycles.
2. Untrapped guiding centers are separated from the trapped on the basis of the value of `threshold`.
3. Untrapped guiding centers are integrated for the remaining cycles, up to a total of `Tf` cycles. In the case of one step only, trapped particles are separated from the untrapped only at the very end of the integration.

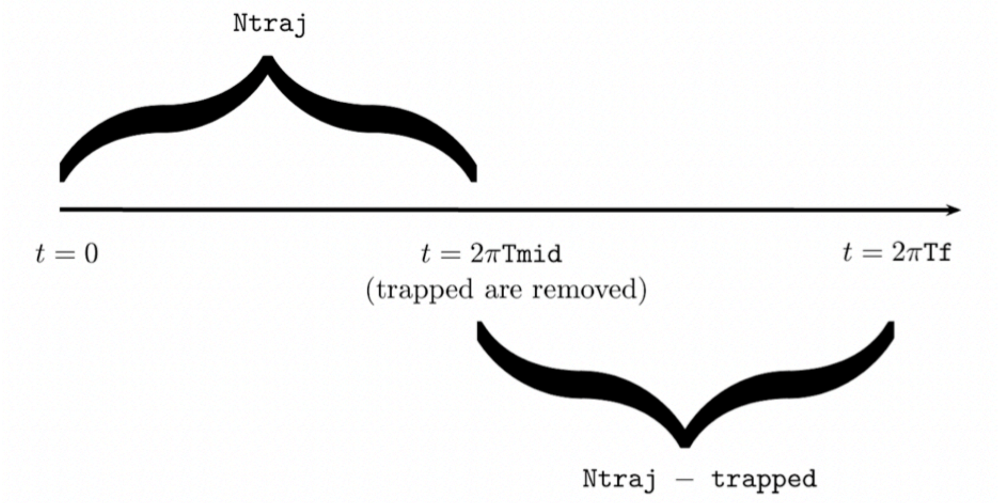


Figure 4.6: Visual representation of how integrations in two steps work. It is based on the fact that the percentage of trapped trajectories, for a reasonable value of `threshold` (e.g. 4, as it will be later justified), almost does not change after a sufficiently high number of cycles, which is assigned to `Tmid`.

Integration via `scipy.integrate.solve_ivp`

```python
import numpy as xp
from scipy.integrate import solve_ivp
if case.init == 'random':
        y0 = 2 * xp.pi * xp.random.rand(2 * case.Ntraj)
elif case.init == 'fixed':
        y_vec = xp.linspace(0, 2 * xp.pi, int(xp.sqrt(case.Ntraj)),
        ↪  endpoint=False)
        y_mat = xp.meshgrid(y_vec, y_vec)
        y0 = xp.concatenate((y_mat[0], y_mat[1]), axis=None)
        case.Ntraj = int(xp.sqrt(case.Ntraj))**2
t_eval = 2 * xp.pi * xp.arange(0, case.Tf + 1)
sol = solve.ivp (case.eqn_phi, (0, t_eval.max()), y0, t_eval=t_eval,
↪  max_step=case.TimeStep, atol=1, rtol=1)
x, y = xp.split(sol.y, 2)
```

Listing 7: Simplified version of a portion of `gc2d_modules.py` associated with the integration of the equations of guiding center dynamics. First, random or fixed initial conditions are generated and stored in `y0`. Then, the `Tf+1` instants of time associated with stroboscopic plots are defined. After that, `scipy.integrate.solve_ivp` acts on `eqn_phi` (see Listing 5) to obtain the temporal evolution of positions of guiding centers. The time step is kept fixed at `TimeStep` thanks to `atol = 1` and `rtol = 1`. Positions are stored in `sol.y` only in correspondence of instants included in `t_eval`. For better clarity, `sol.y` is split into two distinct components `x` and `y`, i.e X and Y).

The numerical integration is essentially performed thanks to the `scipy.integrate.solve_ivp` function (see Listing 7), where `ivp` stands for *initial value problem*. This solver, in default mode, is used to solve systems of differential equations through the explicit Runge-Kutta method of order 5. The use of this function inside `gc2d_modules.py` is associated with Listing 5, which is part of `gc2d.py`, because of `case.eqn_phi`. The file, through the definition of `t_eval`, is made in such a way that the positions of the guiding centers are recorded only at the end of each cycle to form a *trajectory*. This means that, if the equations are integrated with reference to `Tf` cycles, for example, then the data will be available at `Tf` $+ 1$ instants of time.
For purposes of clarity, Listing 7 shows simpler form of the implementation of

scipy.integrate.solve_ivp than it actually is inside gc2d_modules.py. The real version (see Listing 8) is longer because it takes into account the possibility to do the integration in one or two steps.

```python
import numpy as xp
from scipy.integrate import solve_ivp
if not case.TwoStepIntegration:
        sol = solve_ivp(case.eqn_phi, (0, t_eval.max()), y0, t_eval=t_eval, max_step=case.TimeStep, atol=1, rtol=1)
        x, y = xp.split(sol.y, 2)
        untrapped = compute_untrapped((x, y), thresh=case.threshold)
        x_un, y_un = x[untrapped, :], y[untrapped, :]
        x_tr, y_tr = x[xp.logical_not(untrapped), :], y[xp.logical_not(untrapped), :]
else:
        sol = solve_ivp(case.eqn_phi, (0, t_eval[case.Tmid]), y0, t_eval=t_eval[:case.Tmid+1], max_step=case.TimeStep, atol=1, rtol=1)
        x, y = xp.split(sol.y, 2)
        untrapped = compute_untrapped((x, y), thresh=case.threshold)
        x_un, y_un = x[untrapped, :], y[untrapped, :]
        x_tr, y_tr = x[xp.logical_not(untrapped), :], y[xp.logical_not(untrapped), :]
        print('\033[90m        Continuing with the integration of {} untrapped particles... \033[00m'.format(untrapped.sum()))
        y0 = xp.concatenate((x_un[:, -1], y_un[:, -1]), axis=None)
        sol = solve_ivp(case.eqn_phi, (t_eval[case.Tmid], t_eval.max()), y0, t_eval=t_eval[case.Tmid:], max_step=case.TimeStep, atol=1, rtol=1)
        x, y = xp.split(sol.y, 2)
        x_un = xp.concatenate((x_un, x[:, 1:]), axis=1)
        y_un = xp.concatenate((y_un, y[:, 1:]), axis=1)
```

Listing 8: Actual portion of `gc2d_modules.py` associated with the integration of the equations of guiding center dynamics. It takes into account the possibility to have an integration in one or two steps. The number of untrapped trajectories is determined after one step in both cases. If `TwoStepIntegration = False`, this allows Python to save computational time during the second step. See Listing 9 for the definition of `compute_untrapped`.

4.4.5 Computation of trapped particles

As already said, it is possible to distinguish between trapped and untrapped particles by analyzing their trajectories. It is important to note that plotted trajectories will be discrete and disconnected (some of them may appear continuous lines, depending on the level of zoom), always because here the term *trajectory* is not associated with positions recorded at every simulated instant of time.
However, there is not an unique way to do that. Indicatively, trapped particles have trajectories in the transverse plane which tend to remain close to the point where they were at $t = 0$. Guiding centers tend to draw closed lines of limited size (usually, they are called *orbits* and they are associated with *islands* of stability). One of the characteristics of orbits is that they are usually completed much earlier than `Tf` cycles (if `Tf` is sufficiently high to allow a good determination of the diffusive regime of the system), in the sense that guiding centers cyclically return to the areas they had already passed through multiple times.

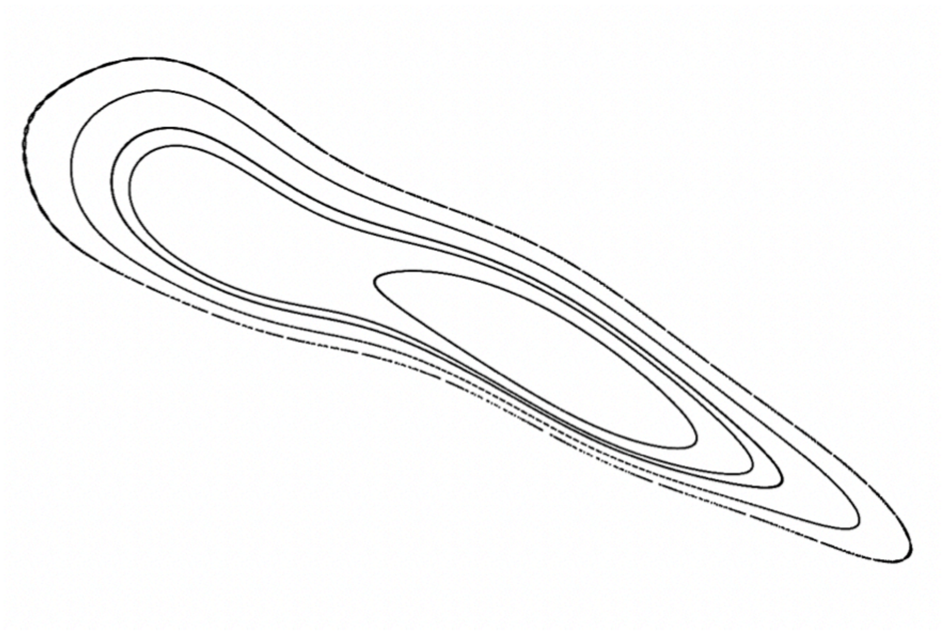


Figure 4.7: Example of regular trapped trajectories in the transverse plane.

Untrapped particles, on the other hand, as the name suggests, tend to move considerably away from their initial position. They could do that in a chaotic way or in a more regular way. The term *chaotic* is generally associated with a guiding center whose trajectory:

- Tends to fill a 2D region of phase space (if it is folded using the modulo function). In this case, a chaotic trajectory would fill the fundamental square.

- Is very sensitive to initial conditions.
- Tends to escape (even if this is not always true).

On the other hand, a guiding center has a *regular* evolution when its path is very clear and easily identifiable (even when it is considered together with other trajectories). Chaos and regularity are better recognizable when looking at modular or zoomed non-modular stroboscopic plots (see, for example, Figs. 4.5 and 4.20).

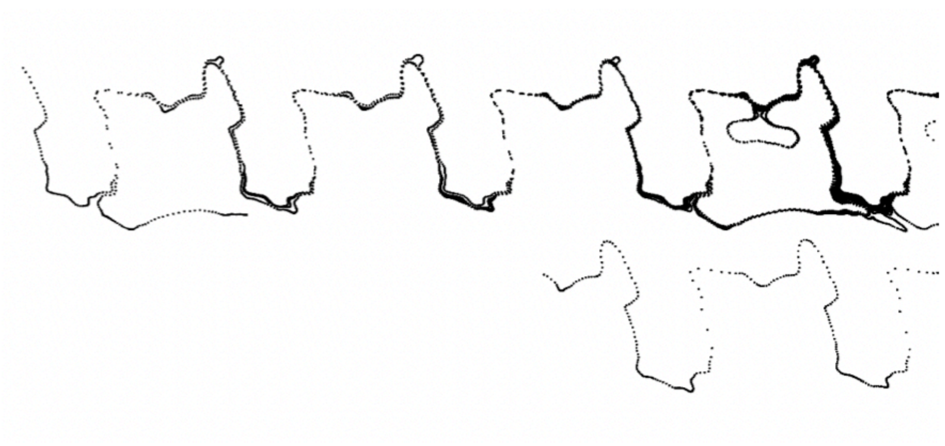


Figure 4.8: Example of untrapped trajectories in the transverse plane. It can be seen that trajectories are not continuous, but are formed by a discrete set of points. In the regions where lines seem continuous, single dots would be obtained by zooming in by a sufficient amount.

The need to distinguish between these two types of trajectories derives from a series of factors:

- Trapped particles do not diffuse (neither in the classical way nor in an anomalous way) and therefore are not relevant for calculating coefficients associated with diffusion.
- Continuing to integrate the equations for trapped particles beyond necessary is only a useless computational effort that increases computation time: an island, as already mentioned, is completed after a limited number of cycles, much less than `Tf` , if $\texttt{Tf} \gg 1$.

In theory, the correct number of cycles to dedicate to the integration of a trapped trajectory is the one that allows one to understand if the trajectory actually belongs to a trapped particle.
There may be some cases in which a particle that appears trapped after a certain number

of cycles, at some point abruptly changes behavior. However, if one or more particles of this kind are categorized incorrectly, it is not a problem for the purposes of the simulations: simulating a very high number of trajectories allows one to properly counteract these scenarios. In fact, the purpose of the simulations is not to count exactly the number of trapped particles, but to understand, or at least record, the general behavior of the system as the main parameters `A`, `rho` and `eta` vary.
However, the main reason for which I have decided to introduce `TwoStepIntegration` is essentially computational in nature and serves to save time. In fact, it should not be forgotten that, in general, simulating the individual trajectories still requires a considerable effort.
The idea of increasing the number of integration steps beyond 2 was considered, but it was not then implemented, since with two steps already the gain is considerable and the script remain quite easy to write and read. However, the simulations can also be conducted just by keeping a single step for the integration. Moreover, this is also useful as a tool for checking and validating the results that can be obtained with two steps. The method that is used to effectively find the trapped trajectories is:

1. For each trajectory, the maximum and minimum values of their transverse coordinates are identified. A total of 4`Ntraj` coordinates are collected among 4`Ntraj(Tmid+1)` or 4`Ntraj(Tf+1)` total values, depending on the number of integration steps (one or two, respectively).

2. Ideally, these coordinates allow one to construct `Ntraj` rectangles, each containing the entire associated trajectory.

3. The length of the diagonal of each of these rectangles, which can be determined using the simple Pythagorean theorem, is compared with `threshold`. Any trajectory whose diagonal is less than or equal to `threshold`, is considered to be trapped. The others are consequently untrapped.

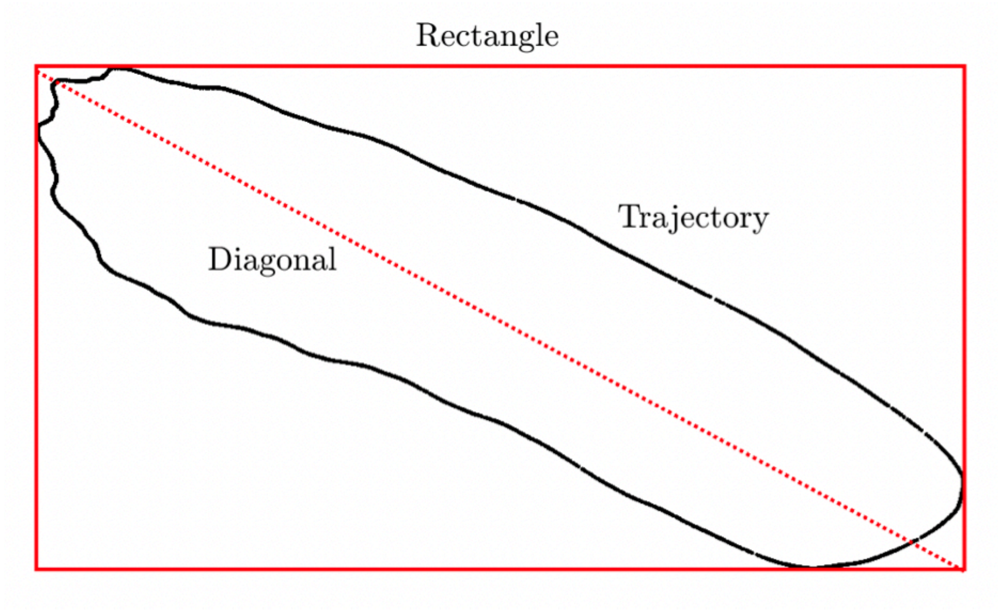


Figure 4.9: Example that shows how the method for detecting untrapped trajectories work. First, a rectangle is ideally drawn just outside the trajectory, then the length of its diagonal is compared with `threshold`. In Python, *drawing a rectangle* translates to finding maxima and minima of a trajectory in both directions of the transverse plane.

```python
import numpy as xp
untrapped = compute_untrapped((x, y), thresh=case.threshold)
def compute_untrapped(x, thresh=0, axis=1, output=[True, False]):
        vec = xp.sqrt(xp.sum([xel .ptp (axis=axis)**2 for xel in x], axis=0))
        ↪  > thresh
        return xp.where (vec==True, *output)
```

Listing 9: In order to determine the number of untrapped trajectories after a certain number of steps, the `compute_untrapped` function is used. First, thanks to `.ptp`, which stands for *peak to peak*, distances between maxima and minima coordinates along both axes are found for each trajectory (this corresponds to individuating a rectangle for each trajectory). Then, the sum of the square of those distances is evaluated for each trajectory (this corresponds to individuating a diagonal for each rectangle). After that, the number of untrapped trajectories is determined with the help of `numpy.where`, which compares each diagonal with a threshold value (which in the script corresponds to `threshold`).

The choice of using the diagonal, although not representative of the true max. distance spaced by the particle, is an extremely fast method and a good approximation. The diagonal suits well almost any type of trajectory and, if it is paired with a wise selection of `threshold`, it allows one to get to a good level of reliability.

4.4.6 Curve fitting to account for anomalous diffusion

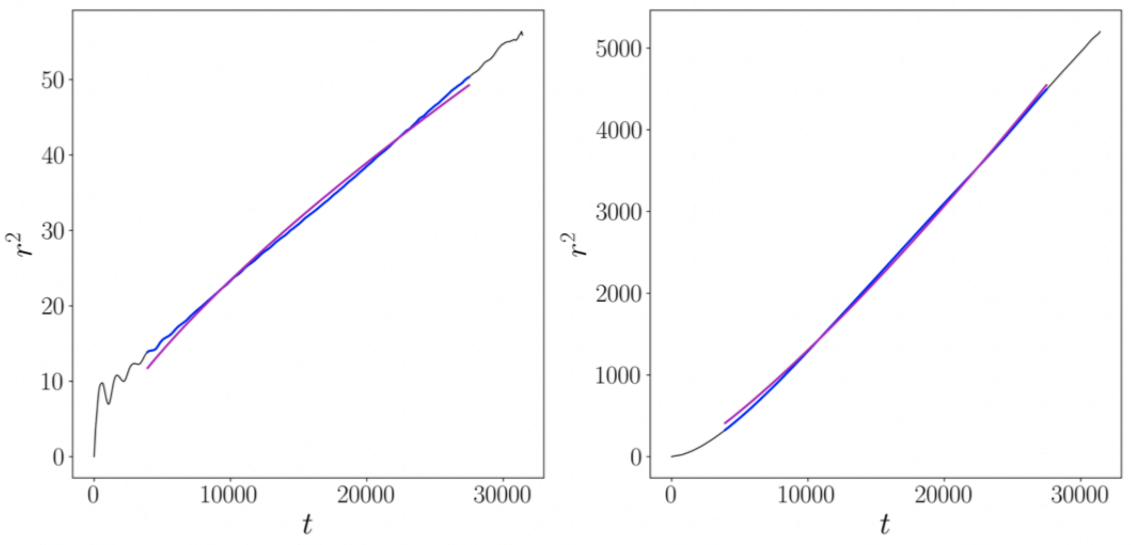


Figure 4.10: Plots of the evolution of mean square displacements (in blue and black) and their curve-fittings (in magenta) obtained with Python. Both plots are associated with `Ntraj = 1024`, `Tf = 5000`, `TimeStep = 0.05`, `A = 0.700`, `rho = 0.115` and initial random conditions. On the left, the system has `eta = -0.100` and is sub-diffusive, while on the right it has `eta = 0.100` and is super-diffusive (see subsection 5.3.2). Both curve fittings have $R^2 \geq 0.99$. Initial and final parts of r^2 (in black) and are not considered for the curve fittings, since they can have anomalies, as it is the case for the plot on the left.

Depending on the values of (`A`, `rho`, `eta`), the system can give rise to anomalous diffusion. For this reason, it is not recommended to have a part of the file dedicated to determining a diffusion coefficient based on the mean square displacement $r^2(t)$ of the trajectories (see Appendix C). Instead, it makes more sense to proceed with a power-law curve fitting of the type $(at)^b$. This is a more general approach which:

- Can detect the type of diffusion regime (anomalous or normal) through the value of b.
- In the case of b being close to 1, allows one to know the diffusion coefficient of the system, which is practically equal to a.
- Allows one to compare the behavior of different systems through b and, in smaller measure, also through a.

The curve fitting is obtained in this way:

1. $r^2(t)$ is calculated as a vector `r2` of `Tf` terms by implementation of Eq. C.4, with only the untrapped trajectories giving a contribution. Each term of `r2` is build as described in Appendix C using the positions of guiding centers at $t = 0$ and the end of each cycle. The instant of time corresponding to the generic k-th element of `r2` is equal to $2\pi k$, with $k = 0, ..., \texttt{Tf} - 1$.

2. The beginning and the tail of `r2` are cut to eliminate possible noises or anomalous behaviors that can typically appear in these regions (see, for example, the left plot in Fig. 4.10).

3. The curve fitting is carried out for the remaining part of using a power-law model `(a*t)**b` implemented thanks to the `scipy.optimize.curvefit` function (see section C.2 of Appendix C).

4. The value of the coefficient of determination R^2, associated with the fitting, is calculated via the
`sklearn.metrics.r2score` function (see Eq. C.9 in Appendix C). The result is stored in the variable `R2`.

In the analysis of the results, the value of `R2` should provide an indication of the goodness of the fit. Acceptable values are typically those greater than or equal to 0.990. Using a power-law curve fitting with two variable parameters almost never causes `R2` to be too low. The curve fitting is the specific part of the file that is activated if `Method = 'diffusion'`. Furthermore, by default, an output file in `.txt` format is produced with this method. It contains a table with the following data gathered from the various sub-simulations associated with a single run: `A`, `rho`, `eta`, `trapped`, `a`, `b` and `R2`, where `trapped` is the percentage of trapped particles with respect to the true value of `Ntraj`.

```python
import numpy as xp
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
r2 = xp.zeros(case.Tf)
for t in range(case.Tf):
        r2[t] = ((x_un[:, t:] - x_un[:, :-t if t else None])**2 + (y_un[:,
        ↪  t:] - y_un[:, :-t if t else None])**2).mean()
t_win, r2_win = t_eval[case.Tf//8:7*case.Tf//8], r2[case.Tf//8:7*case.Tf//8]
func_fit = lambda t, a, b: (a * t)**b
popt, pcov = curve_fit(func_fit, t_win, r2_win, bounds=((0, 0.25),
↪  (xp.inf, 3)))
r2_fit = func_fit(t_win, *popt)
R2 = r2_score(r2_win, r2_fit)
trapped = xp.logical_not(untrapped).sum()
```

Listing 10: Portion of `gc2d_modules.py` associated with curve fittings. First, the mean square displacement of all untrapped trajectories is evaluated at `Tf` instants of time (from $t = 0$ to $t = 2\pi(\texttt{Tf} - 1)$) by implementing Eq. C.4. Then, time and r^2 are cut to remove initial and final portions. With them, a power-law curve fitting is performed thanks to `scipy.optimize.curve_fit`. After that, the associated coefficient of determination is obtained with `sklearn.metrics.r2_score` and the number of trapped particles is determined.

4.5 Description of `Reader_diffusion.m`

The first of the two MATLAB files is the self-explanatory `Reader_diffusion.m`. It specifically deals with plotting the data collected inside the `.txt` file produced by Python when `Method = 'diffusion'`. The convenience of using a file which is not the one that produced the results is the ability to work more easily on it, other than being able to analyze the results more quickly. In fact, it can be written using a large number of `for` loops, which are intuitive to write, without compromising its speed, since it is simply a file for reading and plotting data. The usage of `for` loops is very limited in the Python files because they tend to be very slow. Moreover, it can be edited and changed live while the same set of Python output data is being used as a sample.
What `Reader_poincare.m` does is essentially producing 2D or 3D plots of `trapped`, `a`, `b` and/or `R2` as `A`, `rho` and/or `eta` vary.

In general, the file includes some editable lines that allow one to completely disable some of the graphs and to select secondary parameters such as colors, text size, type of graphs (e.g. `surf`, `heatmap`, `plot` and `contourf`) and so on. It is also immediate to change, if needed in particular occasions, some lines in the file to realize more specific plots.
The most convoluted part of this file is the one that puts the quantities in the correct order and form to be plotted, regardless of the of the graphs that are used. Tables are usually very readable with MATLAB, but the structure of the `.txt` file forces some additional efforts in MATLAB.

4.6 Description of `Reader_poincare.m`

The second MATLAB file is `Reader_poincare.m` and it takes care of the `.mat` file produced in Python when `Method = 'poincare'` and `SaveData = True`. Actually, it can also analyze the results contained in the `.mat` file associated with `Method = 'diffusion'` and `SaveData = True`. By reading those files, it can create a series of plots:

- Non-modular stroboscopic plots with up to three colors: as in Python, the trajectories are plotted in the transverse plane at $t = 0$ and at the end of each cycle. The novelty here is that from one to three colors can be used to highlight trajectories. Basically, each color can be associated with a different threshold value, allowing to put the focus on different kinds of trajectories, e.g. trapped, untrapped and superdiffusive. Of course, the code can be easily adapted to be able to display even more than three colors, however it must be kept in mind that too many colors can lower the visual efficiency of Poincaré sections. An advantageous aspect in using MATLAB is that if the Python data are handled in a certain way, it is possible to use the figure environment to highlight the evolution, clockwise or counterclockwise, of the trajectories using the computer cursor. Graphs like the one just described can be made even if `modulo = True` in Pyton. In fact, the data are saved in the `.mat` file before it is scaled inside the fundamental square. Furthermore, the Poincaré sections can also be turned into animations through small additions to the file.
- Modular stroboscopic plots with up to three colors: as in Python, it is the scaled version to the fundamental square of the Poincaré section of point (1).
- Untrapped trajectories as a function of a threshold value: this graph is particularly useful to provide a feedback to properly set `threshold` in Python and also to compare particles from another point of view, in some ways similar to that of the curve fitting. In particular, this plot shows on the horizontal axis a dense range of possible values of the Python `threshold` parameter, while on the vertical axis it shows the number of trajectories whose diagonal values are greater than or equal to a certain value. The values of diagonals are recalculated in MATLAB in the

same way as they are calculated in Python. The comparison of diagonals with each element of the `threshold` vector is done by means of `for` loops. In other words, what is done is a multiple distinction between *numerically* (and not necessarily *true*, where the *true* is associated with the threshold value of 4) trapped and untrapped trajectories, each time based on a different threshold value. The information that can be obtained from this graph is discussed in sections 4.8.4 and 4.8.5.

- Curve fitting for the mean square displacement (only if the `.mat` file comes from a `diffusion`-type simulation): exactly as in Python, the curve fitting is recreated starting from the trend of the truncated mean square displacement (also a non-truncated version or a differently cut version can be used). This allows one to compare curve fittings performed in two different environments and gave the idea to implement $r^2(t)$ plots in Python too.

4.7 Effects of A, ρ and η on the potentials

The three parameters A, ρ and η that appear in the theory as well as in the Python files in the form of `A`, `rho` and `eta`, can be characterized by the effect that they have on the potentials. There are different ways that can be used to properly show their influences, which are discussed in the following subsections. Other important quantities for the characterization of the potential are the phases φ_{nm}, which in the files appear as elements of the matrix `phases`. Each term of the summation through which the potential is defined is associated with a different phase chosen at random between 0 and 2π. Although it may seem like a secondary parameter, the phase is crucial for a correct modelling of the potential: if all the phases were equal (for example, all null), then the potential would not be turbulent because it would appear too regular and predictable in its evolution. Having a different phase for each term of the summation is precisely one of the main techniques for modelling a turbulent behavior. In this way, the trend over time of the potential is not symmetric and is characterized by the formation, the movement and the periodic disappearance of eddies in several non-symmetric areas of the fundamental square. The effect of phases is shown in Fig. 4.11.

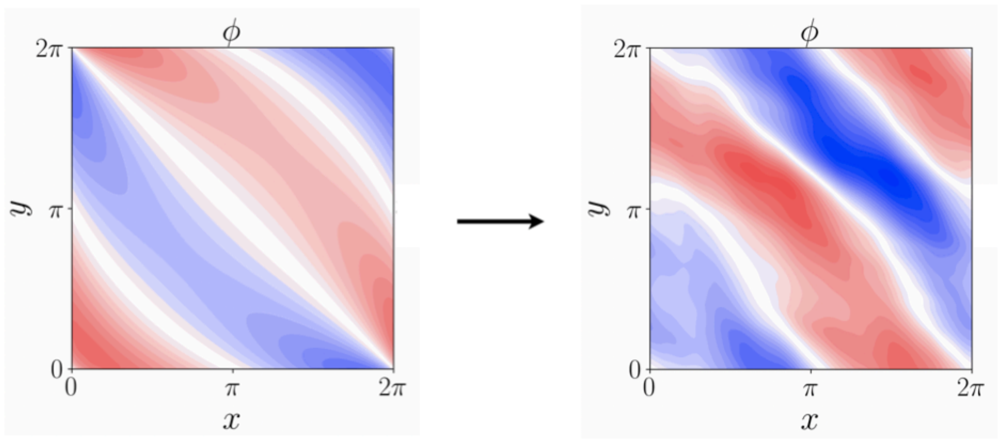


Figure 4.11: Comparison between potentials ϕ, obtained in Python with `A = 1.000`, at $t = 0$. On the left, all the phases are null, while on the right they are random. Random phases allow one to model turbulence.

4.7.1 Effect of A

Of the various parameters, A is certainly the easiest to predict: being the amplitude of the potential, the higher A is, the more the sinusoids characterizing the potential will be marked. By plotting the predominant term of the potential ϕ, evaluated at $t = 0$ and with $\varphi_{11} = 0$, it is possible to clearly evaluate the influence of A. A graphical representation inside the fundamental square is given by Fig. 4.12.

$$\phi^{(1,1)}(x, y, t = 0) = \Im[2^{-1.5} A \mathrm{e}^{\mathrm{j}(x+y)}] \tag{4.6}$$

Some results are shown in Fig. 4.12 Equivalently, the same graphs can be obtained using the predominant term of ψ with $\rho = \eta = 0$:

$$\psi^{(1,1)}(X, Y, t = 0) = \Im[2^{-1.5} A J_0(0) \mathrm{e}^{\mathrm{j}(X+Y)}] = \Im[2^{-1.5} A \mathrm{e}^{\mathrm{j}(X+Y)}] \tag{4.7}$$

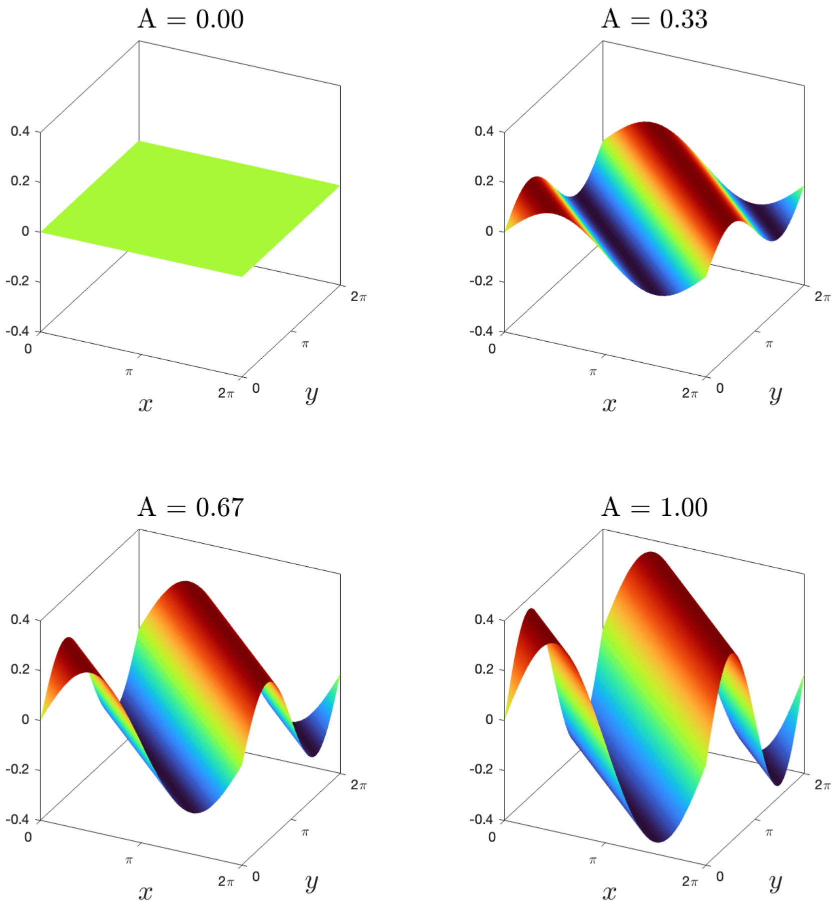


Figure 4.12: Changes in the shape of $\psi^{(1,1)}$ inside the fundamental square as A varies. The actual amplitude increases with A, but it is not exactly equal to A because of how ϕ is defined using modes. The four subplots have been obtained with MATLAB.

4.7.2 Effect of ρ

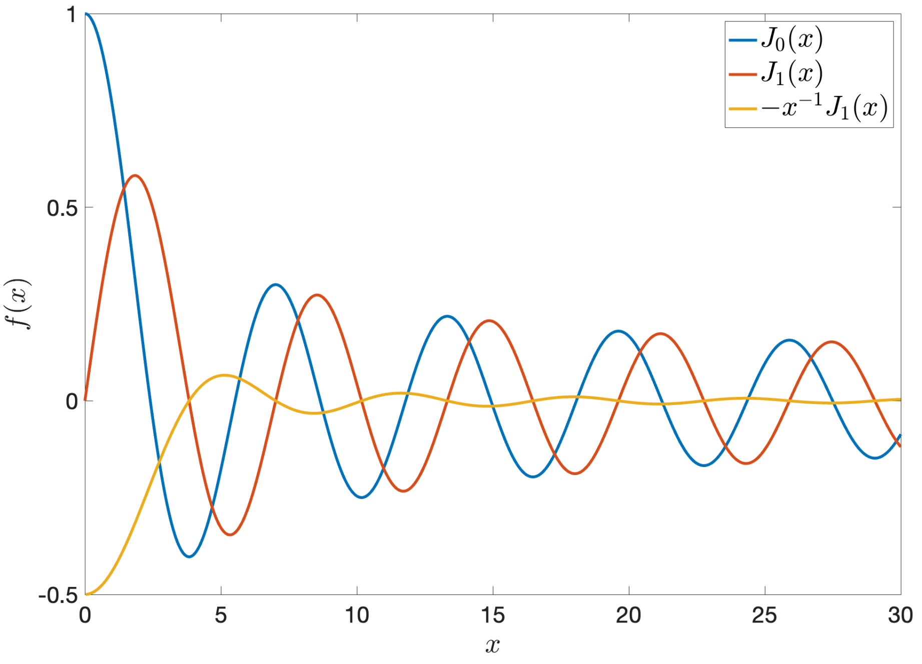


Figure 4.13: Bessel functions of the first kind of order 0 and 1, plus a curve for $-x^1 J_1(x)$ that is there to show the general behavior of $-s_{nm}J_1(\rho s_{nm})/\rho$.

The effect of the Larmor radius on ψ is less straightforward to predict, since it seems to have no direct relationship with potentials from a physical perspective. However, a hint of its possible influence can be obtained if it is recalled that gyroaveraging causes Bessel functions to appear (see, for example, Eq. 3.134). Given that $J_0(s_{nm}\rho)$, for a fixed value of s_{nm}, oscillates and decays as ρ becomes bigger, it is reasonable to assume that ρ has the effect of watering down the sinusoids when moving from ϕ to the first order of ψ (with A fixed). In fact, this is what can be seen in the left plots of Fig. 4.14, where again the effect of the phases is neglected. Positive and negative peaks of ϕ become much weaker in ψ under the effect of the Bessel function J_0.
The influence of the Larmor radius on the second term of the guiding center potential is more difficult to predict, given that η has to be taken into account too and that ρ appears inside the term $-s_{nm}J_1(s_{nm}\rho)/\rho$. By looking at the bottom-right plot of Fig. 4.14, which has been made with `A = 1.00`, `rho = 1.00` and `eta = 1.00`, shows that the Larmor radius generates small peaks at the four corners of the fundamental square (at $t = 0$). Their limited range means that, when the two orders are combined to form

ψ, the end result is much more similar to the first order term than the second (see the upper-right plot of Fig. 4.14).

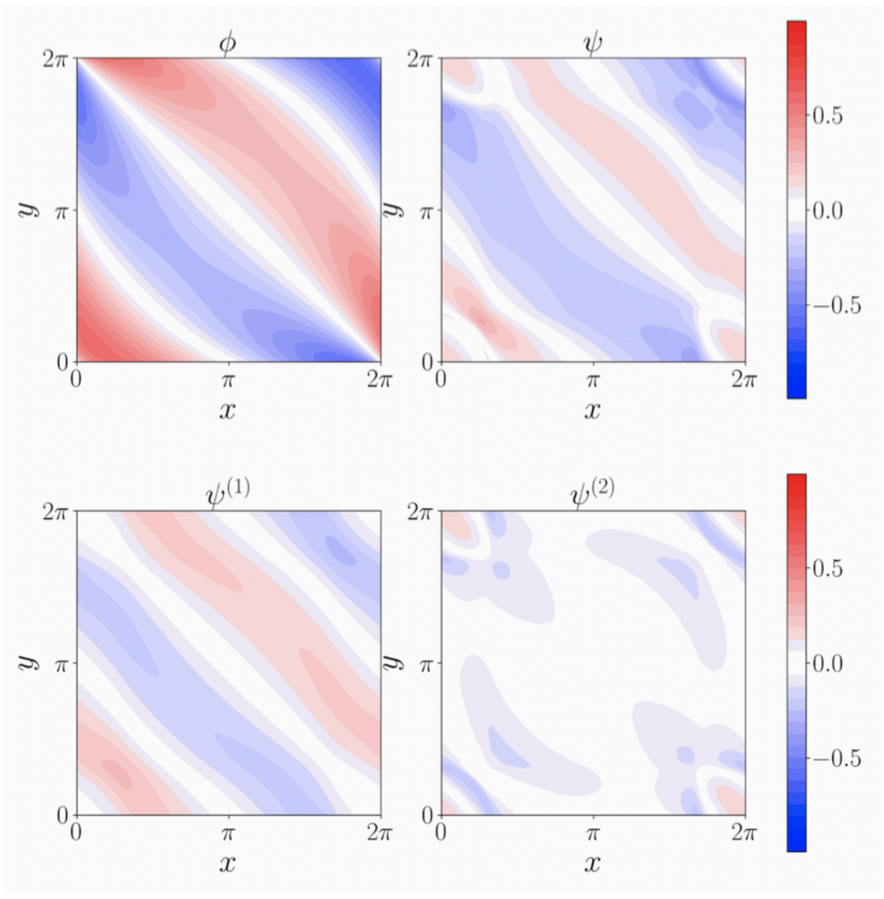


Figure 4.14: Potentials, at $t = 0$, obtained in Python with `A = 1.000`, `rho = 1.000`, `eta = 1.000` and null phases.

4.7.3 Effect of η

It is obvious that η has no effect on the first order of ψ. However, it is interesting, by exaggerating its possible values, what it can produce on ψ by acting inside its second order. The analysis in this case is a little bit easier if compared to the previous one, since ρ (other than the phases) can be set to zero. Moreover, the investigation is also more interesting because η can be increased in both positive and negative directions. By looking at Figs. 4.15 and 4.16, a nice behavior emerges:

- Negative values of η tend to amplify the positive areas of the potential. In other words, larger portions of the fundamental square are occupied by positive values of ψ than of ψ. This is caused by a second order of ψ which is entirely positive or close to being null, with predominant peaks at the four corners of the fundamental square.

- Positive values of η produce the exact opposite, with negative values of the potential becoming more predominant than the positive ones, which see their space reduced. In ϕ, negative and positive values at $t = 0$ occupy the exact same fraction of the fundamental square.

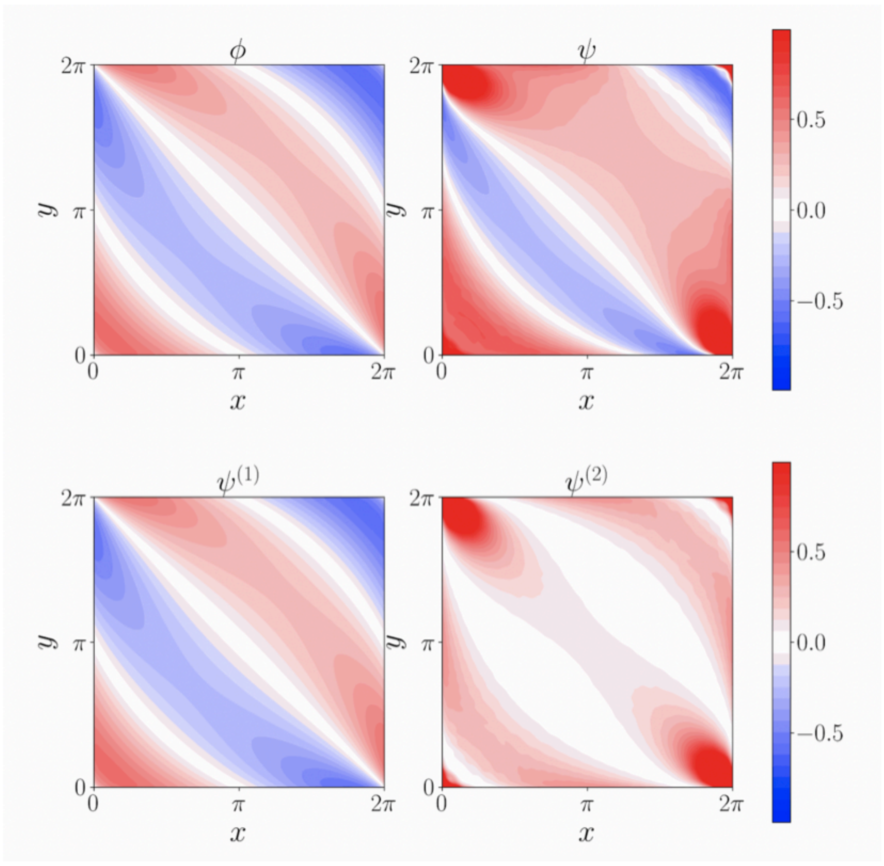


Figure 4.15: Potentials, at $t = 0$, obtained in Python with `A = 1.000`, `rho = 0.000`, `eta = -1.000` and null phases.

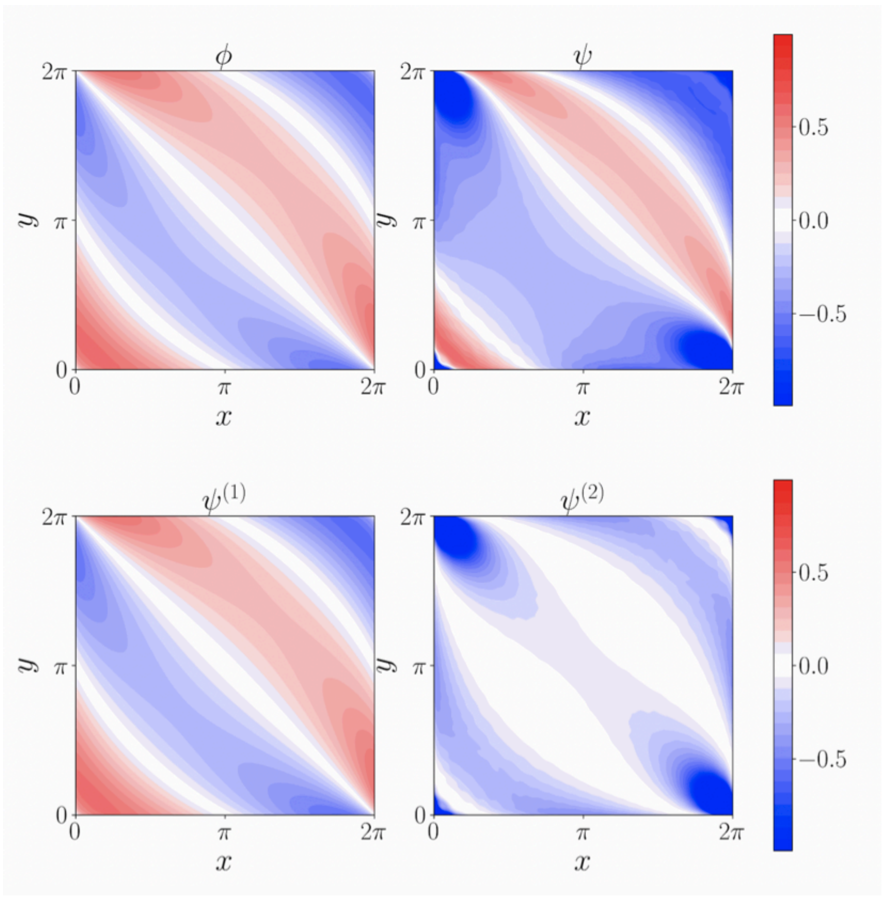


Figure 4.16: Potentials, at $t = 0$, obtained in Python with `A = 1.000`, `rho = 0.000`, `eta = 1.000` and null phases.

4.8 Comparisons between features of different versions of the Python scripts and choices of values and ranges for the parameters

4.8.1 Power-law curve fitting instead of diffusion coefficients

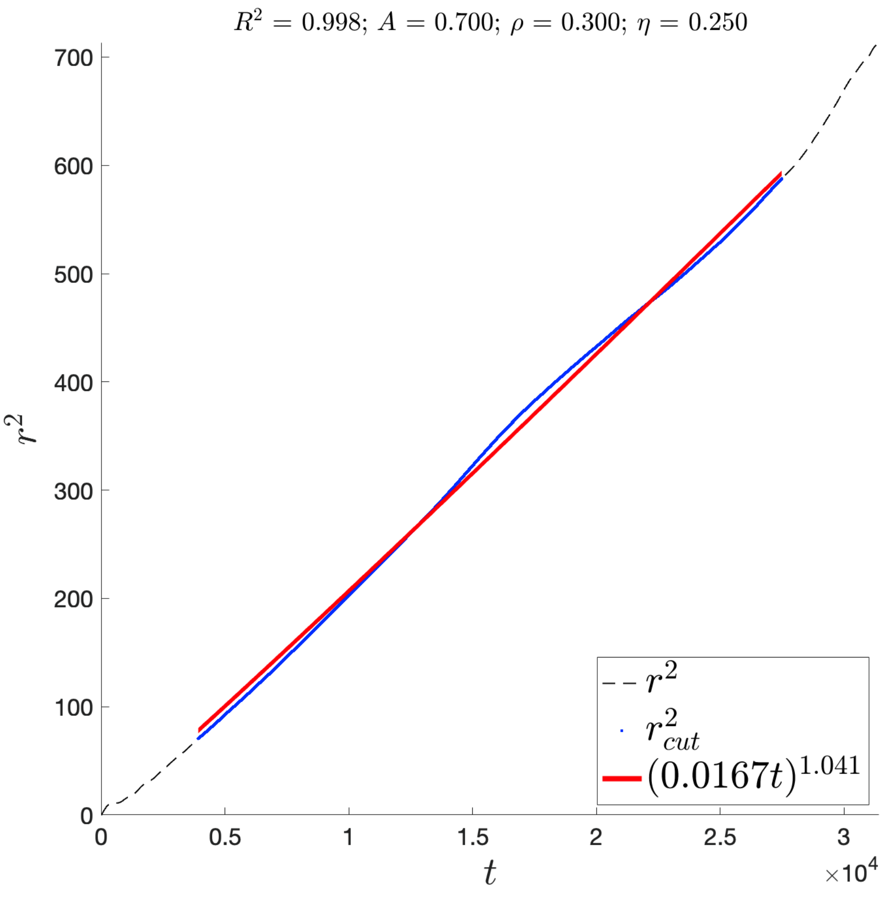


Figure 4.17: Evolution of the mean square displacement (in blue and black) obtained in MATLAB with `Ntraj = 1024`, `Tf = 5000` and `TimeStep = 0.05`. The dashed portions of r^2 are not used to perform the curve fitting (in red). For this particular graph, both trapped and untrapped particles have been used (`threshold = 0`) because it is taken from a `.mat` file associated with a simulation in `poincare` mode. The system is diffusive, having $b \simeq 1$ (the same diffusive regime would have been detected even in the case of `threshold = 4`, obviously with slightly different values for R^2, a and b).

Initially, the main purpose of this work was to investigate the effect of the second term of ψ with respect to using the first term alone.
Within the original `gc2d_modules.py` file, in diffusion mode, the dynamics of guiding centers were used to determine the diffusion coefficient of the system. As indicated more extensively in Appendix C, this means that the system is always assumed to have a linear trend of the mean square displacement with respect to time, with the diffusion coefficient being equal to the slope of the associated curve. However, this created anomalies in some of the output values. In particular, some `R2` values associated with the diffusion coefficients were very far from being acceptable, i.e. below or well below 0.99. This suggested that it is not true that all combinations of `A`, `rho` and `eta` give rise to a linear curve fitting.
Numerous simulations were then carried out with higher values of `Ntraj` and `Tf`, but no improvements on `R2` were made. The low values of `R2` were therefore mostly independent of those parameters, but they were linked to something different: the physics of the systems and the way Python was told to interpret it.
For this reason, a considerable part of this work was dedicated to deciding how to handle the possibility of anomalous diffusion (or no diffusion at all) for some combinations of `A`, `rho` and `eta`. The solution to this problem is the one presented in subsection 4.4.6, which consists in having replaced the calculation of diffusion coefficients with power-law curve fittings, together with eliminating some ranges of values for the three parameters (see subsection 4.8.7). Curve fitting is a more general approach because it does not impose any assumption on the type of diffusive regime, other than containing the evaluation of diffusion coefficients as a particular case.
The original diffusion part of the file looked like in Listing 11, while the new one has already been displayed in Listing 10. What was kept from the original version is the truncation of `r2`.

```python
import numpy as xp
from scipy.stats import linregress
r2 = xp.zeros(case.Tf)
for t in range(case.Tf):
        r2[t] = (xp.abs(sol.y[:, t:] - sol.y[:, :case.Tf-t]) ** 2).sum() /
         ↪  (case.Ntraj * (case.Tf - t))
diff_data = linregress(t_eval[case.Tf//8:7*case.Tf//8],
 ↪  r2[case.Tf//8:7*case.Tf//8])
max_y = xp.sqrt((xp.abs(sol.y[:case.Ntraj, :] - sol.y[:case.Ntraj,
 ↪  0].reshape(case.Ntraj,1)) ** 2 + xp.abs(sol.y[case.Ntraj:, :] -
 ↪  sol.y[case.Ntraj:, 0].reshape(case.Ntraj,1)) ** 2).max(axis=1))
trapped = (max_y <= 3.0 * xp.pi).sum()
```

Listing 11: Portion of the original gc2d_modules.py that is associated with the determination of diffusion coefficients. This part was later changed into Listing 10. The formula for r2[t] is only visually different from the one in the latest version, but it in the end they are the same. Here, instead of a power-law curve fitting, a linear regression is performed with scipy.stats.linregress. From diff_data it was then possible to obtain the diffusion coefficient of the system as the slope of the linear curve fitting, other than the coefficient of determination R^2. The number of trapped particles is evaluated using the old version of compute_untrapped (see Listing 9), i.e. the one that relies on initial positions and not on diagonals and rectangles. In the original script, threshold was not a parameter and was fixed at 3π.

4.8.2 Methods to find trapped and untrapped trajectories

In the dictionary, threshold is used to distinguish between trapped and untrapped guiding centers. In the original version of the file, threshold had a suggested value of 3π and was used as it is described here:

1. For each trajectory, the max. distance from the coordinate that the guiding center had at the beginning of the simulation was determined.

2. For each trajectory, the max. distance was compared with threshold. If the max. distance was bigger, then the associated particle was considered to be untrapped.

This approach presents two problems (even though not of significant relevance): the value of threshold and the criterion used to the determine max. distances.

Leaving aside for now the value of `threshold`, which will be discussed in subsection 4.8.4, the other aspect could have been improved because of some considerations:

- The potential is periodic in time: the initial positions of guiding centers do not have more special characteristics than other positions at other instants of time. Therefore, it makes no particular sense to give initials positions a privileged role in the formula for the max. distance.
- Even under the assumption that that initial positions have, for some reason, the right to be considered more important than others, they would still not allow one to always accurately evaluate the extension of trajectories in the transverse plane.

For these reasons, the first modification that was tested involved measuring the max. distance travelled by each guiding center by extending the procedure just presented in this section. In this case, the max. distance was no longer evaluated by comparing the starting point with any other point of the associated trajectory, but it was instead calculated by comparing every possible pair of points within each trajectory, giving rise to much more accurate evaluations. However, it is evident that this method is much slower than the original, even without considering that it requires the use of `for` loops. This turned out to be a bad solution, because the gain in accuracy was much lower than the increase of computation time.
Therefore, it was decided to adopt another solution, which is the one that has already been presented in section 4.4. Not only this technique is very fast, being associated with the computation of maxima and minima within vectors, but it is also able to be more accurate, in general, than the original solution. A visual comparison of these three methods is available in Fig. 4.18.

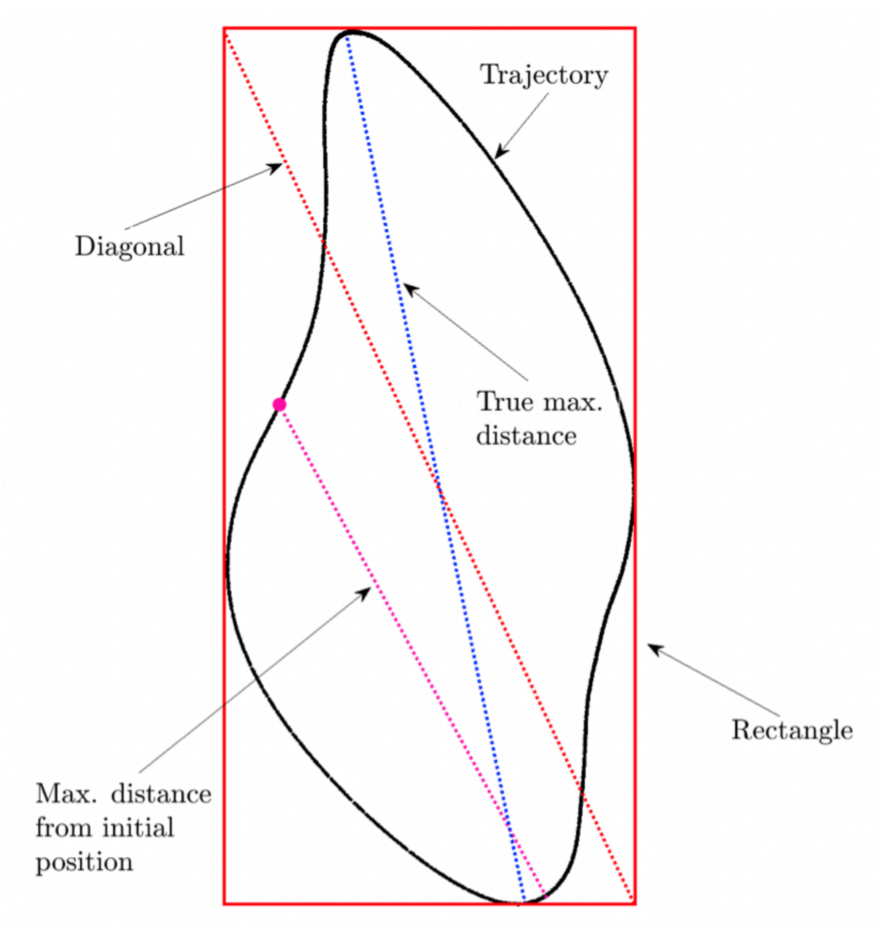


Figure 4.18: Graphical comparison of the methods used and tested to find max. distances covered by guiding centers. The original method, that relies on initial positions (the magenta dot in this case), can be problematic when trajectories are far from being circular or are open. The method that relies on diagonals is much more consistent, even if it slightly overestimates true max. distances.

Interestingly, the worst case scenario for the last version of the max. distance is also the best condition for the original one. Ideally, if the trajectory that is being studied happens to be a perfect and continuous circumference of radius r, then:

- The original method would almost exactly give the max. distance, i.e. the diameter, regardless of the initial position of the guiding center. The same would be true for the *slow* method, obviously.
- The latest method would give a max. distance that is $2r\sqrt{2}$, which is $\sqrt{2} \simeq 1.41$ times bigger than the actual result. This is the maximum error that can be made

with this method, and it can be more directly explained with Fig. 4.19. Also, this method can not give a result that is less than the actual max. distance.

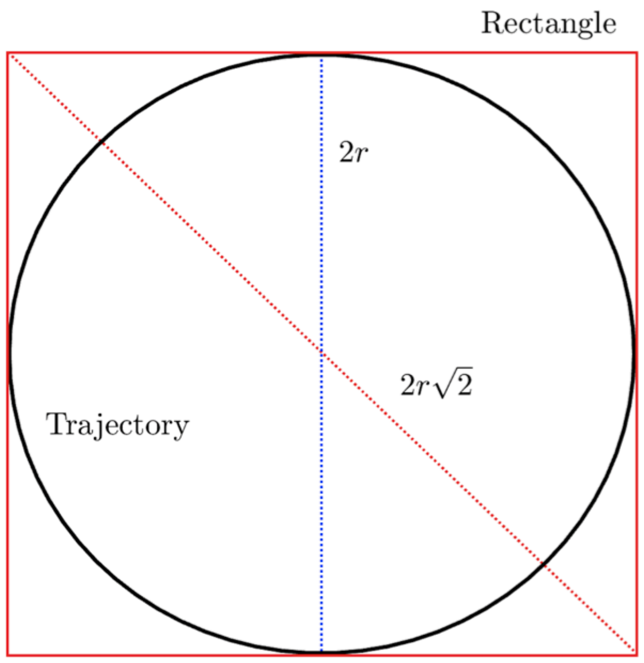


Figure 4.19: In the ideal case of a perfectly circular trajectory, the method that relies on diagonals to find untrapped trajectories has the highest discrepancy with the true max. distance, that is the diagonal of the circle. The same case would result in the minimum error for the method that relies on initial positions.

Fortunately, the biggest trapped trajectories are often far from being perfect circumferences and, even if this was not the case, the error would still be very limited. The randomness of initial positions makes errors in the original method much more unpredictable in the relevant cases. A striking example of how elongated trapped trajectories can actually be is shown in Fig. 4.20.

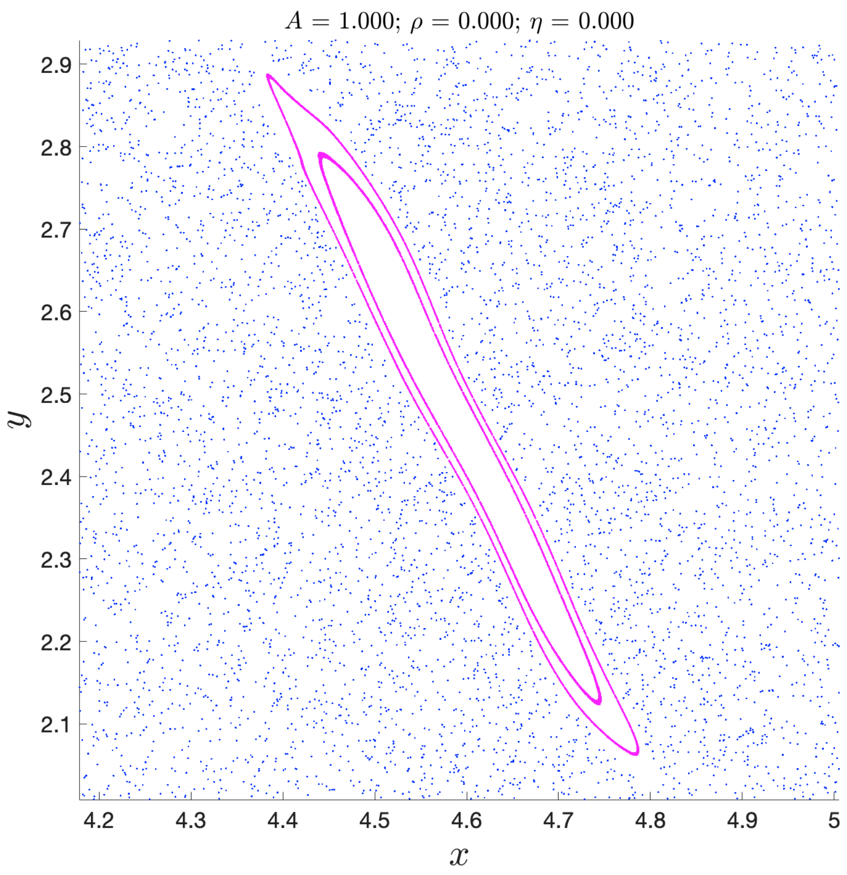


Figure 4.20: Zoomed-in version of a non-modular stroboscopic plot in which two elongated and regular trapped trajectory can be clearly seen in magenta. Around them, there is a blue *chaotic sea* of untrapped trajectories. The plot has been realized with MATLAB.

Description of the error in the original `gc2d_modules.py`

The only error that was found within the three original Python files had to do with calculation of the max. distance. Quite trivially, a square root was missing, (see line 7 in Listing 11, in which `numpy.sqrt` was added only after a while). This error caused the number of untrapped particles to be higher. Numerous simulations were launched before noticing the error, but this did not compromise some of the information obtained from them. In fact, to consider some trapped particles as untrapped does not alter the diffusive regime, since trapped particles do not diffuse. It is clear that this specific error would have been a huge problem if, instead, it caused a decrease in the number of untrapped particles.

4.8.3 Multicolored stroboscopic plots instead of monocolored ones

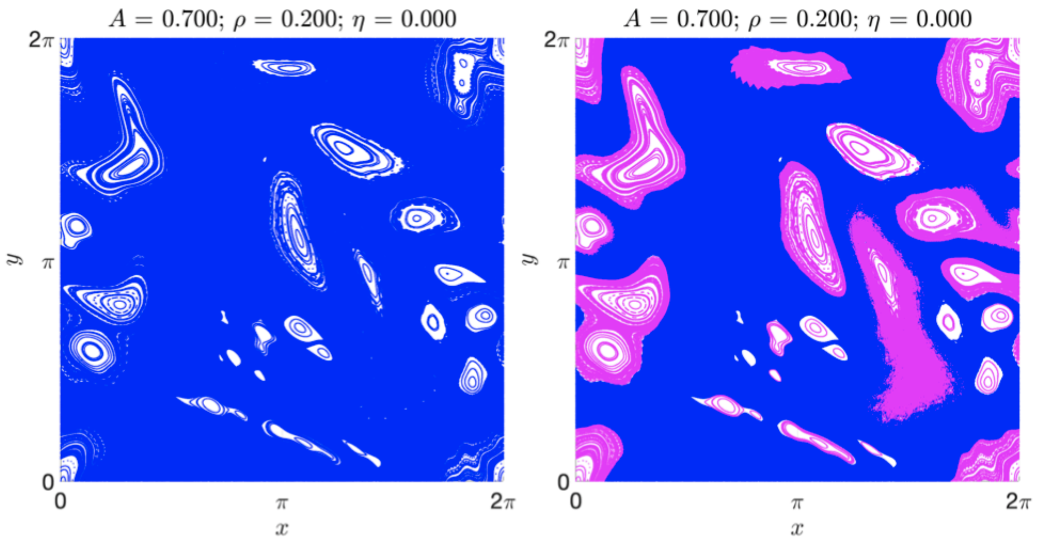


Figure 4.21: Comparison between two different modular stroboscopic plots of the same system On the left, the graph is monochromatic (as it is usually done with Poincaré sections), while on the right two different colors are used to distinguish trapped trajectories (in magenta) from untrapped ones (in blue) using a threshold value of 4. These plots have been made in MATLAB using the outputs of simulations with `Ntraj = 1024`, `Tf = 5000` and `TimeStep = 0.05`.

During the analysis with MATLAB for the determination of `threshold`, it spontaneously appeared the need to highlight with a different color the trajectories associated with trapped guiding centers. Thus, in MATLAB, the lines of code dedicated to the creation of stroboscopic plots were modified to be able to color the trajectories in two different ways according to an arbitrary threshold value (not necessarily equal to `threshold`). Adding a color makes it possible to read Poincaré sections much better and also to highlight aspects that would otherwise be more complicated to identify, such as the intermediate cases between the well-marked trapped guiding centers and the clearly untrapped ones. This possibility was then also implemented in Python, where different colors are assigned this time on the basis of `threshold`, which is fixed for a given simulation (in this case, it is clear that using a different file to process the outputs is more flexible).
In this context, it is possible to use `threshold` (or, more conveniently, its equivalent in MATLAB) not only as a discriminating value between trapped and untrapped guiding

centers, but also, for example, as a detector for particular types of guiding centers: those with very tiny closed trajectories, those with very high values of the max. distance, and so on.
Focusing on those systems that exhibit anomalous diffusion through the value of `b`, being able to highlight super-diffusive particles can help to understand if there is some relationship between their behavior and other aspects like their initial positions or the first part of their paths within the fundamental square.
In the final version of `Reader_poincare.m`, it is possible to choose up to three colors, which are usually (but not necessarily) assigned to:

- *Physically* trapped guiding centers, i.e. associated with a threshold value of 4.
- Untrapped guiding centers with very high max. distances. Their threshold values are typically chosen case by case.
- The remaining part of untrapped guiding centers.

4.8.4 Choice of a reasonable value for `threshold`

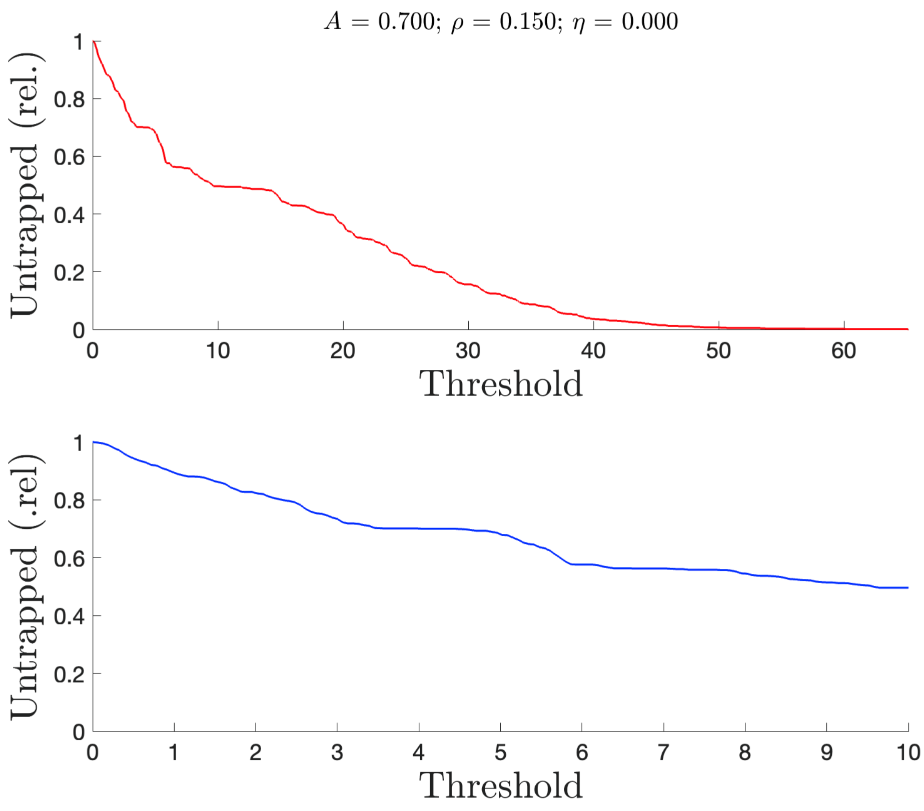


Figure 4.22: Percentage of *numerically* untrapped trajectories as the threshold value to compare with max. distances vary from 0 to the highest of max. distances. The blue curve is a zoomed-in version of the red one. It allows one to see the recurring plateau around the threshold value of 4 (other flat areas are present, but they tend to appear in different regions of the curves as other systems are considered. These plots have been made in MATLAB using the outputs of simulations with `Ntraj = 1024`, `Tf = 5000`, `TimeStep = 0.05` and `threshold = 0`.

In addition to the criterion for the max. distance, the other aspect that needed an improvement was the choice of `threshold`, which for a long time remained fixed at $3\pi \simeq 9.42$. This value seemed reasonable because it was comparable with the sides of the fundamental square. However, during the analysis of `.mat` files obtained with Python, it was discovered that a different value, more accurate and supported by numerical evidence, would have suited better. To do this:

1. Some simulations have been launched with `threshold = 0`, so as to be able to obtain only `numerically` untrapped particles, without distinctions of any kind.

2. The associated `.mat` files were analyzed with MATLAB in order to plot, as a function of a series of `threshold` values, the percentage of trajectories with max. distance greater than or equal to those values. The range of thresholds was only partially fixed and went from 0 to the maximum of all max. distances related to a single simulation.

Some graphs obtained in this way are shown in Figs. 4.22 and 4.23.

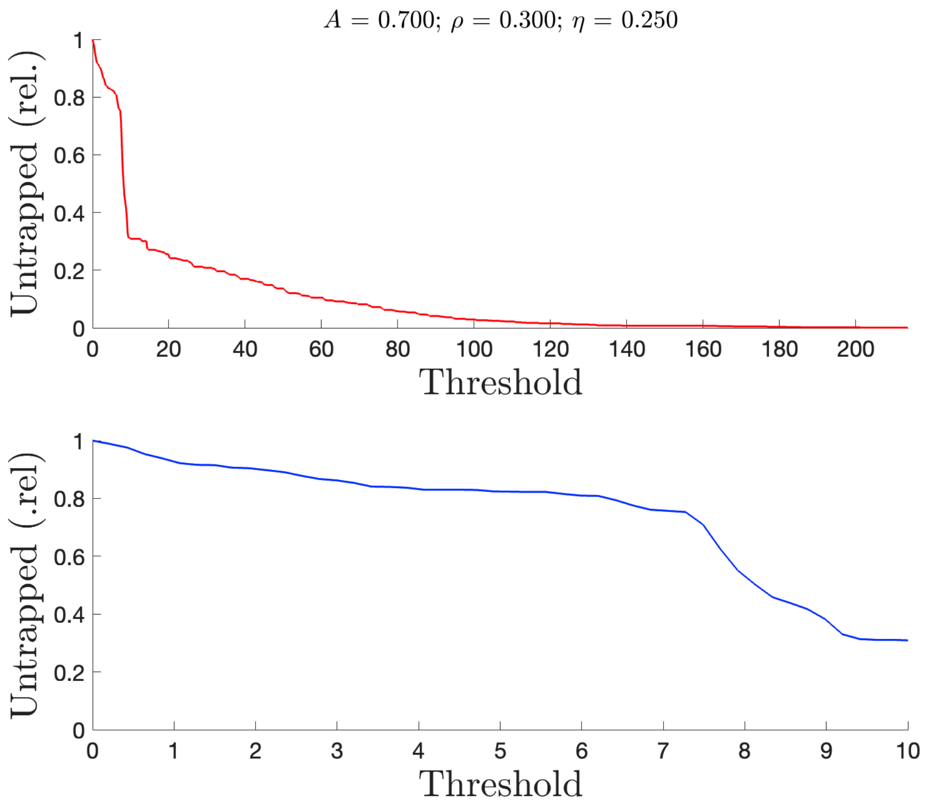


Figure 4.23: Percentage of *numerically* untrapped trajectories as the threshold value to compare with max. distances vary from 0 to the highest of max. distances, as in Fig. 4.22. These plots have been made in MATLAB using the outputs of simulations with `Ntraj = 1024`, `Tf = 5000`, `TimeStep = 0.05` and `threshold = 0`.

From these figures (and many others that were produced), it is clear that there is a tendency of the curve to flatten at around a threshold value of 4, which suggests that trapped trajectories tend to be associated with rectangles whose dimensionless diagonals are not bigger than 4. It is evident from the graphs that 4 is still an arbitrary choice, but it is certainly more accurate than 3π. This evidence was also checked with the help of stroboscopic plots (see, for example, Fig. **??**, where the islands have been correctly identified using a magenta color associated with `threshold = 4`). Moreover, having a recurring plateau around 4 could be seen as a way to characterize the system from a physical point of view: islands tend to be no bigger than a certain size, regardless of the particular form of the guiding center potential. Moreover, threshold-defining plots give hints about the possible diffusion regime, since the horizontal axis is much longer in the case of high `b` (different systems are obviously compared keeping the same values for `Tf`).

4.8.5 Benefits of two-step integrations and choice of a reasonable value for `Tmid`

In the original version of the Python files there was no `TwoStepIntegration` option: all guiding centers were integrated the entirety of time cycles, and at the end only the untrapped portion contributed to the calculation of the diffusion coefficient. Given the nature of scripts, however, any element that can be used to reduce computational efforts should be adopted. One of these is precisely the use of a limited number of test particles that are still able to represent a plasma (in combination with and thanks to the fact that the modelled electric field is produced by the plasma), while another is the exploitation of guiding center theory itself. On a numerical level, other aspects are the limitation of the number of cycles and the `TimeStep` used for integrating the equations.
In any case, however, it is possible to introduce an additional level of improvement, which can act regardless of all the other expedients: breaking the integration into two steps. In essence, this allows one to save numerous resources that would be unnecessarily used (in most cases) to follow the evolution of trapped guiding centers.
From a purely numerical point of view, `TwoStepIntegration` is more effective if `threshold` is high and `Tmid` is low, because both contribute to collect a greater number of trapped particles. For this work, however, it is not possible to ignore the physics: `threshold` is set to 4, a value that is less than half the original 3π, and `Tmid` should be selected with cleverness too. After some tests, it was found out that after 800 cycles the ratio between simulated and untrapped guiding centers tended to remain more or less stable, meaning that in most systems the vast majority of trapped trajectories can be completely individuated within 800 cycles. Knowing that increasing the number of cycles has a somewhat limited effect on the computation time (see subsection 4.8.6), it seemed reasonable to decide that `Tmid` needed to be at least equal to two times 800: at 1500 cycles, the percentage of trapped particles is even more stable.

4.8.6 Choice of reasonable values for `Ntraj`, `Tf` and `TimeStep`

It is intuitive that having a higher number of trajectories or cycles causes an increase of the computation time (what is reported in this section is completely irrelevant in the case of `Method = 'potentials'`). The same effect is also given by a lowering of the time step used for integrating the equations of guiding centers. In general, variations of `Ntraj` and `TimeStep` have more marked effects on the duration of simulations. Some simulations were dedicated to finding a reasonable range of values for the three parameters. In general, a compromise between the duration and the accuracy of simulations has to be made:

- If `Ntraj` is too high, the simulations are very slow because the computer has to integrate numerous equations at the same time (2 for each trajectory), while if it is too low, then the results are likely to be inaccurate because the number of guiding centers is not sufficient to extract reliable data. This last aspect is evidently more relevant in the case of diffusive simulations, since for those aimed at obtaining stroboscopic plots, in certain cases, it may helpful to track a very limited number of trajectories. Among the various possible cases, it can be observed, for example, that a too small `Ntraj` can be particularly counterproductive for those systems with *intermediate characteristics*, that is, those that have a high but not excessively high number of trapped particles in percentage terms: the risk is to consider as non-diffusive (`a` close to 0) a system that is actually weakly diffusive (`b` slightly below 1) only for lack of data.
  Usually, if in a simulation with high `Ntraj` (and other appropriately chosen parameters) only very few guiding centers are untrapped, then the system can be said to be non-diffusive, that is, as if it were composed only of trapped particles.

- If `Tf` is too high, the simulations undergo a moderate slowdown (the time increases approximately linearly with the increase of `Tf`) because the trajectories must be integrated for a greater number of time instants. This is a completely different computational effect with respect to that of increasing `Ntraj`, since `Tf` acts on the number of instants associated with each equation, and not on the number of equations to be managed for each of those instants. Moreover, simulations tends to lose more and more accuracy as they approach higher number of cycles, much like humans gradually age with time.
  If `Tf` is too low, simulations risk to be unrepresentative, because no sufficient time is given to untrapped guiding centers to evolve properly. This could cause some errors in identifying the diffusive regime of the system. Therefore, unlike `Ntraj`, `Tf` should be high regardless of the type of simulation.

- If `TimeStep` is too low, the simulations are highly accurate, but the equations take a very long time to integrate as the number of cycles is divided into a greater number of instants. Instead, using time steps that are too high is to be avoided because the risk is to have trajectories that differ highly from the more precise ones, that is those obtainable with very small time steps.

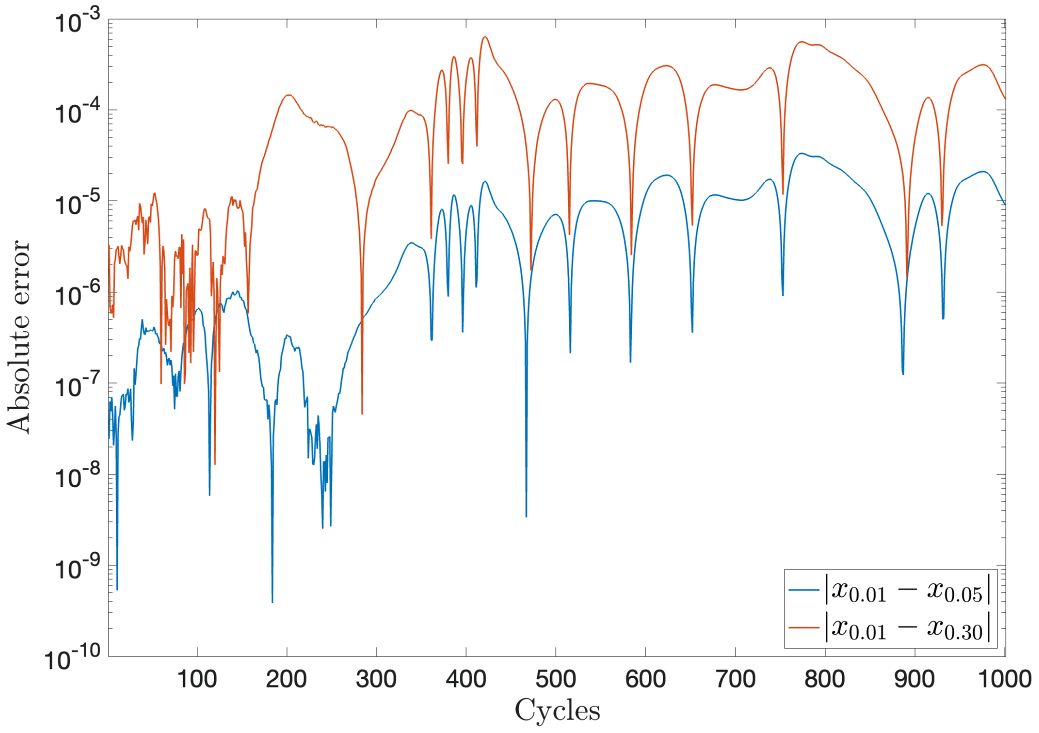


Figure 4.24: Absolute differences between the x-coordinates assumed, at the end of each cycle, by a single trapped guiding center using different time steps. The simulations have been launched with `A = 0.100`, `rho = eta = 0.00` and with three different time steps: `0.01` (the reference one), `0.05` and `0.30`. This is just a semi-logarithmic graph that wants to give an indication of what happens if different time steps are used.

The conclusions that have been reached after numerous simulations (with `TwoStepIntegration = False`) are:

- `TimeStep` equal to 0.05. The suggested value in the original dictionary was 0.03, but some preliminary investigations did not reveal any particular differences in the accuracy of results. In the case of high definition simulations, a value of 0.01 can be used, but the tunings for `Tf` and `Ntraj`, as well as all the simulations that are mentioned in this report, were done using `TimeStep= 0.05`. Just to give an idea, using `TimeStep = 0.01` is around 5 times slower than `TimeStep = 0.05` when `Tf = 500`, `Ntraj = 484`, `eta = 0` and `TwoStepDiffusion = False`, i.e. under very light conditions.

- `Ntraj` greater than 1000. By using 1024, which is a perfect square, it is ensured that there is an equal number of simulated test particles, regardless of `init`.
- `Tf` equal to or greater than 5000. This is perhaps higher than necessary, but this does not heavily influence the computation time as increasing `Ntraj` or `TimeStep` would. The idea of using more than enough cycles was also applied when selecting a reasonable value for `Tf`.

4.8.7 Choice of reasonable ranges for `A`, `rho` and `eta`

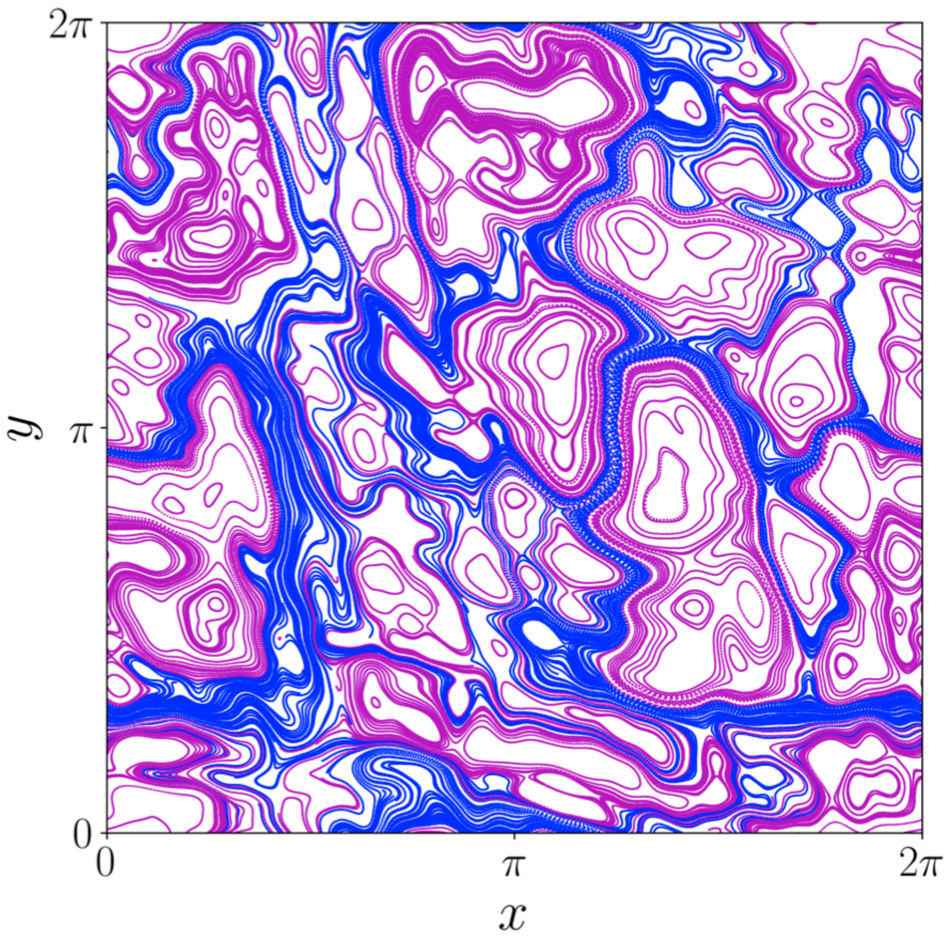


Figure 4.25: Modular stroboscopic plot, made with Python, that shows very regular trajectories. It was obtained with `Ntraj = 400`, `Tf = 1000`, `TimeStep = 0.05`, `threshold = 4`, `A = 0.150`, `rho = 0.050` and `eta = 0.000`. Such systems, given that they have a very high percentage of trapped guiding centers (in magenta) can be considered as non-diffusive.

The ranges of values for `A`, `rho` and `eta` that have been used in the final set of simulations are a subset of the ones chosen at earlier stages of this work.
Initially the focus was only on the first order of ψ, and therefore `eta` was set to 0. Under

these conditions, the ranges were from 0 to 1 for both the amplitude and the Larmor radius. In light of the numerous tests that have been performed, it can be said that:

- For `A` close to 0 (more or less up to 0.20), the system does not diffuse at all, regardless of the adopted Larmor radius.
- For `rho = 0` and `A` approximately in the range from 0.2 to 0.6, there is a transition from a non-diffusive to a diffusive regime. The associated simulations showed problems in the definition of a diffusion coefficient with the original version of the Python files, in the sense that there were unacceptable values of `R2`.
- Too high values of `rho` are unrealistic (this is mostly based on conversations I had with the people that worked with me).
- The combination of medium to low values of `A` and medium to high values of `rho` tends to cause an excessive slowdown in the dynamics of the guiding centers, which makes those systems particularly difficult to study with a reasonable setting of `Tf`. In other words, there are combinations of `A` and `rho` for the first order of ψ whose trapped trajectories take a very long time to close and whose untrapped guiding centers do not travel very far from their initial positions.

In light of these considerations, it was decided by mutual agreement to limit the ranges for the *definitive* simulations for both the amplitude and the Larmor radius. In particular, `A` has a range from 0.6 to 1.0, while the one for `rho` starts at `0` and ends at `0.3`. These values:

- Are associated with the zone just outside the transition zone from non-diffusive to diffusive regimes.
- Leave out those systems that take very long to evolve or that are non-diffusive at all.

These choices can also be justified from another point of view, which has to do with the regularity or chaos in the system. The selected ranges tend to have a good mix of regular and chaotic trajectories, at least when just the first order of ψ is considered. Near `A = 0`, regularity has the tendency to prevail, while systems are almost entirely chaotic for `A` above 1. It is more interesting to investigate those systems where both behaviors are present.
However, this way to fix ranges is not unique: it would have been equally possible to start from `A = 0.5` or to investigate `rho` up to 0.4, for example, everything in this section would have still been presented in the exact same way.
In later stages of the work, it had also to be understood which values of `eta` would have suited the second order of ψ. Considering that the second order is usually not employed

in gyrokinetic codes, it was assumed that `eta` should have been small in magnitude. The chosen range for `eta` includes both negative and positive values, but at the same time it is asymmetric, as it goes from -0.2 to 0.3. These particular selection was made to recover some sort of symmetry in certain portions of the results presented in Chapter 5.

Parameter	Value, range and/or comments
`A` (input)	from 0.6 to 1 amplitude of the potential (corresponds to A)
`rho` (input)	from 0.0 to 0.3 Larmor radius (corresponds to ρ)
`eta` (input)	from -0.2 to 0.3 intensity of $\psi^{(2)}$ (corresponds to η)
`threshold` (input)	4 threshold value for trapped trajectories
`Ntraj` (input)	≥ 1000 (`diffusion` mode only) number of test particles
`Tf` (input)	≥ 5000 number of cycles
`Tmid` (input)	≥ 1500 intermediate number of cycles if integration is in two steps
`TimeStep` (input)	0.05 time step for integrating the equations
`a` (output)	found through `scipy.optimize.curve_fit` using $f(t) = (at)^b$ coefficient of the curve fitting
`b` (output)	found through `scipy.optimize.curve_fit` using $f(t) = (at)^b$ exponent of the curve fitting
`R2` (output)	found through `sklearn.metrics.r2_score`, acceptable if ≥ 0.99 R^2 of the curve fitting
`trapped` (output)	found through `compute_untrapped` number of trapped trajectories

Table 4.1: Summary of the main parameters in input and output.

Original version	**Latest version**
Integration of the equations in one step	Integration of the equations in one or two steps
Monocolored stroboscopic plots	Multicolored stroboscopic plots
Max. distances based on initial positions	Max. distances based on rectangles around trajectories and their diagonals
Linear curve fittings and diffusion coefficients	Power-law curve fittings

Table 4.2: Main modifications to the Python scripts.

Chapter 5

Simulations and analysis of the results

Launching simulations with the final versions of the Python scripts is driven by the willingness to address some questions, such as:

- what are the effects sorted by A, ρ and η on the number of trapped particles, a, b and Poincaré Sections? In other words, can patterns be found in the behavior of the system as its parameters A, ρ and η vary?
- Are curve fittings acceptable?
- Are there any relevant effects caused by considering the second order of ψ with respect to the cases with $\eta = 0$? In other words, is the second order potential negligible or not?
- Are there connections between results obtained in `diffusion` mode, stroboscopic plots and the effects of A, ρ and η on ψ?
- Are there any really unexpected behaviors for some values of A, ρ and η?
- What things can be further investigated on the basis of these simulations and the post-processing of their outputs?
- Does opposite values of η sort similar effects on the system?
- Are there any relevant differences in using $\rho = 0$ or finite values for the Larmor radius?
- Can systems with different diffusive regimes be distinguished by looking at their stroboscopic plots?
- Is it possible to obtain useful information about the diffusive regime of a system by looking at curve of the percentage of its *numerically* untrapped particles?

In some sense, there are considerations and results in Chapter 4 that can be viewed as actual results of simulations, such as the recurrence of the number 4 when dealing with trapped trajectories, or the fact the ratio between untrapped and simulated particles tends to change very little after a certain number of cycles is reached. At the same time, most of what has been reported in Chapter 4 is not accompanied by detailed descriptions of what specific simulations have been conducted to get to this or that conclusion. This is justifiable by the many simulations that have been carried out as *feedback tools*. In other words, it is not far from truth to say that almost over two months of simulations have been analyzed before reaching the conclusion that, for example, `Ntraj` should be no smaller than 1000. This is because those simulations have not been launched with the specific idea of fixing the number of trajectories: they were used to see how the runs behaved, how much the scripts were able to properly match the physics behind the problem and, therefore, being able to improve both the files in Python and MATLAB. The exact same description can be applied to almost every other aspect that has been presented in Chapter 4.
However, it is not the same in Chapter 5. The results that will be presented in the following were obtained specifically with the idea of getting them, i.e. knowing that all the *feedback* simulations allowed to have both a convincing set of values for all the parameters in the dictionary and an improved version of all the files that have been described in Chapter 4. For this reason, the results will be presented together with a clear description of the values assumed by all the relevant parameters. This will help to get a better analysis of the results and ensures that the simulations and their outputs can be perfectly reproduced.
To sum up, *feedback* simulations (Chapter 4) had the main purpose of improving the numerical tools, but still managed to produce some interesting results from a physical or mathematical point of view, while *final* simulations (Chapter 5) were made to get to interesting conclusions on the strength of already improved files at disposal and values of the parameters.

5.1 List of the simulations that have been launched

Among the simulations, a preliminary distinction has to be made between those with `Method = 'diffusion'` and the ones with `Method = 'poincare'`:

- For both cases, the following parameters have been fixed (unless otherwise specifically indicated): `FLR = ('all', 'all')` (as by default), `M = 25` (as by default), `N = 2**10` (as by default), `Ntraj = 1024`, `Tf = 1000`, `TimeStep = 0.05` and `init = 'fixed'`. Moreover, `phases` have been defined using `numpy.random.seed(27)` (the number 27 is not an actual parameter, but it is something that in theory could be changed).

- In `'diffusion'` mode: `threshold = 4`, `TwoStepIntegration = True` and `Tmid = 1500`.
- In `'poincare'` more: `threshold = 0` and `TwoStepIntegration = False`.

A list of the physical parameters `A`, `rho` and `eta` that characterize each single simulation is presented below:

- In `'diffusion'` mode:
  1. `A = numpy.linspace(0.60, 0.10, 9)`,
     `rho = numpy.linspace(0.00, 0.30, 150)`
     and `eta = 0.00` (see subsection 5.2.1).
  2. `A = 0.70`, `rho = numpy.linspace(0.00, 0.30, 28)`
     and `eta = numpy.linspace(-0.20, 0.30, 125)` (see subsection 5.2.2).
- In `'poincare'` mode:
  1. `A = 0.70`,
     `rho = numpy.linspace(0.00, 0.30, 2)`
     and `eta = 0.00` (see subsection 5.3.1).
  2. `A = 0.70`,
     `rho = 0.115`
     and `eta = numpy.linspace(-0.10, 0.10, 2)` (see subsection 5.3.2). In this case, `init = 'random'`.

From the list of simulations in `diffusion` mode, it is clear that:

- The first step was identify a general behavior of the system without the second order of ψ, i.e. under the approximation of considering the particles in the same positions of their guiding centers. This part was driven by the unexpected discovery of anomalous diffusion from previous simulations.
- Focusing on a single value of the amplitude, the next natural step was to identify a behavior when the second order of ψ was also taken into account. This part has to do with was what was originally planned to do, i.e. investigating the effect of a term which is usually neglected.

The choice of not wanting to explore the effects of varying `eta` for more values of the amplitude had to do with the fact that non-zero values of `eta` significantly increase computational time (just to give an idea, they tend to be slower by almost an order of magnitude, e.g. simulations with non-zero `eta` and associated with 100+ sub-simulations could take more over week to complete, even when highly performing computers and parallel computing are exploited). But still, useful information on the modelled system can be obtained even with this limited approach.

5.2 Analysis of the results

5.2.1 Effects of A and ρ on percentage of trapped guiding centers, a and b through the first order of ψ

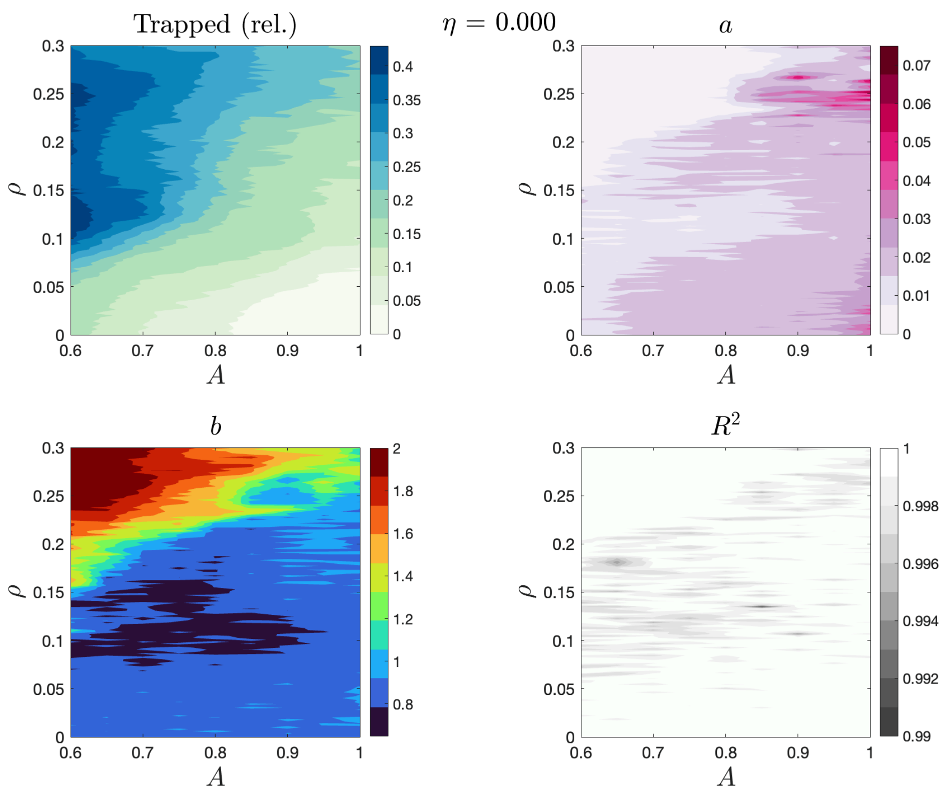


Figure 5.1: Graphical representation of the results of the first simulation in `diffusion` mode. They are four `contourf` plots obtained with MATLAB by post-processing the `.txt` file obtained with Python. Each figure has 10 distinct colors in its *colormap*.

The output `.txt` file from the first simulation in `diffusion` mode has been post-processed with MATLAB, resulting in a series of plots that display the percentage of trapped particles, coefficients a and b from curve fittings and R^2 as functions of A and ρ, with no second order effects. Fig. 5.1 sums up the results in multicolored `contourf` plots, in which *colormaps* with only 10 colors have been used to better highlight transitions and behaviors.
In order to avoid confusion, four different *colormaps* have been used, with some of them

(all but the one for b) being obtained with the help of the freely downloadable file `brewermap.m`. Moreover, the same *colorbar* limits (upped and lower limits change from quantity to quantity) and number of colors have been used to produce similar figures, but associated with different simulations (see, for example Fig. **??**).

Effects on Trapped (rel.)

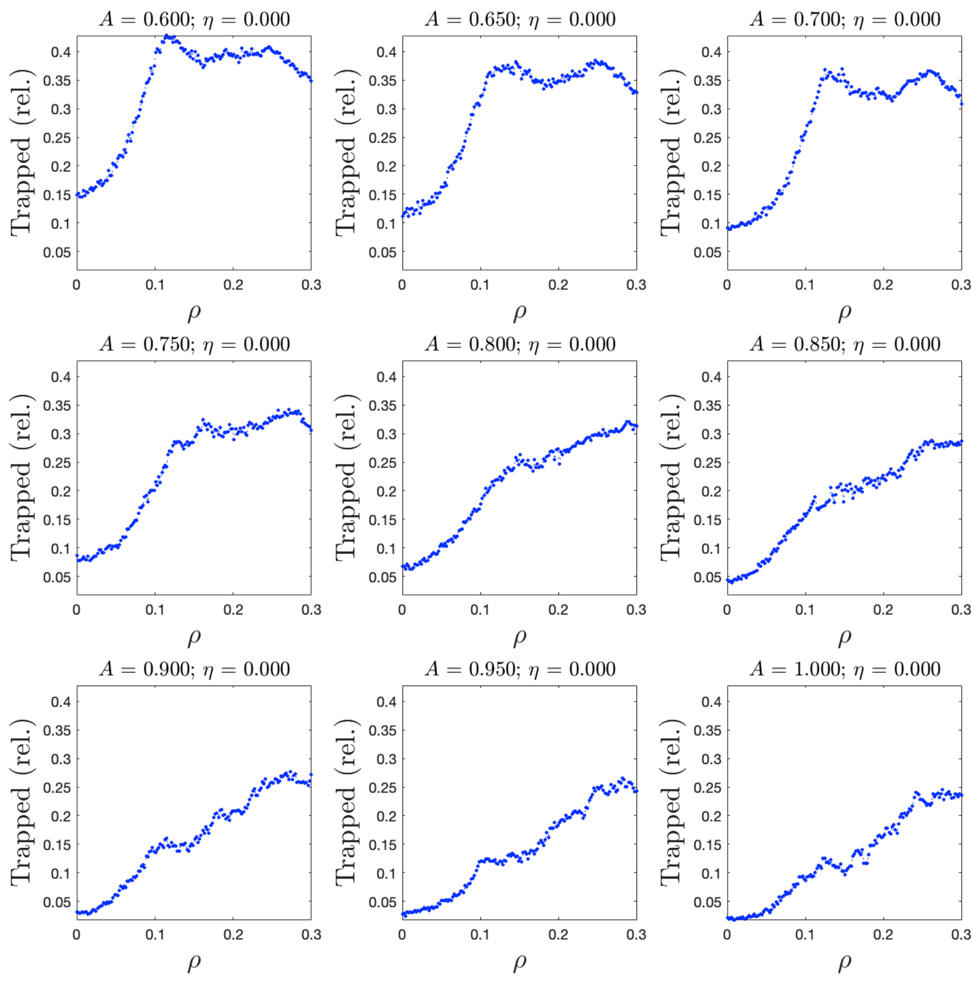


Figure 5.2: Plots obtained with MATLAB of the percentage of trapped particles as a function of ρ, with A and η fixed. They are associated with the first simulation in `diffusion` mode.

The general trend of the percentage of trapped particles can be appreciated by looking at either the upper-left plot in Fig. 5.1 or the nine subplots in Fig. 5.2:

- An imaginary and almost straight line could be drawn in the `contourf` plot connecting the bottom-right and the upper-right corners. Below this line, the percentage of trapped guiding centers is more or less within the range $0\% - 25\%$, while above it the range is roughly $25\% - 50\%$ (these *clean* values are used on purpose to stress the different behaviors). The lowest values correspond to high amplitudes of the potential and small Larmor radii, i.e. the bottom-right corner. The highest values are found along the vertical line on the opposite side, where A is around 0.60.

- Two dark-blue spikes, pointing to the right, can be easily identified on the left side of the `contourf` plot ($\rho \simeq 0.10$ and $\rho \simeq 0.25$). Both tend to propagate and decay moving towards the right side of the plot, i.e. less-intense spikes are visible within almost each other color. The lower spikes are more or less associated with the same values of ρ as A increases, while the higher ones have the tendency to slightly bend towards higher values of the Larmor radius. This trend can be also appreciated by looking at Fig. 5.2, where it is revealed that the spikes tend to be less and less recognizable as A becomes bigger. In particular, starting from $A = 0.750$, the spikes gets confused within the rest of the points.

- As it is evident when the darker spikes are taken into account, the *valley* between them is approximately in the range $0.15 - 0.20$ for the Larmor radius.

- The fact that fewer trapped particles are found for higher values of the amplitude makes sense: the potential becomes more corrugated and guiding centers trajectories more chaotic.

- The fact that fewer trapped particles are found for lower values of the Larmor radius makes also sense: the more ρ is near 0, the more it has a lighter effect on the first order of ψ, with the effect being a relaxation of the waves. In fact, it can be recalled that $\psi = \phi$ when $\rho = \eta = 0$.

- In general, when both A and ρ increase, there are less trapped particles and the associated curves are less irregular, becoming more similar to straight lines with a positive slope. This trend is even more clearer by looking at Fig. 5.3.

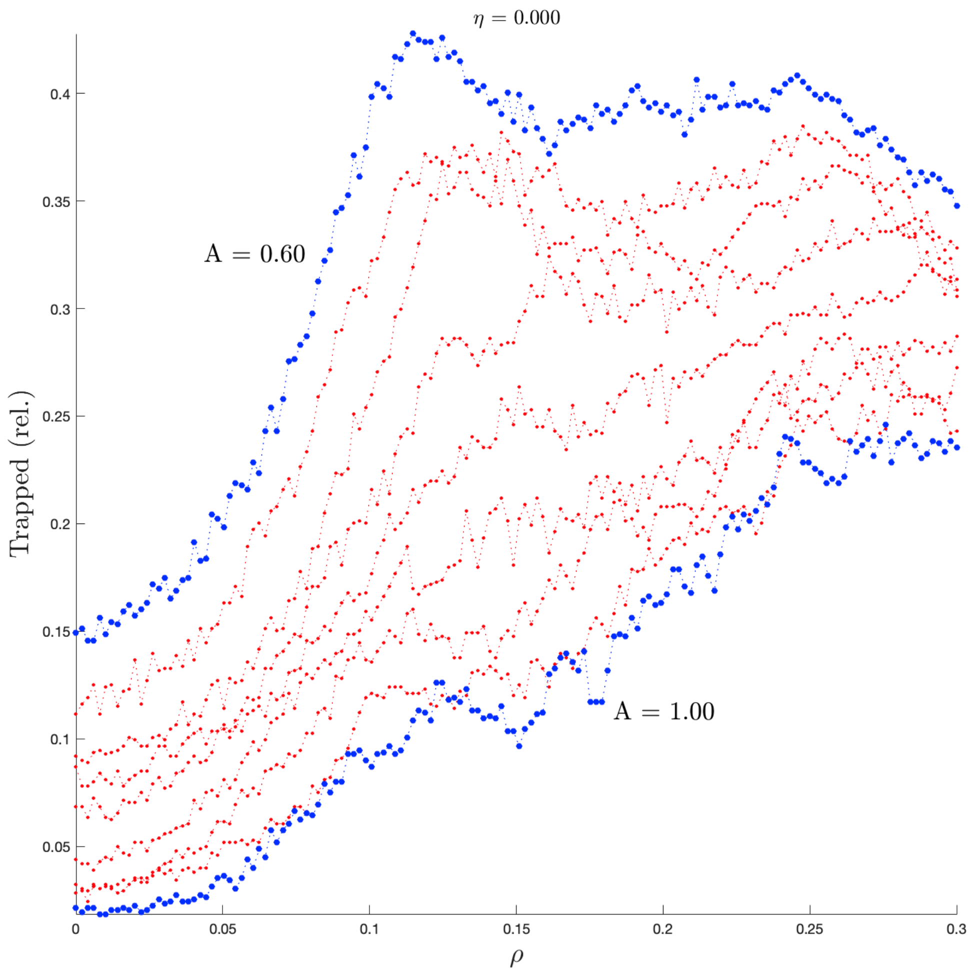


Figure 5.3: A different version of Fig. 5.2 with all the sub-plots that are merged in a single figure.

Effects on b

In analogy with the previous analysis, investigating the behavior of b can be done with either 5.1 or 5.4:

- Just by comparing how the colors are spread out in the `contourf` plots for b and trapped guiding centers, it is immediately evident that the first has much more violent transitions, i.e. there are colors that occupy considerably less portions of the b plot than the trapped one.

- The vast majority of pairs (A, ρ) have an almost diffusive behavior or a slightly sub-diffusive one (cold-colored areas);
- there is a small range of the pair (A, ρ) for which the system is strongly super-diffusive (hot-colored areas), i.e. almost ballistic;
- as already pointed out, the regions where the system is mildly super-diffusive (greenish and yellowish areas) are very limited in extension regardless of the position in the `contourf` plot, suggesting that moving from diffusion to super-diffusion is an abrupt process. The transition happens approximately along a diagonal that goes from $(A, \rho) = (0.60, 0.15)$ to $(A, \rho) = (1.00, 0.30)$, even though some exceptions are present.
- Before reaching the super-diffusive regions, there is an area with the lowest values of b, which can be more clearly identified by looking at Fig. 5.4.
- In general, moving towards higher values of ρ with A fixed, causes the system to gently move from diffusive ($\rho \simeq 0$), to slightly sub-diffusive, to diffusive again and then, very rapidly, to almost ballistic.
- The abrupt transition of b towards super-diffusive regimes are very clearly depicted in the subplots of Fig. 5.4, where it also shown that the change of behavior required higher Larmor radii as the amplitude of the potential increases. For the highest values of A the transition seems to become less marked, suggesting maybe that is happens for non-investigated values of ρ, i.e. above 0.3. In addition to that, high A also cancels the sub-diffusive behavior. It seems that having more chaotic dynamics is connected with diffusive regimes (at least when only the first order of ψ is considered).
- The fact that the transition happens very late in ρ (above 0.15) makes an interesting effect when all the separated curves of Fig. 5.4 are put together in a single plot, as displayed in Fig. 5.5. Using blue to represent the curves for both $A = 0.6$ and $A = 1.00$, with the others being in red, allows one to notice that, for $\rho < 0.15$, it is very difficult to recognize two distinct blue lines, while just beyond $\rho = 0.15$ there is a complete split between the lines (even the red ones). This suggests that, at least for the first order of the potential, a small Larmor radius can influence the number of trapped particles much more than the diffusive regime. Moreover, given that ρ sorts a different effect in the two cases (in terms of how fast and where transitions happen), it seems that the number of trapped particles has a relatively limited relation with the diffusive regime, with the most interesting aspect being that the largest differences between blue lines of trapped trajectories in Fig. 5.3 take place just before the threshold value $\rho = 0.15$.

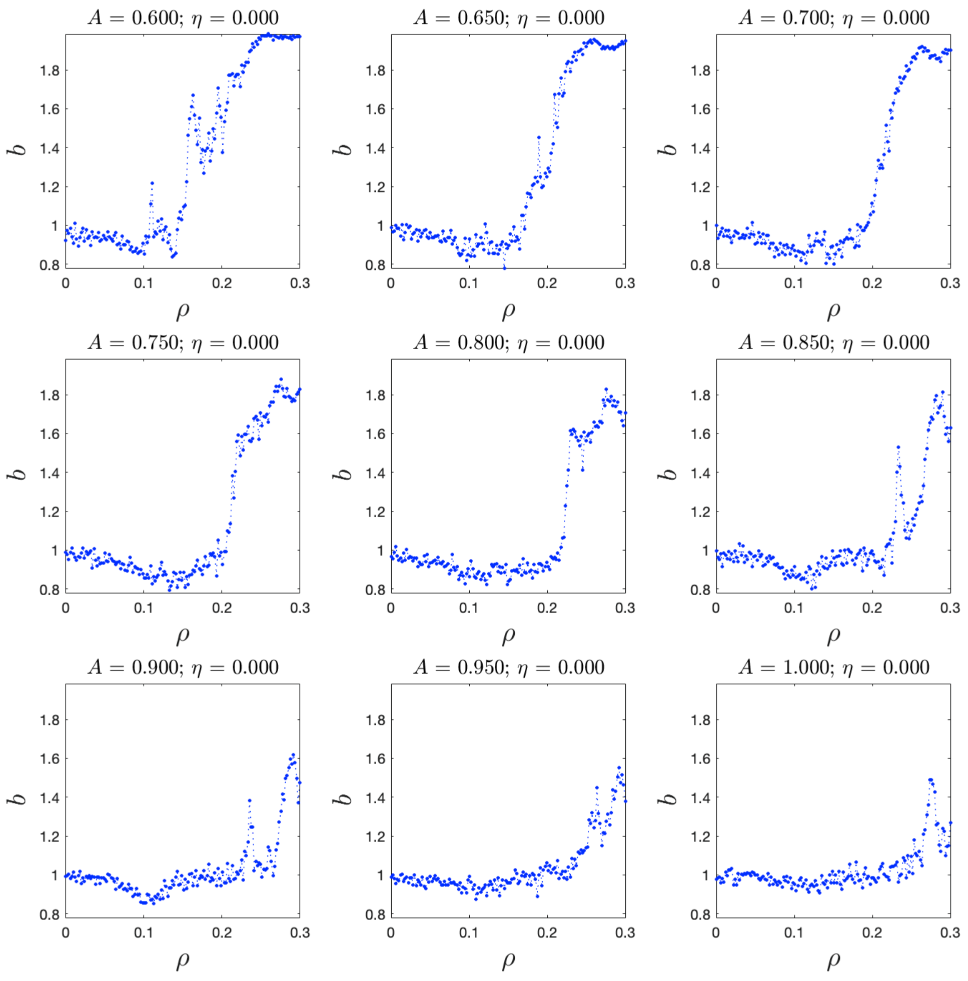


Figure 5.4: Plots obtained with MATLAB of the curve fitting exponent b as a function of ρ, with A and η fixed. They are associated with the first simulation in `diffusion` mode.

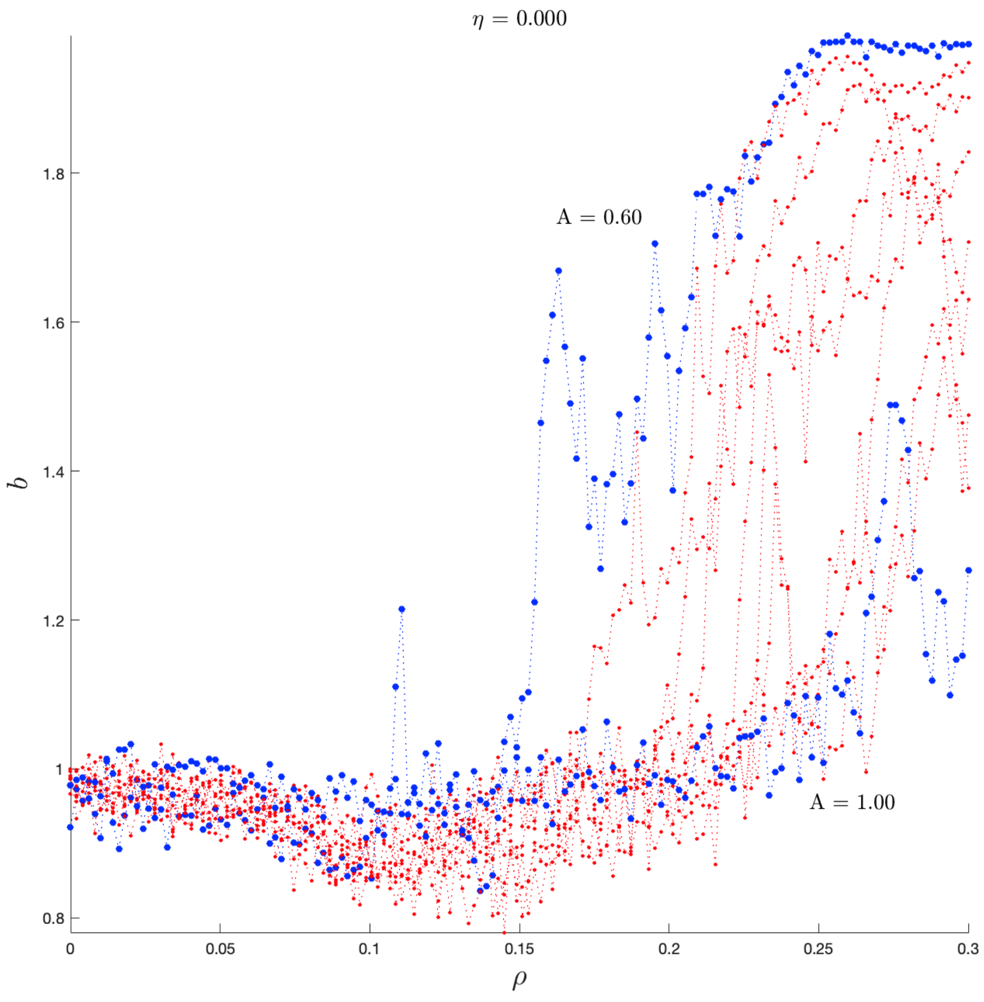


Figure 5.5: A different version of Fig. 5.4 with all the sub-plots that are merged in a single figure.

Effects on a

Fig. 5.1 can be also exploited to analyze the effect that A and ρ have on the coefficient a, but before doing that it is important to underline that the physical meaning of a (even though in this report it is being considered after a nondimensionalization process) changes with b. This can be explained by going through the two simplest cases and recalling that $r^2(t) \longrightarrow f(t) = (at)^b = a^b t^b$:

- If $b = 1$, then $a^b = a^1 = a$ and, given that the r^2 would have the units of an area if it had dimensions, this means that a represents a (dimensionless) diffusion coefficient (area over time).
- If $b = 2$, then $a^b = a^2$ and, for the same reason as before, this time a is a (dimensionless) velocity.

Having said that, the following list of observations can be better interpreted:

- Roughly speaking, the upper-right plot in Fig. 5.1 can be divided into two main regions much like it was done for the trapped trajectories. This time, however, the curve delimiting the upper-left and the bottom-right regions should be more like an oblique sinusoid. While the division resembles the one for the trapped guiding centers, what can be said about these areas has more to do with b. In fact, the most striking fact is that a is very small and constant (white or almost white coloring) in the regions where the system is either sub-diffusive or super-diffusive, i.e. in correspondence of dark-blue and red areas within the `contourf` graph for b. Instead more shadings of purple and magenta, even if not many, can be found in those regions where the system is diffusive of almost diffusive.
- Particular attention has to be put towards the regions of almost ballistic regimes, because there the coefficient a is very close to 0. It seems that reaching such conditions has the effect of uniforming the curve fittings much more than in any other cases. This aspect is even more evident when looking Fig. 5.6. This plot also shows that a oscillates a lot more when A is closer to 0.6 than 1.0. In addition, it is interesting to notice that the highest and the lowest values of a are reached (for different values of A) within the same ranges of the Larmor radius, i.e. for $\rho > 0.15$.
- Similarly to the already investigated quantities, different systems have narrower red-and-blue dotted plots for smaller values of ρ, while they tend to more spread out after $\rho \simeq 0.15$.

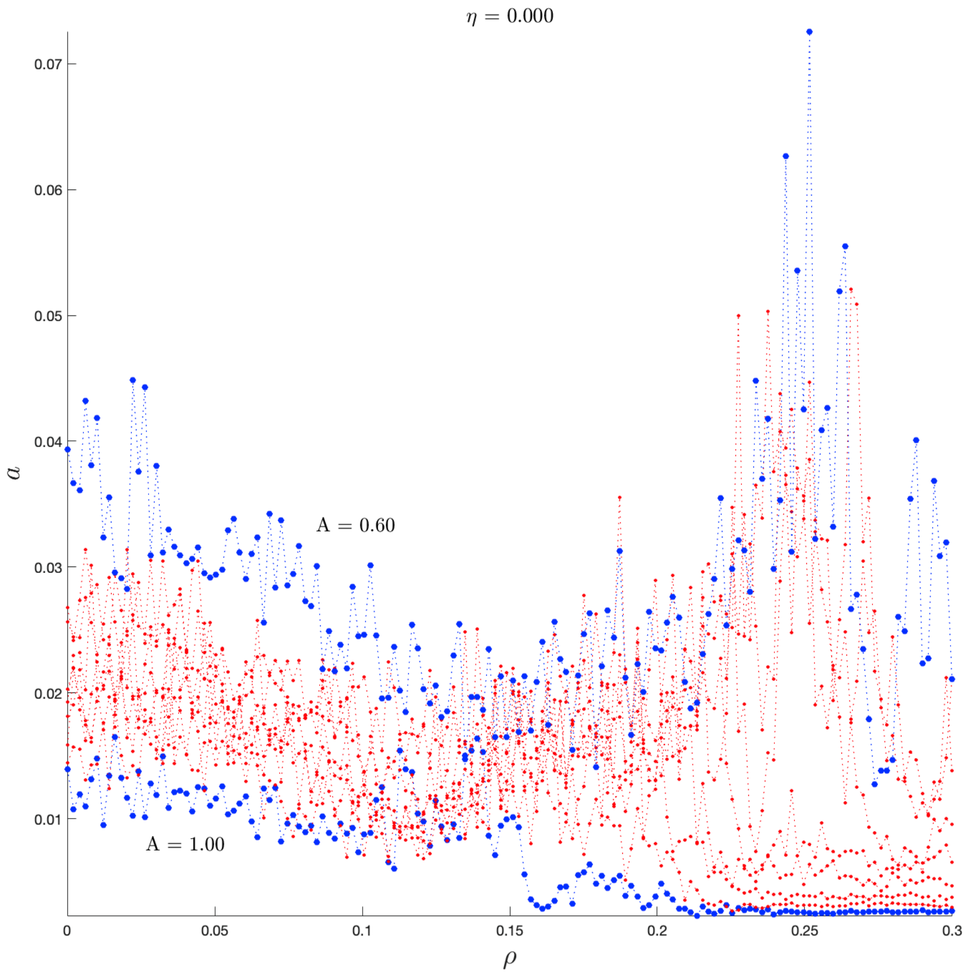


Figure 5.6: Merged plot obtained with MATLAB of the coefficient a as a function of ρ, with η fixed. Each curve refers to a different values of A. They are associated with the first simulation in `diffusion` mode.

Effects on R^2

A brief comment can also be given to the R^2 subplot of Fig. 5.1: all the values of the coefficient of determination are not below 0.99, meaning that all curve fittings can be treated as reliable.

Summary of the effects

To recap, increasing A tends to:

- Corrugate the potential.
- Make the dynamics more chaotic.
- Gently reduce the number of trapped guiding centers.
- Raise the values of a.
- Reduce the values of b when, at lower values, the system is super-diffusive.

Moving from $\rho = 0.00$ to $\rho = 0.30$ has the effect of:

- Relax the corrugation of the potential.
- Make the dynamics more regular.
- Gently increase the number of trapped guiding centers.
- Reduce and then suddenly raise the values of b.

Moreover, a particularly relevant Larmor radius seems to approximately be $\rho = 0.15$, because a blend of interesting phenomena happen around it:

- The relative number of trapped trajectories, at least for the cases in which $A < 0.75$, has one of the two peaks just before that threshold.
- The more ρ is far from the threshold value, the less the percentage of trapped particles are spread, with the systems tending to form narrower windows.
- The system tends to be diffusive or sub-diffusive before that threshold and, somewhere near it or after it, the regime abruptly becomes almost ballistic. In other words, different systems have much more similar values of b before $\rho \simeq 0.15$ rather than after.
- Similarly to b, different systems have much more similar values of a before $\rho \simeq 0.15$ rather than after, with the narrowest window of values of a among the systems being reached almost exactly at $\rho = 0.15$. Another way of describing this phenomenon is that the more ρ is far from the threshold value (in both directions, but predominantly towards $\rho = 0.30$), the more the values of a are widely spread.

5.2.2 Effects of ρ and η on percentage of trapped guiding centers, a and b through both orders of ψ

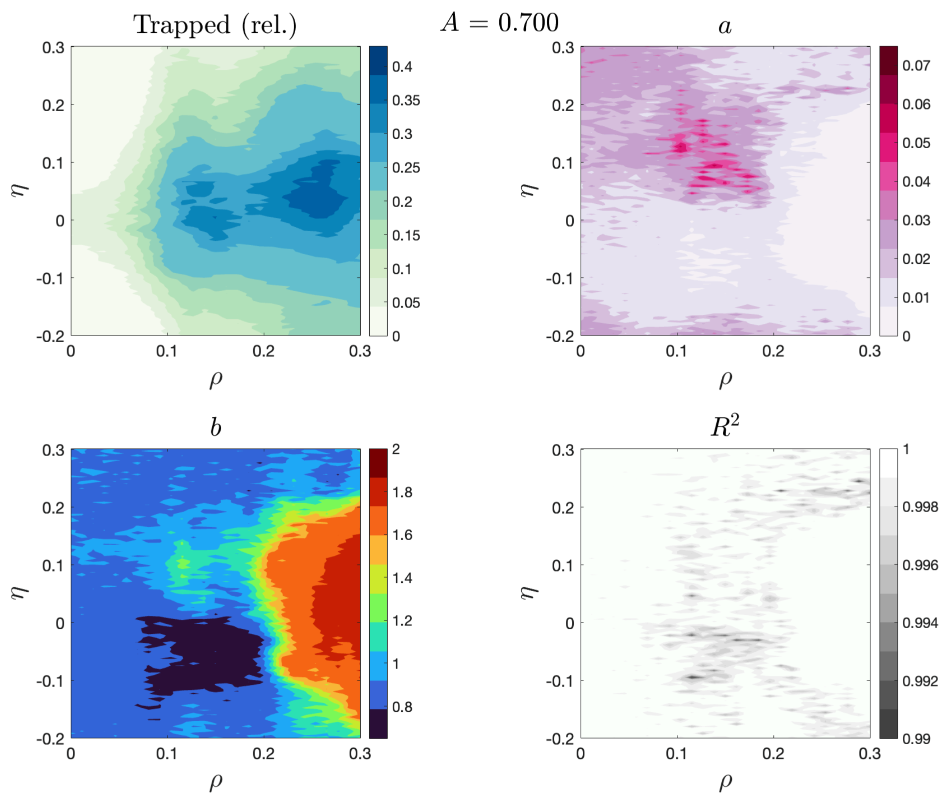


Figure 5.7: Graphical representation of the results of the second simulation in `diffusion` mode. They are four `contourf` plots obtained with MATLAB by post-processing the `.txt` file obtained with Python. Each figure has 10 distinct colors in its *colormap*.

Simulations that are associated with full ψ are longer to computer because of the complexity added by the second term. Given that the latter is usually neglected, it is reasonable to expect that simulations do not differ much if ψ or just its first order are used. However, the outputs revealed interesting scenarios, much like simulations with $\eta = 0$ highlighted the possibility of anomalous diffusion. In short, if η varies while A and ρ are fixed, then the behavior of the system changes (more significantly in some cases than others).
As in the previous case, reference `contourf` sub-plots have been obtained with MATLAB. They are displayed in Fig. 5.7.

Effect on Trapped (rel.)

With A fixed at 0.70, the percentage of trapped particles as ρ varies is influenced by the value of η. In particular, by looking at the upper-left subplot:

- There is a moderate symmetry along the vertical direction, which however is centered slightly above $\eta = 0$, where the coefficient is positive ($\simeq 0.05$).
- The general trend is that the number of trapped guiding centers decreases as $|\eta|$ gets bigger.
- The two spikes that were present in the graph associated with the first order of ψ are clearly evident here too (the darkest areas). They seem to have an effect on the shape of the differently colored levels, as it is possible to see some sort of *propagation* of these spikes. However, a closer look reveals that this propagation (i.e. a decay in their intensity) is not entirely symmetric. In fact, for $\eta > 0$, interfaces between different colors are like sinusoids, whereas for $\eta < 0$ they are more similar to straight lines. For example, a sign of asymmetry (or shifted and partial symmetry) is represented by the outer sides of the light-blue areas, given that they approximately reach $\eta = 0.20$ and $\eta = -0.10$.
- Large values of *eta* correspond to almost absence of trapped guiding centers. From a computational point of view, this means that the associated sub-simulations are longer and for them there is a very limited benefit of having implemented a two-step integration of the equations. Unfortunately, this adds up to fact that the second order of ψ is already demanding on its own (regardless of how many steps are used);
- As already seen for trapped trajectories, the transition from blue to light-green is rather smooth.

Effect on b

The scenario that is depicted in the bottom-left subplot of Fig. 5.7 reveals some very interesting aspects for the value of b:

- The plot can be vertically split into three regions, with the first approximately in the range $0.00 - 0.10$ of ρ, the second in $0.10 - 0.20$ and the third in $0.20 - 0.30$. An equal quantity of very different behaviors can be associated with each one of them. Within the left one, varying η has little to no effect, with the system remaining more or less diffusive or weakly sub-diffusive with respect to the reference values at $\eta \simeq 0$. In the middle section, some positive values of η cause the system to become slightly super-diffusive, while negative values maintain the system in its *first order* *state*, that is sub-diffusive. However, high values of $|\eta|$ lead to an almost diffusive behavior. The remaining regions is associated with a decay of the value of b, as $|\eta|$ increases, in such a way that almost ballistic regimes become diffusive.

- It is not clear why the system has opposite (while still not too different) behaviors when *first order regimes* are sub-diffusive. This is even more strange when it is taken into account that the associated range for η involves very small absolute values. This is one of the cases in which using stroboscopic plots can be particularly helpful.

Effects on a

Sub-plots for a and R2 are more or less similar to the previously investigated ones. In particular, a is small and constant in the sub-diffusive and highly super-diffusive areas, whereas it has higher and more diversified values in the diffusive and mildly super-diffusive ones. All the values of R^2 are above 0.99.

Summary of the effects

Below, the main effects of η are put together:

- Increasing $|\eta|$ corrugates the potential. In this sense, η is more similar to A than ρ.
- For $\eta > 0$, negative values of ψ prevail over positive ones in terms of how much area they involve.
- For $\eta < 0$, positive values of ψ prevail instead.
- Ultimately, when $|\eta|$ is sufficiently big, there are very few trapped guiding centers and the systems leans toward diffusive regimes.
- It can have asymmetric or opposite influences on some quantities (e.g. b) when certain pairs (A, ρ) are considered.

5.3 Analysis of some stroboscopic plots

5.3.1 Comparison between diffusive and almost ballistic regimes without the second order of ψ

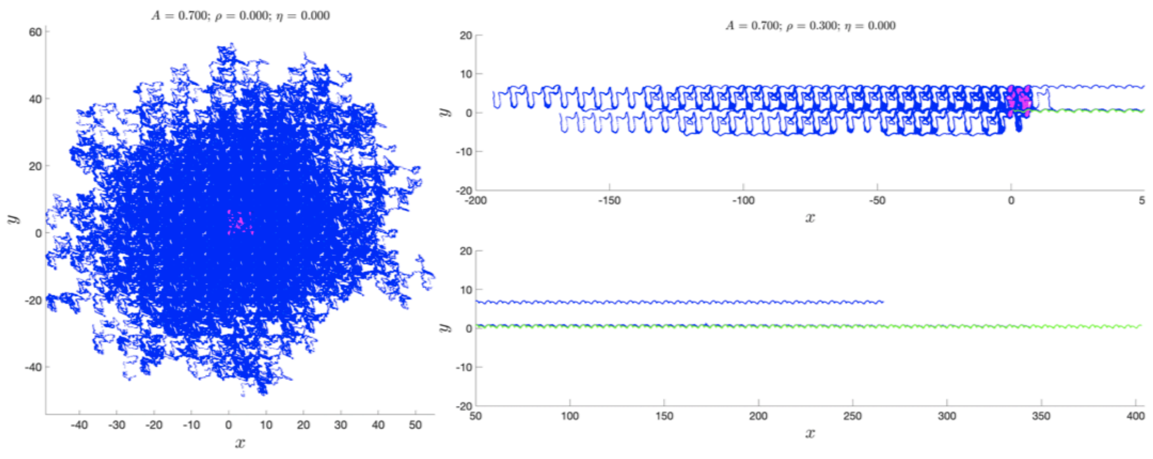


Figure 5.8: Comparison between a diffusive and an almost ballistic system through their non-modular stroboscopic plots (the one on the right has been split in two for clarity). Trapped trajectories are in magenta, trajectories with max. distances above 350 are in green, while the rest is in blue. They are associated with the first simulation in `poincare` mode.

There are some correlations between the results in `diffusion` mode and stroboscopic plots, which are more evident when extreme cases are considered. Other than Poincaré sections, another useful tool is provided by the graphs of untrapped trajectories as a threshold value is varied, because the shapes of their curves could present patterns or peculiar behaviors.
The starting point is Fig. 5.8, in which a comparison between non-modular stroboscopic plots for systems with $(A, \rho, \eta) = (0.70, 0.00, 0.00)$ and $(A, \rho, \eta) = (0.70, 0.30, 0.00)$ is made. They are associated with diffusive and almost ballistic regimes, respectively. From these plots, where magenta trajectories are trapped particles (max. distances no greater than `threshold = 4`), green trajectories have max. distances above 350 (arbitrarily chosen value, based on the right-hand plot in Fig. 5.12) and the blue ones are the remaining, the following can be said:

- Guiding centers in diffusive regime are more evenly distributed in the transverse plane, whereas those of the super-diffusive system are almost entirely developed only along the horizontal axis.

- Overall, trajectories in diffusive regime cover shorter distances and as a consequence their strboscopic plot is much more zoomed-in than the other. In other words, when a system is super-diffusive, axes have wider ranges (in this report, Poincaré sections have always an aspect ratio equal to 1).

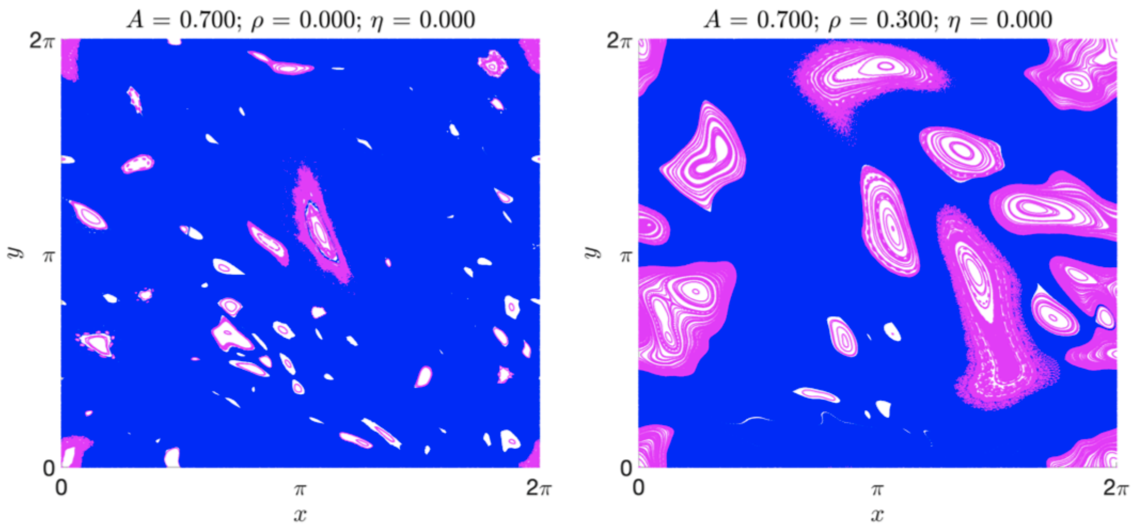


Figure 5.9: Modular version of Fig. 5.8, but with green trajectories that have been colored in blue as all the other untrapped ones.

Different aspects can be highlighted when modular versions of stroboscopic plots, which are displayed in Fig. 5.9, are taken into account (this time, all green trajectories have been substituted with blue ones, leaving just the distinction between trapped and untrapped guiding centers):

- Trapped trajectories are visually less and, on average, much smaller in the diffusive system. In addition to that, they occupy more diversified areas, meaning that there are more concentric islands in the super-diffusive case.
- Modular untrapped trajectories evenly cover the space in both plots (if single dots were smaller in size or if figures were zoomed, those blue areas would reveal an very high number of blue points over a white background).
- The super-diffusive system has a lot of regular trapped trajectories, with only a very limited amount of them being more chaotic. This is in accordance with the fact that increasing ρ relaxes the shape of ψ.

At this stage, there is little that could explain why the second system is super-diffusive. Actually, just considering modular plots, it could seem counterintuitive that the super-diffusive system is more regular and has more and bigger trapped trajectories. This is why twists on both modular and non-modular plots have to be made in order to highlight some otherwise hidden details. These twists are shown in Fig. 5.10 and **??**:

- The modular left plot in Fig. 5.10 shows in green those untrapped trajectories which are also highlighted in Fig. 5.8. The area occupied by them (in modulo version) is very precise and narrow. Moreover, it is well inside the blue area, that is there are blue regions of considerable size that separate green and magenta areas. This would seem to suggest that super-diffusive regime is activated because of a phenomenon that happens away from islands in the potential (but it requires additional investigation).

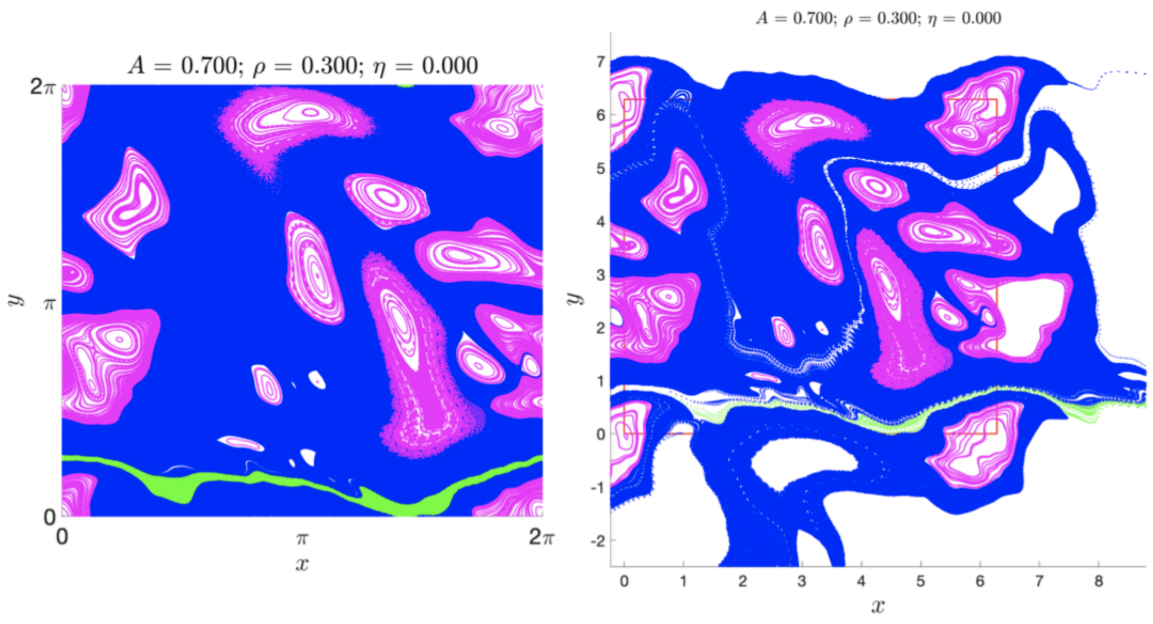


Figure 5.10: An almost ballistic system represented through its modular (on the left) and its zoomed-in non-modular Poincaré section (on the right). Untrapped trajectories with max. distances above 350 are in green. The non-modular plot clearly shows the presence of white *highways*. They are associated with the Poincaré section on the right side of Fig. 5.8.

- The non-modular right plot in Fig. 5.10 is zoomed-in and shows an area slightly wider than the fundamental square. Even though less clearly, the first part of those few green trajectories can still be noticed. Here, the advantage of using the non-modular Poincaré section is that it gives a strong hint about what could be associated with strong super-diffusion regimes, even more so if the non-modular Fig. 5.11 is used as a comparison, where a diffusive system is represented instead. In the almost ballistic system, it is as if white *highways* breach the blue areas within the fundamental square. Those *highways* are clearly not present in the diffusive regime, suggesting that they appear as a consequence of increasing ρ over a certain value (it has to be kept in mind that $\rho > 0$ can also lead to slightly sub-diffusive conditions). If the *green threshold* were reduced to 300 or 250, for example, then most of the additional areas to become green would be the ones next to or inside *highways*.

- A possibly interesting aspect, that can be verified by using the cursor in MATLAB, is the fact that trapped trajectories on opposite sides of highways (and close to them) tend to have different curls, i.e. if one evolves clockwise, the other spins counterclockwise.

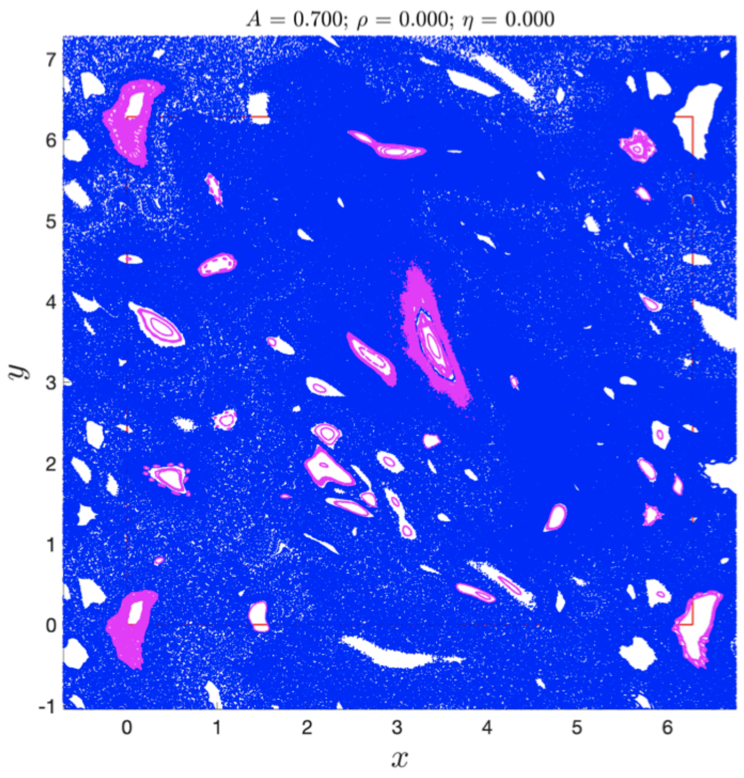


Figure 5.11: Zoomed-in version of the non-modular stroboscopic plot on the left side of Fig. 5.8. The fundamental square can be seen, in red, in some areas.

Some other pieces of information are obtained by looking at Fig. 5.12, where the percentage of *numerically* untrapped trajectories is plotted:

- Focusing on the plots with red curves, the horizontal axis for the diffusive system is considerably shorter (by a factor close to 6). This can be seen as another representation of non-modular stroboscopic plots of Fig 5.8, with one being much more developed in a specific direction than the other.

- Focusing on the plots with blue curves, which are just zoomed-in versions of the others around the zone of `threshold = 4`, reveal that in both cases 4 is associated with a plateau. However, there is a major difference, given that the diffusive system has a much extended flat zone than the super-diffusive one. In addition and that, these plots confirm the visual sensation that trapped trajectories are smaller and less in quantity. The plateau for the plot on the right starts approximately when threshold is 1 and the percentage of untrapped guiding centers is well above 90%, whereas in the other case the starting point is $\simeq$ 3.0, 70%.

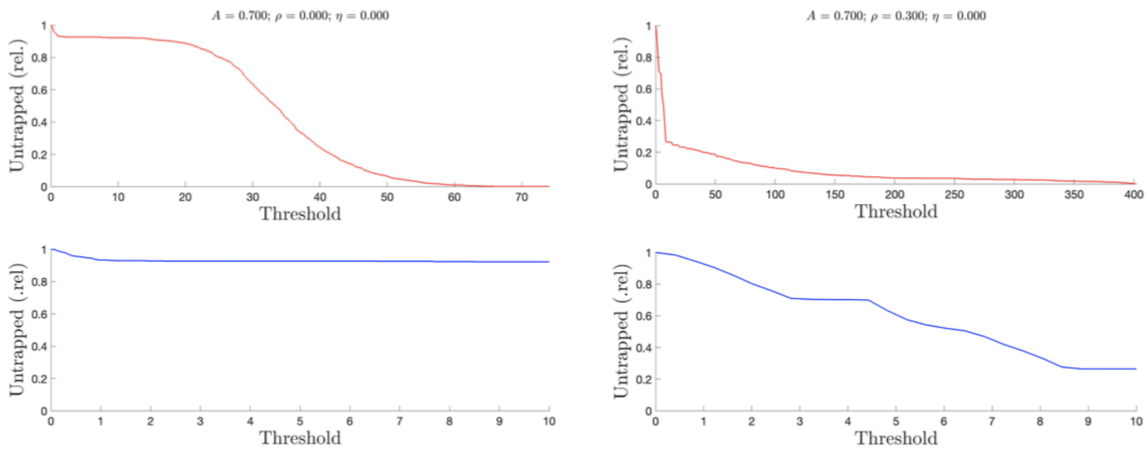


Figure 5.12: Visual comparison between the same systems of Fig. 5.8, but through the variation of the percentage of *numerically* untrapped trajectories as a function of the threshold value for max. distances. Flat zones can be spotted around the threshold value of 4 in both plots.

- The red curve on the left is rather smooth, with two relatively flat zones connected by an oblique section of transition. In contrast to that, the red curve on the left can be divided in two main areas which are very different from one another. The first, which strongly descends, ends around the point 10.0, 30%). The second, which occupies a wider range of threshold values, has a very soft decay, but at the same time appears abruptly and sharply. It seems that the super-diffusive system tends to have sharper transitions and this is somehow in line with the formation of *highways*, whereas the smoothness that characterizes the diffusive system is relatable to having evenly distributed trajectories in the transverse plane (both inside and outside the fundamental square).

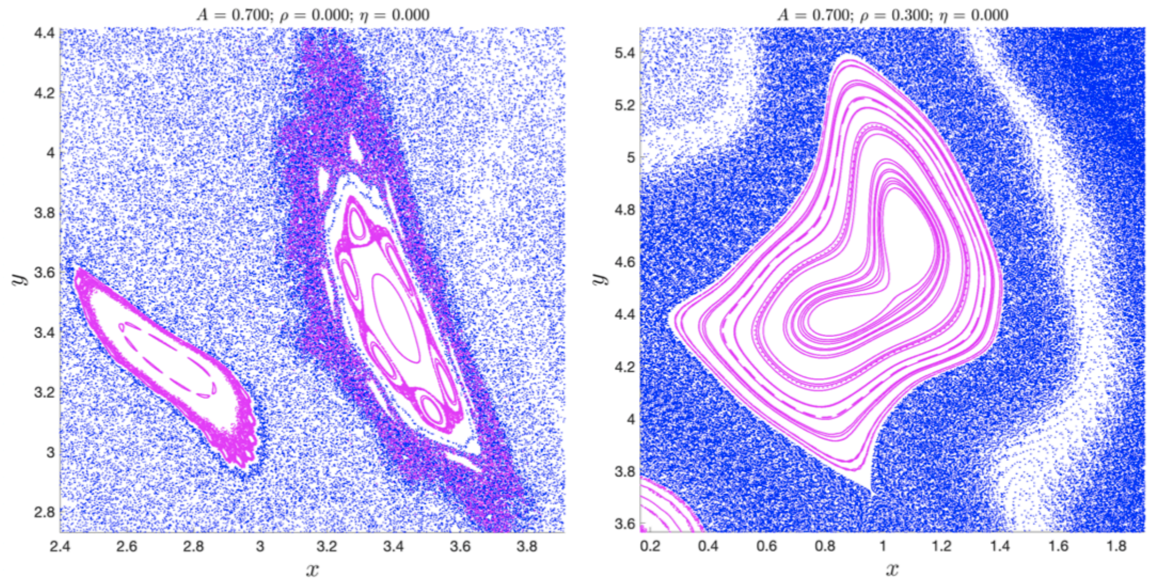


Figure 5.13: Zoomed-in versions of the non-modular stroboscopic plots of Fig. 5.8. The diffusive system (on the left) has more chaotic trajectories. White *highways* can be seen in the super-diffusive system.

5.3.2 Comparison between sub-diffusive and super-diffusive regimes associated with opposite values of η

During the analysis of the effect of η on diffusive regimes with A fixed, it emerged that around $\rho = 0.115$ increasing $|\eta|$ could lead to sub-diffusive ($\eta < 0$) as well as super-diffusive ($\eta > 0$) systems, before they eventually become diffusive as $|\eta|$ becomes significantly high. Given that this is rather unexpected, considering that this happens when the system is sub-diffusive for $\rho \simeq 0$, but not when it is diffusive or super-diffusive, it could be interesting to compare associated Poincaré sections and curves for *numerically* untrapped trajectories. In particular, the systems that have been explored are $(A, \rho, \eta) = (0.700, 0.115, -0.100)$ and $(A, \rho, \eta) = (0.700, 0.115, 0.100)$. The two cases can be compared using modular and non-modular stroboscopic plots shown in Figs. 5.14 and 5.15 (green trajectories have max. distances above 350 as for the cases with $\eta = 0$):

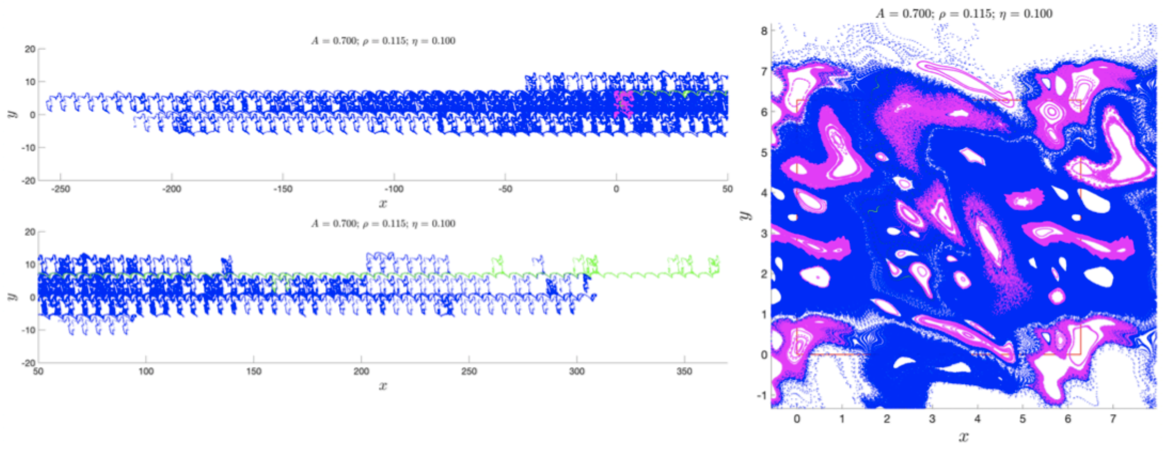


Figure 5.14: Non-modular stroboscopic plots of the super-diffusive system associated with $\eta > 0$ for the second simulation in `poincare` mode. The plot on the right is a zoomed-in version of the plot on the right, which for convenience has been split in two. Colors are used in the same way as in Fig. 5.8. White *highways* can be seen in the zoomed-in plot.

- The sub-diffusive system has a non-modular Poincaré section very similar to the left one in Fig. 5.8. This is normal given that diffusive and slightly sub-diffusive systems have very similar power-law exponents.

- The super-diffusive systems has a non-modular Poicaré section very similar to the right one in Fig. 5.8. The main differences are the fact that in this case the axes have smaller ranges and the horizontal direction is bolder in terms blue trajectories. Again, this is in line with having a lower value of b.

- The size of trapped trajectories is visually very similar, but the ones in the sub-diffusive system are more regular. It is probable that $\eta \simeq 0.100$ is the cause of a partial loss of regularity in the super-diffusive system, because $\rho > 0$ can not do that. The reason why this is not matched for $\eta \simeq -0.100$ is not clear, with the obvious answer being that changing the sign of η alters ψ in different ways. However, it can also be said that opposite values of η modify ψ in exactly opposite ways when $\varphi_{nm} = 0$ and $\rho = 0$. It could be that a combination of all these factors (random phases, Larmor radius, amplitude and η) is the cause for this loss of symmetry when moving from a plotting simplified non-turbulent potential to analyzing guiding center dynamics associated with actual forms of ψ;

- The number of trapped trajectories also seems to be more or less the same, with a good mix of islands with few and many concentric trajectories in both scenarios.

It seems that opposite values η does not have a huge impact on this particular aspect.

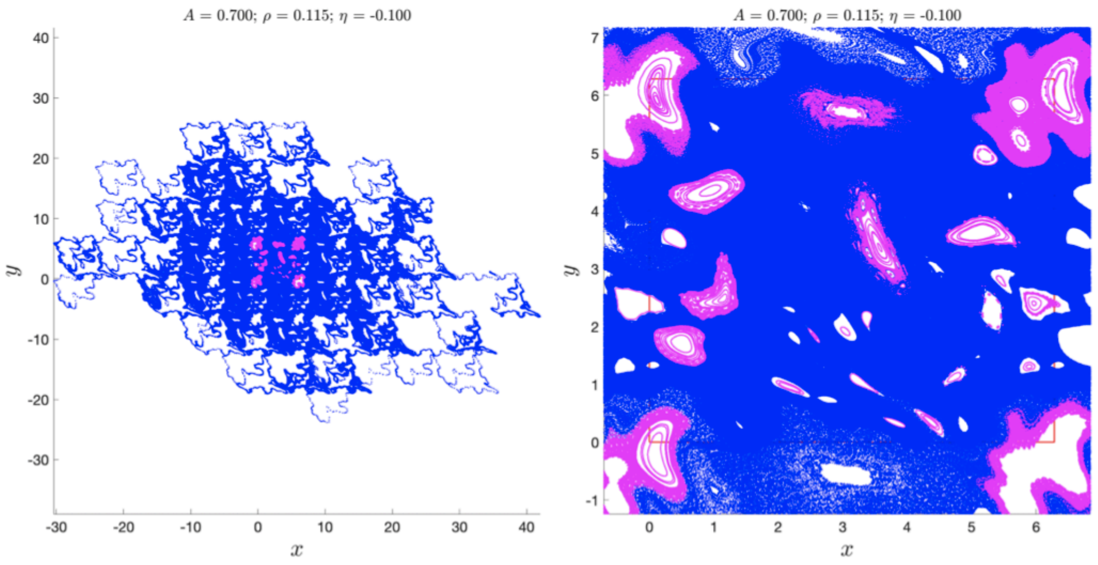


Figure 5.15: Non-modular stroboscopic plots of the sub-diffusive system associated with $\eta < 0$ for the second simulation in `poincare` mode. The plot on the right is a zoomed-in version of the plot on the right. Colors are used in the same way as in Fig. 5.8.

- by looking at the plot on the right of Fig. 5.14, hints of a familiar situation can be found. In fact, *highways* (and some green trajectories) can be detected. They are not present for the sub-diffusive system, as they were not present with diffusive regimes too. What is rather strange is the fact that, in this case, a super-diffusive regime is reached with the help of a term that acts on ψ more similarly to A than ρ, while in the case of $\eta = 0$ a much stronger condition was obtained by means of increasing ρ, i.e. a parameter that introduces a relaxation in the potential.

This last point is a natural connection to Fig. 5.16:

- The red curve on the left is somewhat reminiscent of the one for the diffusive case of the previous section. The range of thresholds is smaller than in the diffusive regime by almost a factor of 2, providing an indication of a sub-diffusive condition. The blue curve around the threshold value of 4 is not so regular, but a plateau is still present. The non-zero Larmor radius contributes to having the flat zone starting at around 80% of untrapped trajectories and also to having a loss in the smoothness of the entire curve (a lower degree of chaos is associated with sharper transitions).

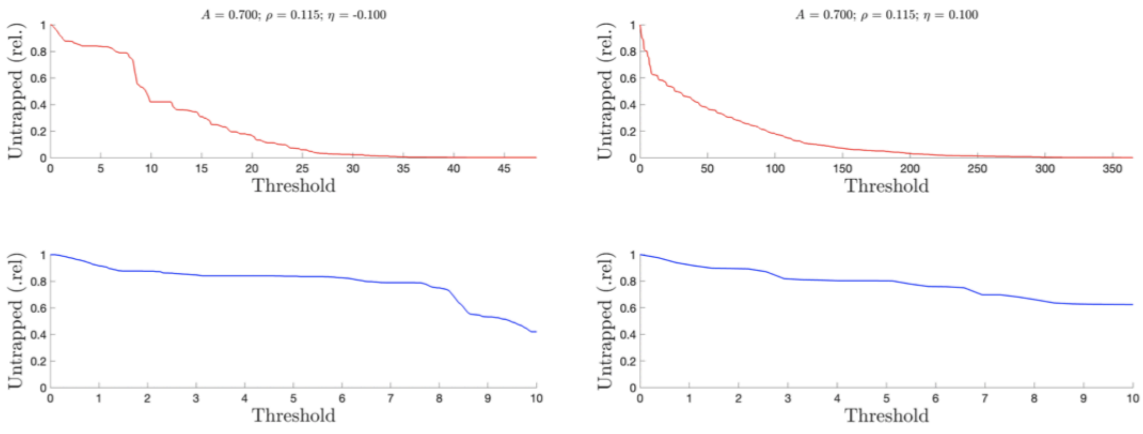


Figure 5.16: Visual comparison between the same systems of Figs. 5.14 and 5.15, but through the variation of the percentage of *numerically* untrapped trajectories as a function of the threshold value for max. distances. Flat zones can be spotted around the threshold value of 4 in both plots.

- The red curve on the right has some similarities with the one of the almost ballistic system of Fig. 5.12, but it also has at least one relevant difference. While it is true that the range of thresholds is more or less the same (slightly smaller here, in accordance with having a lighter super-diffusive system), it is evident that there is a much weaker transition between the descending first part and the slowly decaying second part of the red curve. From another perspective, the percentage of untrapped particles at which the first part stops is almost 60%, while before it was closer to 30%. As for the blue curve, it is very similar to the other one with $\eta = -0.100$, again suggesting that opposite values of η have more relevant influences on other aspects than the number and the sizes of trapped trajectories.
- In this case, red curves for opposite values of η show many more differences than modular stroboscopic plots. This observation is being made here to underline that it is often important to analyze systems by exploiting various kinds of outputs and ways of representing them.

5.4 Possible developments and future analysis

Many other simulations could be launched and analyzed to reach a better understanding of the system, for example:

- The influence of Bessel functions could be investigated by approximating them with their truncated series. This would require to change the values assigned to

`FLR`.

- Smoother `contourf` plots could be made by increasing the number of sub-simulations through `A`, `rho` and `eta` (while keeping the same ranges that have already been used).

- Analyzing other Poincaré sections and plots that display the percentage of *numerically* untrapped trajectories to find patterns or hidden aspects (such as *highways*).

- More precise data could be extracted with higher values of `Ntraj`, `Tf` and `Tmid`, together with a lower `TimeStep`. However, it should be kept in mind that simulations could be very slow because of the number of trajectories and/or instants used for integrating the equations. Moreover, increasing `Tf` too much could cause an unacceptable loss of accuracy.

- The effect of phases on super-diffusive systems could be tested by varying the argument of `numpy.random.seed`, given that their strboscopic plots tend to be highly developed only along one axis, whereas this is not the case for other diffusive regimes. This difference is unusual and unexpected.

- It should be important to check whether almost ballistic regimes have physical relevance or are just the result of some issues with how the problem was approached or its model was implemented and treated in Python.

An important aspect that has to be noticed is that nothing has been done here to check which nondimensionalized values of A ρ and η have corresponding values in real cases, e.g. it could be that using `eta = 0.100` in Python has no pratical applications even thought it seems to be a reasonable value. This is something that should be investigated case by case if these scripts (or similar versions) were to be used in contexts where all the physical information is known (nature of particles, measurements of the electricmagnetic field inside a tokamak, plasma temperature in the transverse plane and so on). In addition to using the existing tools to explore new aspects, there are also other ways to develop what has been presented in this report:

- To better model the real situation (now that some information about the main parameters has been collected), variations could be made to the Python files in order to implement a different magnetic field, either toroidal or helicoidal, and a toroidal geometry. This would require to find new equations for the guiding centers, which would certainly be more challenging to obtain than having a uniform and constant magnetic field in Cartesian coordinates. Fortunately, the steps to follow would pretty much be the same of Chapter 3 (in essence, changes of coordinates and elimination of fluctuating parts from low-order terms of the Hamiltonian).

- Even without exploring different geometries or magnetic fields, other models for similar or different electrostatic potentials could be used.

- Given that the second order term of the potential seems to have relevant influences on the behavior of the system, a similar approach could be used to include it in gyrokinetic codes. In order to *include the second term inside gyrokinetic codes*, the procedures would be much more complicated and the *second term* would be different from the one used in this work. For example, guiding centers would be substituted by gyrocenters and Maxwell's equations would be used to account for electromagnetic feedback. Moreover, the equations to be numerically implemented or modified would be those of statistical mechanics, that is equations that allow one to find distribution functions.

Chapter 6

Conclusions

With this work, which exploits guiding center theory, an existing simplified model of turbulent plasmas inside tokamaks, consisting in the 2D equations for guiding center dynamics, has been checked by re-deriving it from scratch. The equations were then characterized with an analytic electrostatic potential based on measurements made in real tokamaks. After that, they underwent a nondimensionalization procedure before being implemented in Python scripts for numerical simulations.
While the initial idea was to investigate the effect that the usually-neglected second order of the potential ψ had on the system, with η (i.e. the intensity of the second order term) being the main parameter in that sense, some additional aspects emerged during the work. In particular, unexpected cases of anomalous diffusion showed up, for some combinations of A (i.e. the amplitude of the potential) and ρ (the Lamor radius of charged particles), when simulations that involved only the first order of ψ were launched as feedback tools to improve the scripts. Feedback simulations allowed to properly set the values or the ranges of the main parameters of the simulations. Moreover, they opened up to many modifications to the original Python files that were not necessarily related to the second order of ψ, such as:

- The introduction of multicolored stroboscopic plots to distinguish between trapped and untrapped trajectories.
- The use of power-law curve fittings for the evolution of mean square displacements over time to account for anomalous diffusion.
- The implementation of a fast and reliable method to individuate trapped trajectories which is based on drawing rectangles around each trajectory and comparing their diagonals with a wisely chosen threshold value.
- The possibility to integrate the equations for guiding center dynamics in one or two steps. In the case of two steps, trapped particles are removed after an intermediate number of temporal cycles in order to save computational time (this is

possible because trapped guiding center manifest their nature very early and do not contribute to diffusion).

Most of the ideas behind these changes emerged from post-processing the Python outputs with MATLAB files written from scratch.
The main aspects that were highlighted by analyzing the outputs of the simulations with both Python and MATLAB are essentially the following:

- Different sets of A, ρ and/or η are associated with more chaotic or more regular behaviors of the trajectories of guiding centers. In this work, the focus was put on systems neither too regular (because they do not diffuse) nor too chaotic (because other approaches can suit them better).
- There can be differences in the behavior of the system (stroboscopic plots, diffusive regimes) if a finite Larmor radius is considered instead of using the approximation $\rho = 0$, for which particles and guiding centers are treated as if they were in the same places. This is an effect that can be detected even without considering the second order of ψ.
- There can be differences in the behavior of the system if ψ is associated with $\eta = 0$, $\eta > 0$ or $\eta < 0$. The tendency is that increasing $|\eta|$ ultimately leads to diffusive behaviors and higher levels of chaos.
- The systems can be diffusive, slightly sub-diffusive, super-diffusive or almost ballistic depending on the values of A, ρ and η. Increasing ρ can lead to almost ballistic regimes, with this effect that can be countered if A and/or η are sufficiently high. Power-law curve fittings are therefore better at describing the physical system associated with this work than linear regressions and diffusion coefficients (which were originally implemented in Python).
- The size of trapped trajectories on the transverse plane seems to be almost independent of the values assumed by A, ρ and η, in the sense that it is possible to utilize a unique and fixed value in the Python scripts to distinguish between untrapped and trapped guiding centers, with the latter not giving contribution to normal of anomalous diffusion. Such *unique and fixed value* was found during the simulations, is equal to 4 and has to be intended as the maximum possible distance between two points of the same trapped trajectory.

Among all, the facts that systems can have anomalous diffusion and that η seems to have relevant effects on the dynamics of guiding centers are the ones that should be further investigated to understand them better and to check if they can have implications in real applications. From an analytical point of view, it should be investigated why super-diffusive systems tend to have stroboscopic plots characterized by *highways* (i.e.

almost empty regions traveled by few untrapped guiding centers that are responsible for super-diffusive regimes) and developments of trajectories along only one axis, depending on the values assumed by the random phases of ψ and which are essential to reproduce turbulence. It could be, for example, that these aspects are closely related to the particular choice of the potential.
In any case, this work, even with all its limitations, opens up to future research in this field and has shown some interesting behaviors of modelled turbulent plasmas for pacific fusion applications.

Appendix A

Hamiltonian systems

This Appendix is based on [HS20] and [Cha18].

A.1 Hamilton's equations in canonical form

Charged particles can be treated in the framework of Hamiltonian systems, i.e. their dynamics are described with Hamilton's equations, which, considering a system with N degrees of freedom, have the following canonical form:

$$i = 1, ..., N : \begin{cases} \dfrac{dq_i}{dt} = \dfrac{\partial H}{\partial p_i}, \\ \\ \dfrac{dp_i}{dt} = -\dfrac{\partial H}{\partial q_i}, \end{cases} \tag{A.1}$$

where $(q_1(t), ...q_N(t))$ are the positions canonically conjugated with $(p_1(t), ..., p_N(t))$, which are the momenta. The canonical form of Hamilton's equations is therefore associated with these particular coordinates, which for convenience can be gathered to form a single vector:

$$\mathbf{z}(t) = (\mathbf{q}, \mathbf{p}) = (q_1, ..., q_N, p_1, ..., p_N). \tag{A.2}$$

The $2N$-dimensional phase space for Hamiltonian systems in canonical form is defined by $(\mathbf{q}, \mathbf{p})$. It is important to notice that, for example, in the case of a Hamiltonian system composed of a single free particle in 3D space, then there would be $N = 3$ degrees of freedom and therefore that particle would be associated with three positions and three momenta.

The relevant quantity that appears in Eqs. A.1 is the Hamiltonian $H(\mathbf{z})$, a scalar function which is closely related to the energy of the system that is being considered. Functions like the Hamiltonian are more properly called observables, and even each p_i and q_i fall into that category. Since these are independent of one another by definition, then the following equations featuring the Kronecker delta hold true:

$$i, j = 1, ..., 2N : \frac{\partial z_i}{\partial z_j} = \delta_{ij}. \tag{A.3}$$

In general, the evolution in time of any observable $F(\mathbf{z})$ can be expressed by exploiting the Hamiltonian:

$$\frac{dF}{dt} = \sum_{i=1}^{N} \left(\frac{\partial F}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial F}{\partial p_i} \frac{\partial H}{\partial q_i} \right). \tag{A.4}$$

A.2 Canonical Poisson brackets

It is simple to recover Hamilton's equations as a special case of Eq. A.4 by setting $F = z_i$ for $i = 1, ..., 2N$ and by using Eqs. A.3. A cleaner expression for Eq. A.4 is obtained with the Poisson bracket $\{\cdot, \cdot\}$, which is a bilinear operator acting on scalar functions and that is equipped with the following:

- Anti-symmetry property
$$\{F, G\} = -\{G, F\}. \tag{A.5}$$
- Leibniz rule
$$\{F, GH\} = \{F, G\}H + G\{F, H\}. \tag{A.6}$$
- Jacobi identity
$$\{\{F, G\}, H\} + \{\{H, F\}, G\} + \{\{G, H\}, F\} = 0. \tag{A.7}$$

F, G and H in Eqs. A.5-A.7 are just generic names given to scalar functions. When this operator is used, the time evolution of F is:

$$\frac{dF}{dt} = \sum_{i=1}^{N} \left(\frac{\partial F}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial F}{\partial p_i} \frac{\partial H}{\partial q_i} \right) = \{F, H\}. \tag{A.8}$$

By exploiting Einstein's notation for repeated indices it is possible to obtain the following:

$$\{F, H\} = \frac{\partial F}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial F}{\partial p_i} \frac{\partial H}{\partial q_i}. \tag{A.9}$$

Given that the sum in Eq. A.10 can be associated with the scalar product, another possibility would be to introduce vectors:

$$\{F, H\} = \frac{\partial F}{\partial \mathbf{q}} \cdot \frac{\partial H}{\partial \mathbf{p}} - \frac{\partial F}{\partial \mathbf{p}} \cdot \frac{\partial H}{\partial \mathbf{q}}. \tag{A.10}$$

Whenever a Poisson bracket has canonical coordinates as its variables, it is said to be canonical. Hamilton's canonical equation can be also expressed in the Poisson bracket form:

$$i = 1, ..., N : \begin{cases} \dfrac{dq_i}{dt} = \{q_i, H\}, \\ \\ \dfrac{dp_i}{dt} = \{p_i, H\}. \end{cases} \tag{A.11}$$

A.3 Time-dependent Hamiltonian systems

Up to this point, the Hamiltonian has always been intended as autonomous, i.e. a function that at most depends implicitly on time through $\mathbf{z}(t)$. While this is the case for many physical systems, there are however also ones whose corresponding Hamiltonians happen to have an explicit time dependence:

$$H = H(\mathbf{z}(t), t). \tag{A.12}$$

These systems are the ones that have an external force acting on them, e.g. plasmas confined by magnetic fields in fusion devices. Non-autonomous systems can be harder to deal with and for this reason it can be highly convenient to autonomize them. In order to do that, a new variable h is introduced as the coordinate canonically conjugated with time. The new Hamiltonian $\mathcal{H}$ is related to the old one H in this way:

$$\mathcal{H}(\mathbf{z}, t, h) = H(\mathbf{z}, t) + h. \tag{A.13}$$

The newly autonomized Hamiltonian system is associated with observables that in general depend on both t and h (other than $\mathbf{z}$). They have an extended Poisson bracket $\{\cdot, \cdot\}_\tau$ which is slightly different from the one already introduced:

$$\{\mathcal{F}, \mathcal{G}\}_\tau = \{\mathcal{F}, \mathcal{G}\} + \frac{\partial \mathcal{F}}{\partial t}\frac{\partial \mathcal{G}}{\partial h} - \frac{\partial \mathcal{F}}{\partial h}\frac{\partial \mathcal{G}}{\partial t}, \tag{A.14}$$

where $\mathcal{F} = \mathcal{F}(\mathbf{z}, t, h)$ and $\mathcal{G} = \mathcal{G}(\mathbf{z}, t, h)$. In this case it could be helpful to modify the notation to obtain a more compact and cleaner version of the extended Poisson bracket. In particular, it is obtained by making the variables associated with each portion of the whole bracket explicit:

$$\{\mathcal{F}, \mathcal{G}\} = \frac{\partial \mathcal{F}}{\partial \mathbf{q}} \cdot \frac{\partial \mathcal{G}}{\partial \mathbf{p}} - \frac{\partial \mathcal{F}}{\partial \mathbf{p}} \cdot \frac{\partial \mathcal{G}}{\partial \mathbf{q}} \longrightarrow \{\mathcal{F}, \mathcal{G}\}_{(\mathbf{q},\mathbf{p})}, \tag{A.15}$$

$$\frac{\partial \mathcal{F}}{\partial t}\frac{\partial \mathcal{G}}{\partial h} - \frac{\partial \mathcal{F}}{\partial h}\frac{\partial \mathcal{G}}{\partial t} \longrightarrow \{\mathcal{F}, \mathcal{G}\}_{(t,h)}, \tag{A.16}$$

$$\{\mathcal{F},\mathcal{G}\}_\tau = \{\mathcal{F},\mathcal{G}\}_{(\mathbf{q},\mathbf{p})} + \{\mathcal{F},\mathcal{G}\}_{(t,h)}. \tag{A.17}$$

An extendend Poisson bracket which derives from a canonical Poisson bracket, like the one in Eq. A.14, is called extended canonical Poisson bracket. Moreover, the extended phase space for the autonomized Hamiltonian system is defined by:

$$\mathbf{z}_\tau = (\mathbf{z}, t, h) = (\mathbf{q}, \mathbf{p}, t, h) \tag{A.18}$$

A.4 Changes of coordinates

The canonical Poisson bracket for an autonomous Hamiltonian system in canonical form with N degrees of freedom can be also expressed, in compact way, with the introduction of the symplectic (i.e. nonsingular skew-symmetric) $2N \times 2N$ matrix $\mathcal{J}$:

$$\mathcal{J} = \begin{pmatrix} \mathcal{O}_N & \mathcal{I}_N \\ -\mathcal{I}_N & \mathcal{O}_N \end{pmatrix}, \tag{A.19}$$

$$\{F, G\} = \sum_{k,l=1}^{2N} \frac{\partial F}{\partial z_k} \mathcal{J}_{kl} \frac{\partial G}{\partial z_l} = \frac{\partial F}{\partial \mathbf{z}} \cdot \mathcal{J} \left(\frac{\partial G}{\partial \mathbf{z}} \right)^{\mathrm{T}}, \tag{A.20}$$

where $\mathcal{I}_N$ is the $N \times N$ identity matrix, $\mathcal{O}_N$ is the null $N \times N$ matrix and the apex T indicates that the row vector $\partial G/\partial \mathbf{z}$ has to be transposed before operating the multiplication with $\mathcal{J}$. This is required to have compatible vectors so the the scalar product can be properly done.
To clarify:

- $\partial F/\partial \mathbf{z}$ is a row vector of length $2N$.
- $(\partial G/\partial \mathbf{z})^{\mathrm{T}}$ is a column vector of length $2N$.
- $\mathcal{J}(\partial G/\partial \mathbf{z})^{\mathrm{T}}$ is a column vector of length $2N$.

Actually, Eq. A.20 is only a special case of a more general formula which allows one to write the Poisson bracket for any Hamiltonian system given any set of coordinates. In fact, it is not mandatory that canonical coordinates have to be used to describe a Hamiltonian system: other coordinates can be used, depending maybe on what the purposes are or what suits better in a certain situation. It happens very often that physical systems are described with sets of variables which are very different from $(\mathbf{q}, \mathbf{p})$. However, it is always possible, in a Hamiltonian framework, to write the chosen coordinates in terms of the canonical ones. It could be the case that, for example, the final set that is being used is the result of a subsequent series of changes of coordinates, with this chain starting from $(\mathbf{q}, \mathbf{p})$.
Moreover, regardless of the changes, it is possible to autonomize time-dependent systems

even at later stages, which means that it is not necessary to immediately add h to $(\mathbf{z}(t), t)$ before performing the various transformations leading to the final set.
Going back to Eq. A.20, the generalized version $\mathcal{J}$ is the square matrix that accounts for the change of coordinates, meaning that it contains the information linking the old and the new coordinates. In particular, each element of the generalized version of $\mathcal{J}$ is a Poisson bracket that acts on the new coordinates and has the old ones as its variables. The value of the generalized version of $\mathcal{J}$ can be found, in practice, if the chain of transformations between the sets of coordinates is known. In order to show how, it is easier to first write the generalized version of Eq. A.20 and then start the reasoning from there. Given a generic set of M coordinates $\mathbf{z}_\alpha = (z_1, ..., z_M)$ that forms the phase space for a Hamiltonian system, and given two generic observables $F(\mathbf{z}_\alpha)$ and $G(\mathbf{z}_\alpha)$, then:

$$\{F, G\} = \sum_{k,l=1}^{M} \frac{\partial F}{\partial z_{\alpha_k}} \mathcal{J}_{kl}(\mathbf{z}_\alpha) \frac{\partial G}{\partial z_{\alpha_l}} = \frac{\partial F}{\partial \mathbf{z}_\alpha} \cdot \mathcal{J}(\mathbf{z}_\alpha) \left(\frac{\partial G}{\partial \mathbf{z}_\alpha} \right)^{\mathrm{T}}, \tag{A.21}$$

$$k, l = 1, .., M : \mathcal{J}_{kl}(\mathbf{z}_\alpha) = \{z_{\alpha_k}, z_{\alpha_l}\}. \tag{A.22}$$

In true essence, finding the value of the left-hand side in Eq. A.21 relies on being able to evaluate the terms in Eqs. A.22, because the rest is much easier. If, for example, the considered Hamiltonian system has N degrees of freedom and its coordinates are directly linked to the canonical ones, in the sense that only the change

$$\mathbf{z} = (\mathbf{q}, \mathbf{p}) \mapsto \mathbf{z_0}(\mathbf{z}) = \mathbf{z_0} = (z_{0_1}, ..., z_{0_{2N}}) \tag{A.23}$$

has been performed, then it is possible to determine $\{z_{0_k}, z_{0_l}\} = \{z_{0_k}(\mathbf{z}), z_{0_l}(\mathbf{z})\}$ by exploitation of Eq. A.19 thanks to $\mathcal{J}(\mathbf{z})$ being exactly $\mathcal{J}$. This fact comes from Eqs. A.3, because all the elements of $\mathcal{J}(\mathbf{z})$ are canonical Poisson brackets acting on observables that are the canonical coordinates themselves. In other words, this is the special case that allows one to recover Eq. A.20 from Eq. A.21.
The elements of $\mathcal{J}(\mathbf{z_0})$ are then evaluated by applying A.21 with $\{F, G\} = \{z_{0_k}, z_{0_l}\}$ (for $k, l = 1, ..., 2N$) and $\mathbf{z}_\alpha = \mathbf{z}$:

$$k, l = 1, ..., 2N : \mathcal{J}_{kl}(\mathbf{z_0}) = \{z_{0_k}, z_{0_l}\} = \frac{\partial z_{0_k}}{\partial \mathbf{z}} \cdot \mathcal{J}(\mathbf{z}) \left(\frac{\partial z_{0_l}}{\partial \mathbf{z}} \right)^{\mathrm{T}} = \frac{\partial z_{0_k}}{\partial \mathbf{z}} \cdot \mathcal{J} \left(\frac{\partial z_{0_l}}{\partial \mathbf{z}} \right)^{\mathrm{T}}. \tag{A.24}$$

It is important to notice that the change of coordinates A.23 can also be viewed as a *wrapping* change of coordinates which takes into account every intermediate transformation that could take place when a physical system is being studied. However, if multiple steps are considered or there is the intention to keep them well distinguished for some reason, then the procedure is very similar to the one just shown. In particular, if n changes of coordinates

$$\mathbf{z} = \mathbf{z_0} \mapsto \mathbf{z_1} \mapsto ... \mapsto \mathbf{z_k} ... \mapsto ... \mapsto \mathbf{z_n} \tag{A.25}$$

have been performed starting from $2N$ canonical coordinates, then, following the example where $n = 1$ and given that $\mathbf{z_n} = \mathbf{z_n}(\mathbf{z_{n-1}})$, $\mathcal{J}(\mathbf{z_n})$ would require $\mathcal{J}(\mathbf{z_{n-1}})$ to become a known matrix:

$$k, l = 1, ..., 2N : \mathcal{J}_{kl}(\mathbf{z_n}) = \{z_{n_k}, z_{n_l}\} = \frac{\partial z_{n_k}}{\partial \mathbf{z_{n-1}}} \cdot \mathcal{J}(\mathbf{z_{n-1}}) \left(\frac{\partial z_{n_l}}{\partial \mathbf{z_{n-1}}} \right)^{\mathrm{T}}. \tag{A.26}$$

In turn, $\mathcal{J}(\mathbf{z_{n-1}})$ would then need $\mathcal{J}(\mathbf{z_{n-2}})$ to be determined and this *backward pairing* pattern would continue up to the couple formed by $\mathcal{J}(\mathbf{z_1})$ and $\mathcal{J}(\mathbf{z_0})$. The general recursive formula, for $s = 0, ..., n-1$, is:

$$k, l = 1, ..., 2N : \mathcal{J}_{ij}(\mathbf{z_{s+1}}) = \{z_{s+1_k}, z_{s+1_l}\} = \frac{\partial z_{s+1_k}}{\partial \mathbf{z_k}} \cdot \mathcal{J}(\mathbf{z_k}) \left(\frac{\partial z_{s+1_l}}{\partial \mathbf{z_k}} \right)^{\mathrm{T}}. \tag{A.27}$$

However, since $\mathcal{J}(\mathbf{z_0}) = \mathcal{J}(\mathbf{z}) = \mathcal{J}$ is given by Eq. A.19, it is now evident that it is possible to evaluate each possible $\mathcal{J}(\mathbf{z_{s+1}})$. Summing up, in a Hamiltonian framework, once the various changes and $\mathcal{J}$ for the canonical coordinates are known, it is possible to determine any Poisson Bracket for whatever system of coordinates.

A.5 Lie transform

A particular class of changes of coordinates is represented by the Lie transform, which has the effect of mapping a generic set of coordinates $\boldsymbol{z}$ into a new set $\bar{\boldsymbol{z}}$ without modifying the form of Poisson brackets. All the transformations of this kind are said to be canonical and in essence they do not require Eq. A.27.
Lie transforms are associated with a numeric parameter $\varepsilon \in \mathbb{R}$ such that $\bar{\boldsymbol{z}}$ is equal to $\boldsymbol{z}$ when $\varepsilon = 0$. In symbols:

$$\boldsymbol{z} \mapsto \bar{\boldsymbol{z}}(\boldsymbol{z}, \varepsilon) : \bar{\boldsymbol{z}}(\boldsymbol{z}, 0) = \boldsymbol{z}. \tag{A.28}$$

The expression for the Lie transform that can be actively used to concretize the change of coordinates is obtained by solving the following differential equation:

$$\frac{d\bar{\boldsymbol{z}}}{d\varepsilon} = \{S, \bar{\boldsymbol{z}}\}, \tag{A.29}$$

where $S = S(\boldsymbol{z})$ is the generating function of the Lie transform. The Poisson bracket in Eq. A.29 can be rewritten with the introduction of the Liouville operator $\mathcal{L}_S$:

$$\{S, \bar{\boldsymbol{z}}\} = \mathcal{L}_S \bar{\boldsymbol{z}}. \tag{A.30}$$

The solution of Eq. A.29 can be expressed in terms of a series similar to the Taylor one for the exponential function:

$$\mathrm{e}^x = \sum_{k=0}^{\infty} \frac{x^n}{n!}, \tag{A.31}$$

$$\frac{d\bar{\boldsymbol{z}}}{d\varepsilon} = \mathcal{L}_S \bar{\boldsymbol{z}} \longrightarrow \bar{\boldsymbol{z}}(\boldsymbol{z}, \varepsilon) = \mathrm{e}^{\varepsilon \mathcal{L}_S} \bar{\boldsymbol{z}}(\boldsymbol{z}, 0) = \mathrm{e}^{\varepsilon \mathcal{L}_S} \boldsymbol{z} = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} \mathcal{L}_S^n \boldsymbol{z}. \tag{A.32}$$

In expanded form, the series in Eq. A.32 is:

$$\sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} \mathcal{L}_S^n \boldsymbol{z} = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} \{S, \{S, \{..., \{S, \mathbf{z}\}...\}\}\}, \tag{A.33}$$

where n-th term has n nested Poisson brackets. In particular, this means that the term for $n = 0$ reduces to only $\boldsymbol{z}$: there are zero Poisson brackets and also $\varepsilon^0 = 0! = 1$. In short, the change of coordinates associated with the Lie transform is:

$$\boldsymbol{z} \mapsto \bar{\boldsymbol{z}} = \mathrm{e}^{\varepsilon \mathcal{L}_S} \boldsymbol{z}. \tag{A.34}$$

The inverse of this transformation is basically obtained by changing ε into $-\varepsilon$:

$$\bar{\boldsymbol{z}} \mapsto \boldsymbol{z} = \mathrm{e}^{-\varepsilon \mathcal{L}_S} \bar{\boldsymbol{z}}. \tag{A.35}$$

If now a generic observable of the starting coordinates $F(\boldsymbol{z})$ is taken into account, it is possible to obtain a new observable which is function of the new coordinates, namely $\bar{F}(\bar{\boldsymbol{z}})$:

$$F(\boldsymbol{z}) = F(\mathrm{e}^{-\varepsilon \mathcal{L}_S} \bar{\boldsymbol{z}}) = \mathrm{e}^{-\varepsilon \mathcal{L}_S} F(\bar{\boldsymbol{z}}) = \bar{F}(\bar{\boldsymbol{z}}). \tag{A.36}$$

Use of the property

$$\beta(\mathrm{e}^{\varepsilon \mathcal{L}_S} G) = \mathrm{e}^{\varepsilon \mathcal{L}_S} \beta(G), \tag{A.37}$$

which is valid for any scalar function β and observable G, has been made here. It can be noticed that this property can suit vectors in the sense that it is applied element by element. Anyway, Eq. A.36 states that F and $\bar{F}$ have the same value when their arguments are $\boldsymbol{z}$ and $\bar{\boldsymbol{z}}$, respectively. Besides Eq. A.37, there are two other important properties of the Lie transform:

- It is a linear operator, i.e. for any number N of constants α_i and scalar functions f_i $(i = 1, ..., N)$ it is possible to write

$$\mathrm{e}^{\varepsilon \mathcal{L}_S} \left(\sum_{i=1}^{N} \alpha_i f_i \right) = \sum_{i=1}^{N} \alpha_i \mathrm{e}^{\varepsilon \mathcal{L}_S} f_i. \tag{A.38}$$

- Given any two observables F and G, it satisfies the relation

$$\{\mathrm{e}^{\varepsilon \mathcal{L}_S} F, \mathrm{e}^{\varepsilon \mathcal{L}_S} G\} = \mathrm{e}^{\varepsilon \mathcal{L}_S} \{F, G\}. \tag{A.39}$$

Appendix B

Discrete and Fast Fourier Transforms

This Appendix is based on [BK22], [Joh11] and [com21].

B.1 Discrete Fourier Transform

Any function $\mathrm{a}(x)$ that is periodic with period $\mathcal{P}$ in $[x_0, x_0+\mathcal{P}[$ can be expressed through a Fourier series:

$$\begin{cases} \mathrm{a}(x) = \displaystyle\sum_{k=-\infty}^{\infty} \mathrm{A}_k \mathrm{e}^{\mathrm{j}kx}, \\ \\ k = -\infty, ..., 0, ..., +\infty : \mathrm{A}_k = \dfrac{1}{\mathcal{P}} \displaystyle\int_{x_0}^{x_0+\mathcal{P}} \mathrm{a}(x)\mathrm{e}^{-\frac{2\pi \mathrm{j}kx}{\mathcal{P}}} dx. \end{cases} \tag{B.1}$$

The numerical implementation of $\mathrm{a}(x)$ that exploits its periodic nature is associated with a discretization of its domain and the construction of a truncated series. If N points are chosen for the discretization, then vectors of length N are obtained:

$$\boldsymbol{x} = (x_0, ..., x_{N-1})^{\mathrm{T}}, \tag{B.2}$$

$$\boldsymbol{a} = (a_0, ..., a_{N-1})^{\mathrm{T}}, \tag{B.3}$$

$$\boldsymbol{A} = (A_0, ..., A_{N-1})^{\mathrm{T}}, \tag{B.4}$$

$$x_n = \frac{n\mathcal{P}}{N}, \tag{B.5}$$

$$\mathrm{a}(x_n) \simeq a_n = \frac{1}{N} \sum_{k=0}^{N-1} A_k \mathrm{e}^{+\frac{2\pi \mathrm{j}nk}{N}}, \tag{B.6}$$

$$A_k = \sum_{n=0}^{N-1} a_n \mathrm{e}^{-\frac{2\pi \mathrm{j} nk}{N}}. \tag{B.7}$$

Eq. B.7 is expressing an inverse Discrete Fourier Transform (DFT^{-1}). On the other hand, Eq. B.7 is a (direct) Discrete Fourier Transform (DFT). $\boldsymbol{A}$ is the vector of Fourier coefficients. The numerical evaluation of the elements of $\boldsymbol{A}$ starting from those of $\boldsymbol{a}$ (and viceversa) is usually done with the help of two high-performing algorithms: the Fast Fourier Transform (FFT) and its inverse (FFT^1). The main concepts behind them are described in section B.2.
Naturally, it is possible to extend the DFT and its calculation with the $FFT$to dimensions beyond 1. For example, the DFT and its inverse in two dimensions involve matrices instead of vectors. Their elements, in the case of an equal discretization along both directions x and y, are given by:

$$A_{kl} = \sum_{n,m=0}^{N-1} a_{nm} \mathrm{e}^{-\frac{2\pi \mathrm{j}(nk+ml)}{N}}, \tag{B.8}$$

$$\mathrm{a}(x_n, y_m) \simeq a_{nm} = \frac{1}{N^2} \sum_{k,l=0}^{N-1} A_{kl} \mathrm{e}^{+\frac{2\pi \mathrm{j}(nk+ml)}{N}}, \tag{B.9}$$

where the generic element y_m is obviously part of a vector similar to $\boldsymbol{x}$:

$$\boldsymbol{y} = (y_0, ..., y_{N-1})^{\mathrm{T}}, \tag{B.10}$$

$$y_n = \frac{n\mathcal{P}}{N}. \tag{B.11}$$

In the special case of $\mathcal{P} = 2\pi$, there is a considerable simplification:

$$A_{kl} = \sum_{n,m=0}^{N-1} a_{nm} \mathrm{e}^{-\mathrm{j}(kx_n + ly_m)}, \tag{B.12}$$

$$\mathrm{a}(x_n, y_m) \simeq a_{nm} = \frac{1}{N^2} \sum_{k,l=0}^{N-1} A_{kl} \mathrm{e}^{\mathrm{j}(kx_n + ly_m)}. \tag{B.13}$$

B.2 Fast Fourier Transform algorithm

The problem with the DFT and its inverse is that they can be significantly slow and expensive to evaluate with a computer when $N \gg 1$. For example, the monodimensional DFT has a quadratic scaling with N, meaning that it involves $O(N^2)$ operations when evaluated by means of matrix multiplication:

$$\boldsymbol{A} = \mathcal{W}_N \boldsymbol{a}, \tag{B.14}$$

$$u, v = 0, ..., N-1 : \mathcal{W}_{N_{uv}} = \omega_N^{uv} = (\mathrm{e}^{-2\pi \mathrm{j}/N})^{uv} = \mathrm{e}^{-2\pi \mathrm{j} uv/N}. \tag{B.15}$$

A solution to this problem is represented by the Fast Fourier Transform algorithm (FFT) and its inverse (FFT^1). As its name suggests, the FFT is much faster and efficient than the DFT, while still getting to the same result. For example, the 1D version takes only $O(N \log_2(N))$ operations instead of $O(N^2)$. Without going into all the details of how the FFT algorithm can be coded, on a base level it exploits the fact that there is a way of efficiently rearranging the DFT matrix (like $\mathcal{W}$) and the quantity it is multiplying (like **a**), when N is a power of 2. For example, if $N = 2^p$, with p being a positive natural so that $N \gg 1$, then the right-hand side of Eq. B.14 can be written as the product of two matrices and a differently stacked version of **a**, which is obtained by putting elements with even indices on top of the remainings:

$$\mathcal{W}_N \boldsymbol{a} = \underbrace{\begin{pmatrix} \mathcal{I}_{N/2} & \mathcal{D}_{N/2} \\ \mathcal{I}_{N/2} & -\mathcal{D}_{N/2} \end{pmatrix} \begin{pmatrix} \mathcal{W}_{N/2} & \mathcal{O}_{N/2} \\ \mathcal{O}_{N/2} & \mathcal{W}_{N/2} \end{pmatrix}}_{\text{factorized } \mathcal{W}_N} \underbrace{\begin{pmatrix} \boldsymbol{a}^{(\mathbf{even})} \\ \boldsymbol{a}^{(\mathbf{odd})} \end{pmatrix}}_{\text{rearranged } \mathbf{a}}, \tag{B.16}$$

where

$$u, v = 0, ..., N/2-1 : \mathcal{D}_{N/2_{uv}} = \omega_N^u \delta_{uv} = \mathrm{e}^{-2\pi \mathrm{j} u/N} \delta_{uv}, \tag{B.17}$$

$$\boldsymbol{a}^{(\mathbf{even})} = (a_0, a_2, ..., a_{N-4}, a_{N-2})^{\mathrm{T}}, \tag{B.18}$$

$$\boldsymbol{a}^{(\mathbf{odd})} = (a_1, a_3, ..., a_{N-3}, a_{N-1})^{\mathrm{T}}. \tag{B.19}$$

As it can be clearly seen, the first of the two matrices in the right-hand side of Eq. B.16 involve identity and diagonal matrices of size $N/2 \times N/2$, while the second one contains null matrices and smaller versions of the original DFT matrix $\mathcal{W}_N$. The power of this new arrangement lies in two facts:

- The first matrix involves four diagonal sub-matrices, which in general make computations much easier.
- The second matrix is half as costly as the original dense $\mathcal{W}_N$ since half of its coefficients are zeros.

The procedure that led to Eq. B.16 can be the reiterated multiple times to further speed up the computing of the DFT. The repetition of the first rearrangement step is what lies behind the FFT algorithm:

- $\mathcal{W}_{N/2}$ can be associated with a similar expression to Eq. B.16. This involves obtaining $\mathcal{W}_{N/4}$, which following the same reasoning can be related to $\mathcal{W}_{N/8}$ and so on up to the point of getting $\mathcal{W}_2$, which is a simple 2×2 matrix:

$$\mathcal{W}_{N/2} = \begin{pmatrix} \mathcal{I}_{N/4} & \mathcal{D}_{N/4} \\ \mathcal{I}_{N/4} & -\mathcal{D}_{N/4} \end{pmatrix} \begin{pmatrix} \mathcal{W}_{N/4} & \mathcal{O}_{N/4} \\ \mathcal{O}_{N/4} & \mathcal{W}_{N/4} \end{pmatrix}, \tag{B.20}$$

$$\downarrow$$

$$\mathcal{W}_{N/4} = \begin{pmatrix} \mathcal{I}_{N/8} & \mathcal{D}_{N/8} \\ \mathcal{I}_{N/8} & -\mathcal{D}_{N/8} \end{pmatrix} \begin{pmatrix} \mathcal{W}_{N/8} & \mathcal{O}_{N/8} \\ \mathcal{O}_{N/8} & \mathcal{W}_{N/8} \end{pmatrix}, \tag{B.21}$$

$$\downarrow$$

$$\cdots$$

$$\downarrow$$

$$\mathcal{W}_4 = \begin{pmatrix} \mathcal{I}_2 & \mathcal{D}_2 \\ \mathcal{I}_2 & -\mathcal{D}_2 \end{pmatrix} \begin{pmatrix} \mathcal{W}_2 & \mathcal{O}_2 \\ \mathcal{O}_2 & \mathcal{W}_2 \end{pmatrix}. \tag{B.22}$$

- The rearranged version of **a** can be futher mixed in similar way to Eq. B.16. In particular, after having reindexed all its elements in order from 0 to $N-1$, the ones that are also part of $\boldsymbol{a}^{\mathbf{(even)}}$ are rearranged in such a way that the even-indexed ones are put on top of the others. Then, the same modification is made for the elements that are also part of $\boldsymbol{a}^{\mathbf{(odd)}}$. This exact procedure, which involved one rearrangement the first time (Eq. B.16) and has two rearrangements now, can then be repeated for a number of times that matches the iterations executed starting from $\mathcal{W}_N$. It can be verified that each subsequent modification of the original **a** involves $2^{(k-1)}$ restacking operations, where k stands for the generic k-th iteration of the process.

The requirement of having N as a power of 2 does not represent a true constriction: if N is not like that, then it can be increased to meet this condition in such a way that the results for the original N and the the matching portion in the modified one are the same. This can be achieved by padding with zeros the vector or the matrix that undergoes direct or inverse transformation. Even if it may seem that padding goes against computation time, it has to be kept in mind that the number of operations depends on a logarithmic factor: increasing N has therefore a very limited and decreasingly relevant effect on the number of operations themselves.
When the FFT algorithm is used to evaluate the DFT in numerical computations, $\boldsymbol{a}$ and $\mathbf{A}$ are square matrices that can be linked using the following compact notations:

$$\boldsymbol{A} = \mathrm{FFT}(\boldsymbol{a}), \tag{B.23}$$

$$\boldsymbol{a} = \mathrm{FFT}^{-1}(\boldsymbol{A}). \tag{B.24}$$

In two dimensions:

$$\boldsymbol{A} = \mathrm{FFT}_2(\boldsymbol{a}), \tag{B.25}$$

$$\boldsymbol{a} = \mathrm{FFT}_2^{-1}(\boldsymbol{A}). \tag{B.26}$$

However, approximating quantities like derivatives, e.g. $d\mathrm{a}(x)/dx$, is a little bit trickier.

B.3 Computing first order derivatives with the *FFT* algorithm

The Fast Fourier Transform can be very useful to efficiently approximate derivatives of functions that can be expressed through Fourier series.
The actual first derivative of $\mathrm{a}(x)$ is given by:

$$\frac{d}{dx}\mathrm{a}(x) = \sum_{k=-\infty}^{\infty} \left(\frac{2\pi \mathrm{j}}{\mathcal{P}} k \mathrm{A}_k\right) \mathrm{e}^{\frac{2\pi \mathrm{j}}{\mathcal{P}} kx} \tag{B.27}$$

When dealing with Discrete Fourier Transforms, the expression is similar, but, in order to fully characterize it, a *trigonometric interpolation* has to be defined. More about it can be found in [Joh11]. The associated procedure leads to a formula that allows one to approximate $d\mathrm{a}(x)/dx$ at the sample points gathered in $\boldsymbol{x}$:

$$\left.\frac{d\mathrm{a}(x)}{dx}\right|_{x_n} \simeq a'_n = \sum_{0<k<N/2} \frac{2\pi \mathrm{j}}{\mathcal{P}} k \left(A_k \mathrm{e}^{\frac{2\pi \mathrm{j}}{N} nk} - A_{N-k} \mathrm{e}^{-\frac{2\pi \mathrm{j}}{N} nk}\right) = \sum_{k=0}^{N-1} A'_k \mathrm{e}^{\frac{2\pi \mathrm{j}}{N} nk}; \tag{B.28}$$

where a'_n is the n-th element of the vector of approximates first derivatives and

- $A'_k = 0$ for $k = \dfrac{N}{2}$, if N is even. In fact, if N is odd, then k, being an integer, can not be equal to half of N.
- $A'_k = \dfrac{2\pi \mathrm{j}}{\mathcal{P}} k A_k$ for $0 \leq k < \dfrac{N}{2}$.
- $A'_k = -\dfrac{2\pi \mathrm{j}}{\mathcal{P}} (N-k) A_k$ for $k > \dfrac{N}{2}$.

The notation $0 < k < N/2$ has been used to take into account that N can be even or odd. If N is even, then that notation means $k = 1, ..., N/2$, whereas if N is odd, then $k = 1, ..., (N-1)/2$ (in the second sum of Eq. B.28, k always goes from 0 to $N-1$).
Fourier coefficients for the first derivative are more easily defined by using a rearranged vector $\boldsymbol{\kappa}$ of length N (this is especially helpful when coding):

- If N is even

$$\boldsymbol{\kappa} = \frac{2\pi}{\mathcal{P}}\left(0, ..., \frac{N}{2} - 1, -\frac{N}{2}, ..., -1\right), \tag{B.29}$$

$$A'_k = \mathrm{j}\kappa_k A_k. \tag{B.30}$$

- If N is odd

$$\boldsymbol{\kappa} = \frac{2\pi}{\mathcal{P}}\left(0, ..., \frac{N-1}{2}, -\frac{N-1}{2}, ..., -1\right), \tag{B.31}$$

$$A'_k = \mathrm{j}\kappa_k A_k. \tag{B.32}$$

Using a direct Fast Fourier Transform acting on the vector of a'_n can be used to calculate the vector of A_k.
Then, using an inverse Fast Fourier Transform acting on the vector of A'_k can be used to obtain a good estimate of da/dx. Because this is a numerical process that involves Fourier space, it is common that using the inverse FFT for derivatives produces complex numbers (with very small imaginary parts) instead of real numbers. This is why the real part is explicitly taken when actual implementations of inverse Fast Fourier Transforms are made in scripts.
A very similar procedure is applied in the case of functions of two or more variables.

B.3.1 Implementation in Python

```python
import numpy as xp
from numpy.fft import fft, ifft, fftfreq
N = 2**10
P = 2*xp.pi
x = xp.linspace(0, 2*xp.pi, N, endpoint=False)
kappa = (2*xp.pi/P)*fftfreq(N, d=1/N)
a = xp.sin(x)
A = fft(a)
A_prime = 1j*kappa*A
a_prime = (ifft(A_prime)).real
```

Listing 12: Simple Python script that shows how to set up and use direct and inverse Fast Fourier Transforms to compute derivatives. In this case, the use of `numpy.fft.fftfreq` with `d=1/N` creates a vector $\boldsymbol{\kappa}$ like in Eq. B.29. `a_prime` (i.e. $\boldsymbol{a'}$) is the vector that approximates the first derivative of `a` (i.e. $\boldsymbol{a}$), while `A` (i.e. $\boldsymbol{A}$), which is the vector of Fourier coefficients of `a`, is then turned into `A_prime` (i.e. $\boldsymbol{A'}$) with a multiplication for `1j*kappa` (i.e. $\mathrm{j}\boldsymbol{\kappa}$).

The way the Fast Fourier Transform and its inverse can be used to compute first order derivatives in Python is simply shown through Listing 12, in which $\mathrm{a}(x) = \sin(x)$, $x \in [0, 2\pi[$ and $N = 2^{10}$ have been implemented. The result, i.e. an approximation of $da/dx = \cos(x)$, is displayed together with the approximation of $\mathrm{a}(x)$ in Fig. B.2.

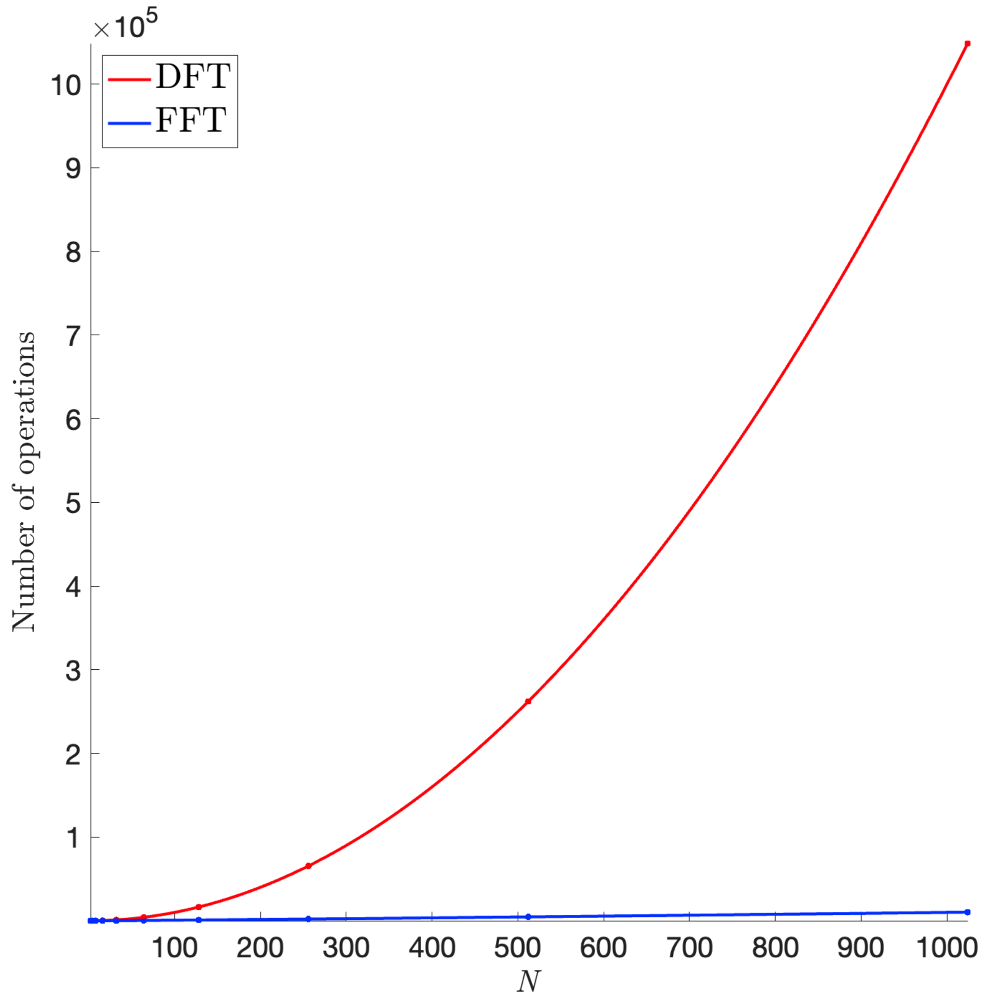


Figure B.1: Visual comparison between the number of numerical operations to compute a mono-dimensional Discrete Fourier Transform with or without the help of the Fast Fourier Transform algorithm, as the number of discretizing points vary. The plot has been realized with MATLAB.

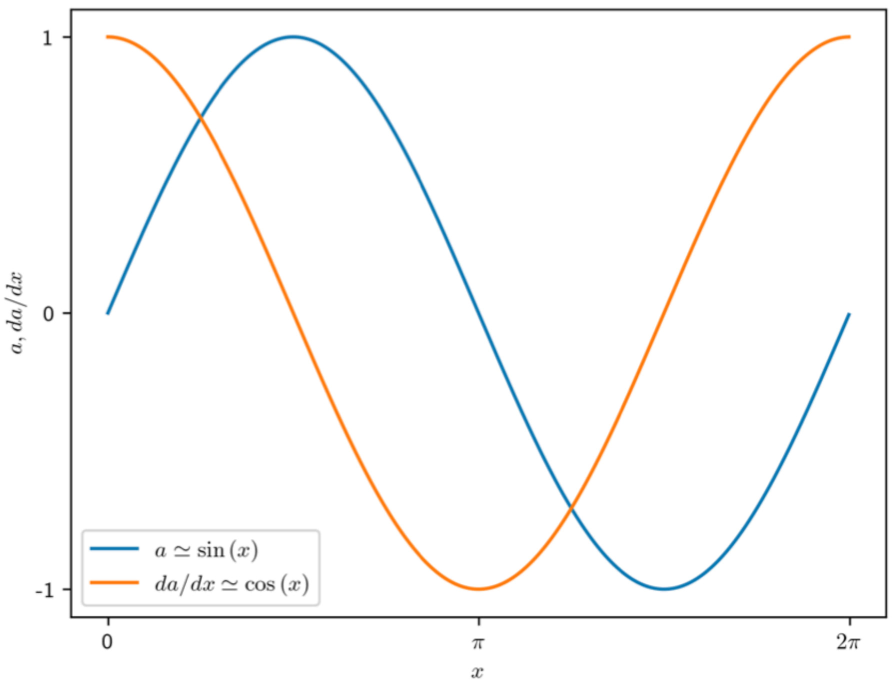


Figure B.2: Visual representation of `a` and `a_prime` of Listing 12. They approximate a sinusoidal function and its derivative, with the latter being obtained through direct and inverse Fast Fourier Transform algorithms.

Appendix C

Mean square displacement and anomalous diffusion of particles

This Appendix is based on [Wik22a], [Wik22b], [Pet+88] and [dev22].

C.1 Mean square displacement

The mean square displacement $r^2(t)$ (also indicated as MSD or $\langle r^2(t) \rangle$) is a quantity that, in essence, measures the portion of space being explored by entities (e.g. a particle or a group of them) through the comparison between their positions at time t and some reference positions.

C.1.1 $r^2(t)$ for a system of N particles

There are many formulae that can be used to calculate $r^2(t)$. Considering N particles constituting a single system to be studies, the most simple formula is the mean value of the deviations of their positions at time t with respect to their initial positions at $t = 0$:

$$r^2(t) = \frac{1}{N} \sum_{n=0}^{N-1} \left| \boldsymbol{r^{(n)}}(t) - \boldsymbol{r^{(n)}}(0) \right|^2, \tag{C.1}$$

where $\boldsymbol{r^{(n)}}(t)$ is the position of the n-th particle at time t. In the case of a 2D space and Cartesian coordinates $\boldsymbol{r} = (x, y)$, the same formula would be written as:

$$r^2(t) = \frac{1}{N} \sum_{n=0}^{N-1} \left[\left(x^{(n)}(t) - x^{(n)}(0) \right)^2 + \left(y^{(n)}(t) - y^{(n)}(0))^2 \right) \right]. \tag{C.2}$$

C.1.2 $r^2(t)$ for single particle tracking

When there is interest in following single particles, their positions are usually registered at specific instants of time, which are commonly spaced with an arbitrary fixed spacing Δt. In those cases, r^2 is defined not as an average of deviations over the number of particles, but as an average over time intervals within the same trajectory:

$$r^2(t) = r^2(\tau\Delta t) = \frac{1}{T-\tau}\sum_{k=0}^{T-\tau}\left|\boldsymbol{r}_{k+\tau} - \boldsymbol{r}_k\right|^2, \quad \tau = 0, ..., T-1, \tag{C.3}$$

where:

- T is the number of time intervals between instants, i.e. there are $T+1$ instants of time at which the particle has been tracked.
- τ represents the generic τ-th instant of time (apart from the very last one).
- $t = \tau\Delta t$ is the numerical value of the τ-th instant of time that is being considered;
- r^2 is evaluated from $t = 0$ to $t = (T-1)\Delta t$. This allows, for example, to plot $r^2(t)$ as a set of T data points.
- $\boldsymbol{r}_{k+\tau} = \boldsymbol{r}(k\Delta t + \tau\Delta t)$ is the position of the particle at the $(k+\tau)$-instant of time.
- $\boldsymbol{r}_k = \boldsymbol{r}(k\Delta t)$ is the position of the particles at the k-th instant of time.

C.1.3 $r^2(t)$ for a system of N particles inspired by single particle tracking

It is possible to modify Eq. C.2 by exploiting Eq. C.3 in the case of N particles whose positions are being recorded as in *single particle tracking*. From another perspective, it could be said that Eq. C.3 can be extended to account for having N particles. In any case, the end result is a merged formula that contains a double averaging and in which r^2 is referred to the entire system of particles:

$$r^2(t) = r^2(\tau\Delta t) = \frac{1}{N}\sum_{n=0}^{N-1}\frac{1}{(T-\tau)}\sum_{k=0}^{T-\tau}\left|\boldsymbol{r}^{(\boldsymbol{n})}_{k+\tau} - \boldsymbol{r}^{(\boldsymbol{n})}_k\right|^2, \quad \tau = 0, ..., T-1. \tag{C.4}$$

C.2 Anomalous diffusion

Plotting the evolution in time of the mean square displacement of a given system allows one to determine its diffusive regime, which can be normal (linear relationship between

r^2 and t), anomalous (non-linear relationship), or absent. If r^2 is associated with a set of experimental or numerical data, then performing a power-law curve fitting can help the analysis. In particular, when the fit is based on the equation

$$f(t) = (at)^b, \tag{C.5}$$

with a and b being real parameters to be found, then:

- If $b \simeq 1$, the fit is linear (or almost linear) and the system exhibits (normal) diffusion. In particular, the diffusion coefficient $\mathcal{D}$ (which can only be defined in this case) of the system is associated with the slope of the curve, that is $a^b \simeq a$. From a mathematical point of view, the diffusion coefficient can be defined as the asymptotic value of the ratio between the mean square displacement and time:

$$\mathcal{D} = \lim_{t\to\infty} \frac{r^2(t)}{t}. \tag{C.6}$$

- If $b \simeq 0$, the system practically does not diffuse at all.
- If $0 < b < 1$, the system is sub-diffusive.
- If $b > 1$, the system is super-diffusive. In particular, if $b > 2$, then it is labelled as hyper-ballistic.

C.2.1 Goodness of fit expressed through the coefficient of determination R^2

The coefficient of determination R^2 is a measure of how good is a curve fitting, i.e. how well the model offered by the fit represents the set of experimental or numerical data. The best possible score for R^2 is 1.0 and it is common to treat curve fittings with $R^2 \geq 0.99$ as acceptable. If $\boldsymbol{y} = (y_0, ..., y_{S-1})$ is the vector that contains the the experimental or numerical data in the correct order and $\hat{\boldsymbol{y}} = (\hat{y}_0, ..., \hat{y}_{S-1})$ is the vector that contain the values of the associated curve fitting, then the coefficient of determination is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{k=0}^{S-1}(y_k - \hat{y}_k)^2}{\sum_{k=0}^{S-1}(y_k - \bar{y})^2}, \tag{C.7}$$

where $\bar{y}$ is the mean value of $\boldsymbol{y}$:

$$\bar{y} = \frac{1}{S}\sum_{k=0}^{S-1} y_k. \tag{C.8}$$

In the case of anomalous diffusion and power-law curve fittings, the value of R^2 would be related to $r^2(t)$ and $f(t)$:

$$R^2(r^2, f) = 1 - \frac{\sum_{k=0}^{S-1}(r_k^2 - f_k)^2}{\sum_{k=0}^{S-1}\left(r_k^2 - \frac{1}{S}\sum_{k=0}^{S-1} r_k^2\right)^2}. \tag{C.9}$$

References

[Lit80] Robert Grayson Littlejohn. *Hamiltonian theory of guiding center motion.* 1980.

[Mon83] B Montagnini. “Lezioni di fisica del reattore nucleare”. In: *Universita degli Studi di Pisa, DIMNP* (1983).

[Pet+88] Marco Pettini et al. “Chaotic diffusion across a magnetic field in a model of electrostatic turbulent plasma”. In: *Physical Review A* 38.1 (1988), p. 344.

[Den90] Richard O Dendy. *Plasma dynamics.* Oxford University Press, 1990.

[HSK00] Archie A Harms, Klaus F Schoepf, and David Ross Kingdon. *Principles of fusion energy: an introduction to fusion energy for students of science and engineering.* World Scientific, 2000.

[Cir+04] Guido Ciraolo et al. “Control of Hamiltonian chaos as a possible tool to control anomalous transport in fusion plasmas”. In: *Physical Review E* 69.5 (2004), p. 056213.

[Bar05] Antonio Barletta. *Introduzione matematica alla trasmissione del calore.* Pitagora, 2005.

[Sta07] Weston M Stacey. *Nuclear Reactor Physics, Second Edition, Completely Revised and Enlarged.* John Wiley & Sons, 2007.

[Str07] Julius Adams Stratton. *Electromagnetic theory.* Vol. 33. John Wiley & Sons, 2007.

[Vil07] Laurent Villard. *La turbulence dans les plasmas.* [Online; Accessed: 03-May-2022. 2007. URL: `https://interstices.info/la-turbulence-dans-les-plasmas/`.

[Fre08] Jeffrey P Freidberg. *Plasma physics and fusion energy.* Cambridge university press, 2008.

[Joh11] Steven G Johnson. “Notes on FFT-based differentiation”. In: *MIT Applied Mathematics, Tech. Rep.* (2011).

[BK12] R Bilato and R Kleiber. “IPP Summer University for Plasma Physics, September 17-21, 2012, Garching”. In: *IPP Summer University for Plasma Physics and Fusion Research.* Max-Planck-Institut für Plasmaphysik. 2012.

[Cha18] Cristel Chandre. *Hamiltonian systems of charged particles in electromagnetic fields: towards gyrokinetic theory.* Institut de Mathématiques de Marseille, 2018.

[TC18] Natalia Tronko and Cristel Chandre. “Second-order nonlinear gyrokinetic theory: from the particle to the gyrocentre”. In: *Journal of Plasma Physics* 84.3 (2018).

[IRF19] IRFM. *Magnetic Fusion : main principles.* [Online; accessed 03-May-2022]. 2019. URL: `https://irfm.cea.fr/en/fusion/`.

[HS20] George Hrabovsky and Leonard Susskind. *Classical mechanics: the theoretical minimum.* Penguin UK, 2020.

[Cha21] Cristel Chandre. *Report on Electrostatic Guiding-Center Theory.* CNRS, Aix Marseille Univ, I2M, 13009 Marseille, France, 2021.

[com21] NumPy community. *NumPy Reference, Release 1.21.0.* 2021.

[GL21] Xavier Garbet and Maxime Lesur. *Gyrokinetics.* CEA Cadarache: Saint Paul lez Durance, Provence-Alpes-Côte d’Azur, FR, 2021.

[BK22] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control.* Cambridge University Press, 2022.

[dev22] scikit-learn developers. *R^2 score, the coefficient of determination.* [Online; Accessed: 10-May-2022. 2022. URL: `https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score`.

[Wik22a] Wikipedia contributors. *Anomalous fiffusion — Wikipedia, The Free Encyclopedia.* [Online; accessed 03-May-2022]. 2022. URL: `https://en.wikipedia.org/wiki/Anomalous_diffusion`.

[Wik22b] Wikipedia contributors. *Mean squared displacement — Wikipedia, The Free Encyclopedia.* [Online; accessed 03-May-2022]. 2022. URL: `https://en.wikipedia.org/wiki/Mean_squared_displacement`.

[RFX] Consorzio RFX. *La configurazione Reversed Field Pinch (RFP).* [Online; accessed 03-May-2022]. URL: `https://www.igi.cnr.it/ricerca/magnetic-confinement-research-in-padova/la-configurzione-rfp/`.