

FEDERAL UNIVERSITY OF PELOTAS
Institute of Physics and Mathematics
Graduate Program in Physics



Master Dissertation

A physics based feature engineering framework for trajectory analysis

Eduardo Henrique Mossmann

Pelotas, 2022

Eduardo Henrique Mossmann

A physics based feature engineering framework for trajectory analysis

Master Dissertation presented to the Institute of Physics and Mathematics of Federal University of Pelotas, partial requirement for obtaining the title of Master in Physics.

Supervisor: José Rafael Bordin

Cosupervisor: Maurício Moreira Soares

Pelotas, 2022

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

M913p Mossmann, Eduardo Henrique

A physics based feature engineering framework for trajectory analysis / Eduardo Henrique Mossmann ; José Rafael Bordin, orientador ; Maurício Moreira-Soares, coorientador. — Pelotas, 2022.

175 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Física, Instituto de Física e Matemática, Universidade Federal de Pelotas, 2022.

1. Aprendizado de máquina. 2. Análise de trajetórias. 3. Difusão. I. Bordin, José Rafael, orient. II. Moreira-Soares, Maurício, coorient. III. Título.

CDD : 533.63

Eduardo Henrique Mossmann

A physics based feature engineering framework for trajectory analysis

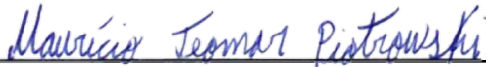
Master Dissertation approved, como requisito parcial para obtenção do grau de Master in Physics, Institute of Physics and Mathematics, Federal University of Pelotas.

Data da defesa: 16 de Agosto de 2022

Banca Examinadora:



José Rafael Bordin (Supervisor)
Doctor – Federal University of Pelotas



Maurício Jeomar Piotrowski
Doutor em Física – Universidade Federal de Pelotas

DocuSigned by:



71ACECC5B67E462

Frederico Schmitt Kremer
Doutor em Biotecnologia – Universidade Federal de Rio Grande



Elizane Efigênia Moraes
Doutora em Física – Universidade Federal da Bahia

RESUMO

MOSSMANN, Eduardo H.. **A physics based feature engineering framework for trajectory analysis**. Supervisor: José Rafael Bordin. Cosupervisor: Maurício Moreira Soares. 2022. 175 f. Master Dissertation (Graduate Program in Physics) – Instituto de Física e Matemática, Universidade Federal de Pelotas, Pelotas, 2022.

Análise de trajetória é de grande importância para o entendimento de sistemas dinâmicos e suas propriedades. Trajetórias podem descrever a evolução temporal de uma variável em diferentes contextos, como o movimento da célula cancerígenas, a volatilidade de uma ação na bolsa de valores, o crescimento populacional e assim por diante. Nos sistemas físicos, onde estamos interessados na evolução temporal da posição de um ou de um conjunto de corpos, existem sistemas onde a posição desses corpos depende da concentração de matéria em diferentes locais. Nesses contextos, o movimento dos corpos tende a ser no sentido de uma área com alta concentração de matéria para uma região de baixa concentração. Este movimento, tecnicamente chamado de fenômeno de transporte, é chamado de Difusão. Como uma maneira geral de descrever a evolução temporal de um corpo em tais sistemas, usamos o Deslocamento Médio Quadrático. Os tipos de movimento, às vezes chamados de Classes de Difusão, são uma maneira de descrever o movimento de corpos em sistemas onde a Difusão é observada. Para classificar a difusão, a literatura utiliza a dependência temporal do Deslocamento Quadrático Médio. No presente trabalho, apontamos que usar esta dependência temporal como a única grandeza para classificar a Difusão dá origem a um problema imediato, onde podemos ter duas ou mais Classes de Difusão possíveis para o mesmo movimento. Com isso em mente, propusemos uma estrutura de engenharia de atributos baseada em física para análise de trajetória chamada TrajPy como uma possível solução para esse problema. A estrutura contém três componentes principais. A primeira nos permite realizar a análise de trajetórias computando múltiplas quantidades de interesse físico e estatístico para qualquer trajetória, seja essa trajetória retirada de experimentos ou gerada em uma simulação computacional. A segunda componente é uma combinação de duas interfaces gráficas, que são modelos de interface que permite a interação com dispositivos digitais por meio de elementos gráficos como menus e botões. A primeira nos permite calcular as quantidades físicas e estatísticas de forma que não precisamos estar familiarizados com programação em Python. A segunda interface gráfica foi desenvolvida para ser um primeiro passo para uma solução geral para o gargalo tecnológico presente no processo de descoberta de medicamentos. A terceira componente nos permite simular os quatro tipos básicos de movimento (Difusão Normal, Anômala, Confinada e Movimento Direto com Difusão) com uma gama de parâmetros para que possamos usar essas simulações como uma conexão entre análise de trajetórias e algoritmos de classificação externo no contexto de Aprendizado de Máquina para que possamos classificar a Difusão de partículas de uma forma mais geral para que evitemos o problema envolvendo a sobreposição de classes de difusão. Como demonstração da aplicação do TrajPy, realizamos análise de trajetória e classificação de difusão para sistemas que mimetizam

a capacidade das células de serem deformadas. Para isso, simulamos vários sistemas, usando Dinâmica Molecular, em uma combinação de valores de pressão e constante de mola relacionada à Lei de Hooke, onde cada sistema é composto por 400 anéis poliméricos bidimensionais. Como resultados, observamos que os anéis poliméricos, uma vez que a pressão atinge um determinado limiar, apresentam uma transição dinâmica de Difusão Normal para Difusão Confinada, ou seja, à medida que a pressão aumenta, os anéis poliméricos ficam confinados dentro de uma região como efeito do aumento da pressão. Em seguida, classificamos a Difusão de cada anel polimérico para cada sistema e observamos o mesmo comportamento sob a perspectiva do algoritmo Random Forest Classifier usando Aprendizado de Máquina.

Palavras-chave: Aprendizado de Máquina. Análise de trajetórias. Difusão.

ABSTRACT

MOSSMANN, Eduardo H.. **A physics-based feature engineering framework for trajectory analysis**. Advisor: José Rafael Bordin. Coadvisor: Maurício Moreira Soares. 2022. 175 p. Dissertation (Master in Physics) – Instituto de Física e Matemática, Universidade Federal de Pelotas, 2022.

Trajectory analysis is of great importance for the understanding of dynamical systems and their properties. Trajectories may describe the time evolution of a variable in different contexts, such as the motion of cancer cell, the volatility of a stock in the stock market, the populational growth and so forth. In physical systems, where we are interested in the time evolution of the position of one or a set of bodies, there are systems where the position of these bodies depends on the concentration of matter at different locations. In such context, the movement of bodies tends to be in the direction of an area with high concentration of matter to a region of low concentration. This motion is called Diffusion. As a general way of describing the time evolution of a body in such systems, we use the Mean Squared Displacement. The motion types, sometimes called the Diffusion Classes, are a way of describing the movement of bodies in systems where Diffusion is observed. To classify the diffusion, the literature uses the time dependence of the Mean Squared Displacement. In the present work, we pointed out that using this time dependence as the only quantity to classify the Diffusion will give rise to an immediate problem, where we can have two or more possible Diffusion Classes for the same motion. With that in mind, we proposed a physics-based feature engineering framework for trajectory analysis called TrajPy as a possible solution to this problem. The framework contains three main components. The first allows us to perform trajectory analysis by computing multiple quantities of physical and statistical interest for any trajectory, whether this trajectory is taken from experiments or generated in a computer simulation. The second component is a combination of two graphical-user interfaces. The first allows us to compute the quantities of the first component in a way that we do not need to be familiar with Python programming. The second graphical-user interface was developed to be a first step towards a general solution to the bottleneck present in the drug discovery process. The third component allows us to simulate the four basic motion types with a range of parameters so that we may use these simulations as a connection between trajectory analysis and Machine Learning, where we aim to classify the Diffusion of particles in a more general way so that we avoid the problem involving the overlapping of diffusion classes. As a demonstration of the application of TrajPy, we performed trajectory analysis and diffusion classification for systems that mimic the capability of cells to be deformed. For this, we simulated multiple systems, using Molecular Dynamics, in a combination of values of pressure and the spring constant related to Hooke's Law, where each system is composed of 400 two-dimensional polymer rings. As a result, we observed that the polymer rings, once the pressure reaches a certain threshold, present a dynamical transition from Normal Diffusion to Confined Diffusion, i.e., as the pressure increases, the polymer rings become trapped inside a region as an effect of the increase in pressure. Then, we classified the Diffusion of each

polymer ring for every system and observed the same behavior from the perspective of the Random Forest Classifier algorithm using Machine Learning.

Keywords: Machine Learning. Trajectory Analysis. Diffusion.

LIST OF FIGURES

Figure 1.1 – Four examples of physical systems in different scales.	20
Figure 1.2 – Time evolution of the concentration of two different particles.	22
Figure 1.3 – Graphical comparison of the four MSD models.	24
Figure 1.4 – The distributions of β for different classes of Diffusion.	26
Figure 2.1 – Graph of the MSDisplacement by Time Average in log-log scale.	33
Figure 2.2 – Visual comparison of the Fractal Dimension for four different trajectories.	34
Figure 2.3 – Asymmetry for four different trajectories.	36
Figure 2.4 – Anisotropy for four different trajectories.	36
Figure 2.5 – Straightness for four different trajectories.	37
Figure 2.6 – Efficiency for four different trajectories.	38
Figure 2.7 – Gaussianity for four different trajectories.	39
Figure 2.8 – Kurtosis for four different trajectories.	39
Figure 2.9 – The different behaviors of the Velocity Autocorrelation Function for four different trajectories.	41
Figure 2.10–The possible values for the velocity skewness and kurtosis.	43
Figure 2.11–A visual representation of the Fourier Transform.	44
Figure 2.12–A visual representation of the effects of the velocity’s magnitude on the overall behavior of Direct Motion with Diffusion.	46
Figure 2.13–A visualization of the first graphical-user interface.	47
Figure 2.14–A visualization of the second graphic-user interface.	49
Figure 2.15–A visualization of the two different regions the second graphic-user interfaces uses to compute important attributes.	51
Figure 2.16–A visualization of the two graphs provided by the second graphical-user interface.	52
Figure 3.1 – Schematic depiction of the bead spring ring model for low and high k_{cm}	55
Figure 3.2 – Density ρ as function of pressure p^* and Isothermal compressibility β_T for all values of k_{cm}	57
Figure 3.3 – Cell center of mass radial distribution functions and cumulative two-body entropy.	59
Figure 3.4 – MSD by Ensemble average for $p^* = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k^* = 10.0$	61
Figure 3.5 – MSD by ensemble average for $p^* = 0.25$ and $k^* = 10.0$	62
Figure 3.6 – A comparison between the MSD by Ensemble average for $p^* = 0.25$ and $k^* = 10.0$ and the MSD calculated from simulated Confined Motion with a radius of confinement $r_c = 0.5$	63
Figure 3.7 – Average MSD ratio for every system.	63
Figure 3.8 – Average Anomalous Exponent for every system.	64
Figure 3.9 – Average Fractal Dimension for every system.	65

Figure 3.10–Average Asymmetry for every system.	66
Figure 3.11–Average Anisotropy for every system.	67
Figure 3.12–Average Straightness for every system.	68
Figure 3.13–Average Efficiency for every system.	68
Figure 3.14–Average Gaussianity for every system.	69
Figure 3.15–Number of polymer rings, per system simulated for all values of k^* and p^* , whose gaussianity was measured to be larger than one.	70
Figure 3.16–Histograms for the values of the observed gaussianity larger than one for every value of p for $k = 10.0$	71
Figure 3.17–Average Gaussianity for every system, once we ignore the large values we observed.	72
Figure 3.18–Average Kurtosis for every system.	72
Figure 3.19–Number of polymer rings, per system simulated for all values of k^* and p^* , whose kurtosis was measured to be larger than 5.	73
Figure 3.20–Histograms for the observed values of kurtosis larger than 5 for every value of p^* for $k^* = 10.0$	73
Figure 3.21–Average kurtosis for every system, once we ignore the large values.	74
Figure 3.22–Average Velocity Kurtosis for every system.	75
Figure 3.23–Number of polymer rings, per system simulated for all values of k^* and p^* , whose velocity kurtosis was measured to be larger than 2.	76
Figure 3.24–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p^* for $k^* = 10.0$	76
Figure 3.25–Average velocity kurtosis for every system, once we ignore the large values.	77
Figure 3.26–Average Velocity Skewness for every system.	77
Figure 3.27–Average Diffusion Coefficient for every system.	78
Figure 3.28–Number of polymer rings, per system simulated for all values of k^* and p^* , whose diffusion coefficient was measured to be larger than 5.	79
Figure 3.29–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 10.0$	80
Figure 3.30–Average diffusion coefficient for every system, once we ignore the large values.	81
Figure 3.31–Scatter matrix comparing the attributes used to perform diffusion classification.	83
Figure 3.32–Confusion Matrix for the predictions of the Random Forest algorithm.	84
Figure 3.33–ROC curve presenting the true and false positive rates for classification model evaluations.	86
Figure 3.34–ROC curves for classification model evaluations in the context of prediction of cardiac surgical operative mortality.	86
Figure 3.35–Results of the diffusion classification process.	87
Figure B.1 – Histograms for the gaussianity values larger than one for $k = 25.0$ for all pressures p	135

Figure B.2 – Histograms for the gaussianity values larger than one for $k = 50.0$ for all pressures p .	136
Figure B.3 – Histograms for the gaussianity values larger than one for $k = 75.0$ for all pressures p .	137
Figure B.4 – Histograms for the gaussianity values larger than one for $k = 100.0$ for all pressures p .	138
Figure B.5 – Histograms for the gaussianity values larger than one for $k = 150.0$ for all pressures p .	139
Figure B.6 – Histograms for the gaussianity values larger than one for $k = 200.0$ for all pressures p .	140
Figure B.7 – Histograms for the gaussianity values larger than one for $k = 250.0$ for all pressures p .	141
Figure B.8 – Histograms for the gaussianity values larger than one for $k = 300.0$ for all pressures p .	142
Figure B.9 – Histograms for the gaussianity values larger than one for $k = 500.0$ for all pressures p .	143
Figure B.10–Histograms for the kurtosis values larger than 5 for $k = 25.0$ for all pressures p .	144
Figure B.11–Histograms for the kurtosis values larger than 5 for $k = 50.0$ for all pressures p .	145
Figure B.12–Histograms for the kurtosis values larger than 5 for $k = 75.0$ for all pressures p .	146
Figure B.13–Histograms for the kurtosis values larger than 5 for $k = 10.0$ for all pressures p .	147
Figure B.14–Histograms for the kurtosis values larger than 5 for $k = 150.0$ for all pressures p .	148
Figure B.15–Histograms for the kurtosis values larger than 5 for $k = 200.0$ for all pressures p .	149
Figure B.16–Histograms for the kurtosis values larger than 5 for $k = 250.0$ for all pressures p .	150
Figure B.17–Histograms for the kurtosis values larger than 5 for $k = 300.0$ for all pressures p .	151
Figure B.18–Histograms for the kurtosis values larger than 5 for $k = 500.0$ for all pressures p .	152
Figure B.19–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 25.0$.	153
Figure B.20–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 50.0$.	154
Figure B.21–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 75.0$.	155
Figure B.22–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 100.0$.	156
Figure B.23–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 150.0$.	157
Figure B.24–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 200.0$.	158
Figure B.25–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 250.0$.	159

Figure B.26–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 300.0$	160
Figure B.27–Histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 500.0$	161
Figure B.28–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 25.0$	162
Figure B.29–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 50.0$	163
Figure B.30–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 75.0$	164
Figure B.31–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 100.0$	165
Figure B.32–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 150.0$	166
Figure B.33–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 200.0$	167
Figure B.34–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 250.0$	168
Figure B.35–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 300.0$	169
Figure B.36–Histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 500.0$	170
Figure B.37–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 25.0$	171
Figure B.38–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 50.0$	171
Figure B.39–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 75.0$	172
Figure B.40–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 100.0$	172
Figure B.41–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 150.0$	173
Figure B.42–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 200.0$	173
Figure B.43–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 250.0$	174
Figure B.44–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 300.0$	174
Figure B.45–MSD by Ensemble average for $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 500.0$	175

LIST OF CODES

Code A.1 – Codes for the computation of each attribute.	103
Code A.2 – Codes for the simulation of trajectories for the four basic motion types.	118
Code A.3 – Code for single object tracking.	122
Code A.4 – ESPResSo code of the simulation.	124

LIST OF TABLES

Table 2.1 – Descriptive table of the attributes.	30
--	----

SUMMARY

1	INTRODUCTION	19
1.1	Overview	19
1.2	Physical concepts of interest	19
1.3	The problem of diffusion classification and a proposed solution	25
2	PHYSICS-BASED FEATURE ENGINEERING	29
2.1	Computing attributes of the trajectory	31
2.1.1	Previous features	31
2.1.2	New Features	40
2.2	Simulating trajectories for the four basic motion types	45
2.3	Using Graphical-User Interfaces - attributes calculations and animal tracking for drug discovery	47
3	TRAJECTORY ANALYSIS AND DIFFUSION CLASSIFICATION OF TWO-DIMENSIONAL POLYMER RINGS	53
3.1	The physical system of interest	53
3.1.1	General Overview	53
3.1.2	The Model and Simulation Details	54
3.1.2.1	The 2D drop-like model for deformable cells	54
3.1.2.2	Simulations Method and Details	55
3.1.3	Results and Discussion	57
3.1.3.1	Thermodynamic and Structural Analysis	57
3.2	Trajectory analysis	60
3.3	Machine Learning and diffusion classification	79
3.3.1	A general description of Machine Learning	80
3.3.2	Diffusion classification	82
4	CONCLUSION	89
	BIBLIOGRAPHY	91
	ATTACHMENTS	101
	ATTACHMENT A – ALGORITHMS	103
	ATTACHMENT B – SUPPLEMENTARY MATERIAL	135

1 INTRODUCTION

1.1 Overview

Trajectory analysis is of great importance to the understanding of dynamical systems and their properties. We define a trajectory to be a series of values of a given variable measured over time. In that sense, trajectory analysis is a quantitative approach that describes the time evolution of that variable, so that we can extract relevant information such as patterns and behavioral changes of the data being analysed.

The variable measured over time may change according to the context of research. In Economics, we might be interested in the time evolution of a set of economic indicators designed to measure the economic growth (MARIANI *et al.*, 2022). In Ecology, the variable of interest can describe oscillations in the number of members of a certain population of animals following the predator-prey model (CHOWDHURY; BANERJEE; PETROVSKII, 2022).

Lastly, in Physics, we are interested in describing the time evolution of the position of one or multiple bodies in a variety of length scales. In the nanometric scale, we might want to determine the optimal drug delivery mechanism to maximize the effects of a certain medicine, seeking the geometric shape of nanoparticles of medicine that allows the drug to travel the furthest in the human body (ZHANG *et al.*, 2018). In the scale of micrometers, the motility of cancer cells has been the object of extensive research due to the process of metastasis (SHI *et al.*, 2022). The modelling of bird migration, in the scale of meters, has been used to study the intricate ways flocks of birds communicate (PANCERASA *et al.*, 2019). Lastly, in astrophysical scales, the process of chaotic lensing around boson stars has been observed and studied for its fractal patterns. Every one of the systems we just described are vastly different from one another. They vary in complexity and scales, however the concept of trajectory remains the same no matter what the system is. Figure 1.1 shows the four different physical systems we described.

Regarding physical systems where the movement of bodies is related to the amount of matter present in different regions of the system, such as the process of dialysis (SCHUETT *et al.*, 2021) and cancer cell motility (FUENTE; LOPEZ, 2020), the concept of diffusion is used to describe such movement. Given that our main goal in the present work involves the idea of diffusion and trajectory analysis, let us discuss some fundamental physical concepts.

1.2 Physical concepts of interest

In order to discuss the idea of diffusion and show where it comes from, we will use Fick's laws of diffusion, first proposed by Fick (1855). To do that, we begin by providing the definition of two main concepts, the flux and concentration of matter in a system.

Simply put, the flux \vec{J} of matter is defined as the amount of matter that crosses an area per unit of time, whereas the concentration C indicates the quantity of matter¹ located at a certain region

¹ In the present work, we will use interchangeably the words "matter" and "substance".

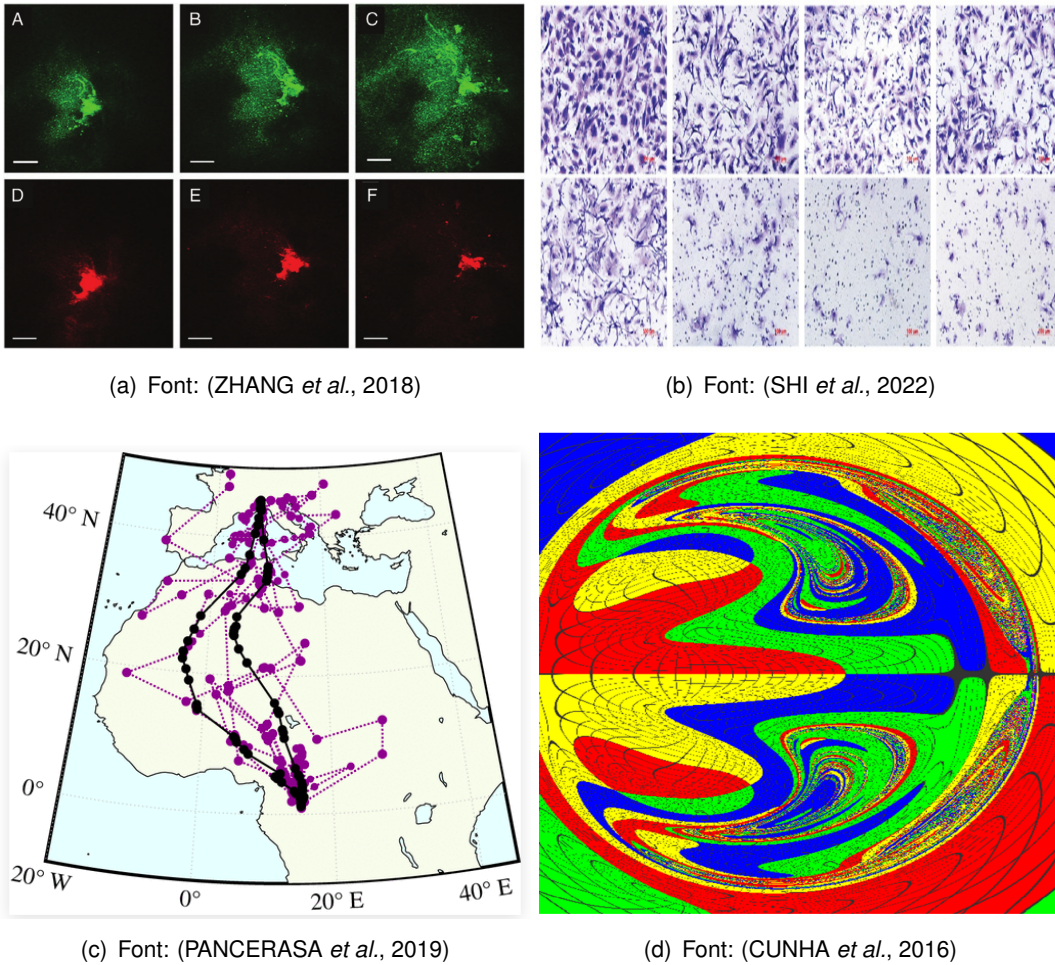


Figure 1.1 – (a) The motion of two different kinds of nanoparticles of medicine over time. (b) The process of migration and invasion of ovarian cancer cells. (c) The trajectory of a flock of birds through the african continent. (d) Fractal patterns that emerge from gravitational lensing around boson stars.

of the system of interest (JACKSON, 2006).

We now make two basic assumptions about the behavior of the substance present in the system. First, we assume that the substance will avoid any location where C is considered large, since the more particles there are in a specific set of coordinates, the more difficult it is for the substance to pass through that area. Second, we assume that the conservation of matter holds true at all times (DAYANANDA, 2022). This means that two measurements of the total mass at different times must yield the same result.

By using our first assumption, we see that the flux of matter goes in the opposite direction of the gradient of concentration, since the gradient points to the direction where C grows larger. In order to express this proportionality between \vec{J} and C as an equation, we must use a constant.

We will call such a constant the “diffusion” of matter through the system. By doing that, we arrive at what is known as Fick’s first law. This law is defined as

$$\vec{J} = -D\nabla C(r,t), \quad (1.1)$$

where D is the diffusion coefficient that tells us that diffusion is the movement of matter from a region

of high concentration C to a region of low concentration. By dimensional analysis, we notice that the diffusion D is written in units of $\text{m}^2 \cdot \text{s}^{-1}$. Also, it is clear that diffusion tells us how fast the substance can sweep out a unit of area.

The negative sign in Equation (1.1) indicates that the flux of the substance happens in the direction of the lowest concentration. In other words, the substance tends to move along the path of least resistance where, in this context, the resistance is provided by the collisions between the substance's and environment's particles.

Given that the particles are not bounded to stay fixed, it is easy to see that in a small region, the flux of particles going into the region from the left might be different from the flux going outside from the right. This means the concentration C changes according to the location and time, which means C is a function of r and t , $C(r,t)$. We may express the amount of substance present in this region per unit of time as the difference between the inward and outward fluxes $\nabla \cdot \vec{J}(r,t)$, combine it with the concept of concentration and rearrange the difference to obtain:

$$\frac{\partial C(r,t)}{\partial t} + \nabla \cdot \vec{J}(r,t) = 0, \quad (1.2)$$

which is the equations that ensures the conservation of mass.

Now we can substitute \vec{J} in Equation (1.2) by substituting Equation (1.1) in Equation (1.2) to obtain:

$$\frac{\partial C(r,t)}{\partial t} - D\nabla^2 C(r,t) = 0. \quad (1.3)$$

Equation (1.3) is known as Fick's second law, also called the Diffusion Equation. This is the equation that governs every system where our two basic assumptions hold true. Given that the Laplacian of the concentration C , $\nabla^2 C(r,t)$, tells us that the value of C located at r is the average value of the concentration in the surrounding region, it is simple to see that Equation (1.3) states that C changes over time according to the nearby concentration. The higher the concentration in the vicinity of the point r , the faster C changes in r .

At this point, it is important to point out that, in the present work, we consider the applications of Fick's laws to the context of Self Diffusion, where the substance we discussed so far is composed of one kind of particle. (FRENKEL; SMIT, 2002). To see an application of Fick's laws in the context of Interdiffusion, where the substance is composed of two or more kinds of particles, see Hirakawa *et al.* (1973).

Let us solve Equation (1.3) to obtain a relation for the concentration C . To do that, we apply the Dirac delta function (BOAS, 1999) as a boundary condition:

$$C(r,0) = M\delta(r). \quad (1.4)$$

In Equation (1.4), we will set $M = 1$ for simplicity. Notice that, by computing the integral

$$\int_{-\infty}^{\infty} C(r,0)dr = \int_{-\infty}^{\infty} \delta(r)dr = 1, \quad (1.5)$$

we assure the conservation of mass².

² If we had not set $M = 1$ in Equation (1.4), we would have obtained M itself as the result of the integration in Equation (1.5).

We may use the Fourier Transforms method following the work of Jackson (2006) in order to solve Equation (1.3). The solution is given by

$$C(r,t) = \frac{1}{(4\pi Dt)^{n/2}} \exp(-r^2/4Dt), \quad (1.6)$$

where n is the number of dimensions of the system.

Since Equation (1.6) is a Gaussian function, we may interpret this result as a Probability Density Function (PDF) that tells us the probability of finding the particle at the point r at time t . Notice that at small t , the probability of finding the particle at $r = 0$ is higher than at any other time. This is the consequence of using the boundary condition given by Equation (1.4). Figure 1.2 shows the evolution of C at all one-dimensional points x for different values of t . As we can clearly see, the fact

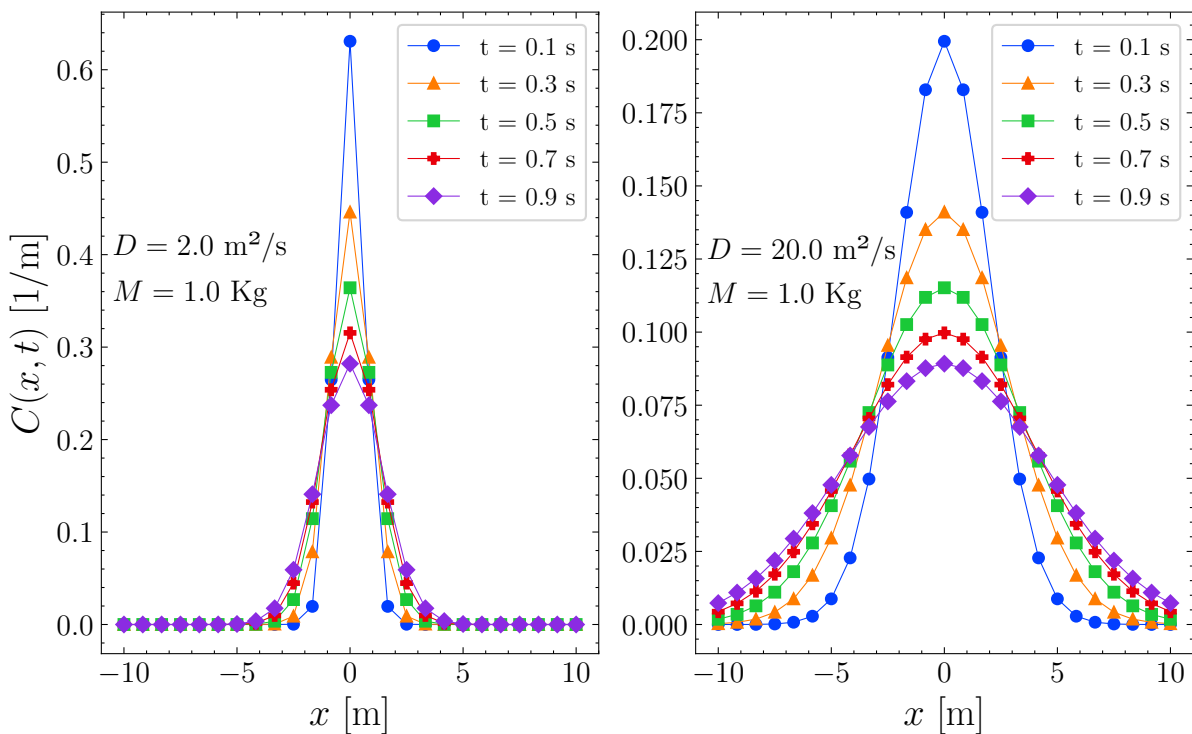


Figure 1.2 – This figure shows the evolution of the concentration C over space and time for different values of the Diffusion Coefficient D . To generate these graphs, we have used Equation (1.6) and set $M = 1 \text{ Kg}$ for both. In the left graph, we have set $D = 2.0 \text{ m}^2 \cdot \text{s}^{-1}$ and in the right one, $D = 20.0 \text{ m}^2 \cdot \text{s}^{-1}$.

Fonte: The author.

that D is ten times higher on the right graph indicates that it is much easier for the particles of that system to move around.

Now that we have solved the diffusion equation, we may use its solution to obtain the Mean Squared Displacement (MSD) of a trajectory, defined as the squared deviation from the body's initial position over time (ALLEN; TILDESLEY, 1989). To do that, we write the MSD as the second moment³ of the PDF represented by C in Equation (1.6). The MSD is mathematically defined as:

$$\langle \vec{r}^2(t) \rangle = \int_{-\infty}^{\infty} r^2 C(r,t) d\vec{r}. \quad (1.7)$$

³ A distribution may be represented by its four moments. The first indicates the central tendency we call the mean. The second is the variance, the third is the skewness that measures the asymmetry of the distribution and the fourth is the kurtosis, a measure of the tails of the distribution (AGARWAL, 2006).

Once we compute the integral in Equation (1.7), we get

$$\langle \bar{r}^2(t) \rangle = 2nDt. \quad (1.8)$$

The diffusion modeled by Equation (1.8) is called **Normal diffusion**. According to Saxton e Jacobson (1997), Normal diffusion is not the only motion type observed in nature. There are infact four basic motion types: Normal, Anomalous, Direct motion with diffusion and Confined diffusion. Each type has its unique relation with the MSD and we can use this relations to extract pieces of information such as the diffusion itself. The mathematical formulations for the different motion types are the following:

$$\langle \bar{r}^2(t) \rangle \propto t, \quad \text{Normal diffusion}, \quad (1.9)$$

$$\langle \bar{r}^2(t) \rangle \propto t^\beta, \quad \text{Anomalous diffusion}, \quad (1.10)$$

$$\langle \bar{r}^2(t) \rangle \propto t^2, \quad \text{Direct motion with diffusion}, \quad (1.11)$$

$$\langle \bar{r}^2(t) \rangle \simeq r_c^2 [1 - A_1 \exp(-2A_2 nDt/r_c^2)], \quad \text{Confined diffusion}. \quad (1.12)$$

In Equation (1.10), β assumes values less than one, according to Saxton e Jacobson (1997). In Equation (1.12), r_c , A_1 and A_2 are the radius of confinement and two constants that characterize the shape of the confinement, respectively. In addition, we could write Equation (1.12) as $\langle \bar{r}^2(t) \rangle \propto t^\beta$ in order to express the time dependency of the MSD directly. Figure 1.3 shows a comparison of the four theoretical models for the MSD according to the motion type.

The linear time dependency of the MSD observed in Equation (1.9) was first calculated by Einstein (1956), when providing a mathematical description of the motion we nowadays refer to as Brownian motion. The concept of Brownian motion was named after the botanist Robert Brown, who was the first person to observe the irregular motion of particles of polen generated by collisions with particles when immersed in water (GENTHON, 2020). In order to mimic such behavior, the Random Walker, a particle that may take multiple steps of random length in a random direction was proposed (CODLING; PLANK; BENHAMOU, 2008). We will interpret, from now on, Normal diffusion as the motion type for the Random Walker.

Let us discuss the meaning of Anomalous diffusion. If we recall the two assumptions we made in order to obtain Fick's Laws - where we have stated that the substance in question will avoid any crowded location and that matter is conserved, we considered the environment the substance is located in to be structureless. We made this consideration in an indirect way, since we have not mentioned any details about the structure of the environment.

However, if we do consider the environment to impose geometric constraints of some kind, the first assumption we just mentioned becomes obscure, once we can no longer be sure that the substance will avoid crowded locations. These geometric constraints are one of the reasons we observe Anomalous Diffusion (SANTOS; JUNIOR; CIUS, 2022). This behavior may also present itself in many systems, such as Brownian Motion in inhomogeneous systems (OLIVEIRA *et al.*, 2019a).

An alternative way of describing the rise of Anomalous Diffusion would be to consider systems where the steps a Random Walker takes each time are not independent from each other. Systems as the ones we just described are observed, for instance, in Biology (KLAFTER; SOKOLOV, 2005).

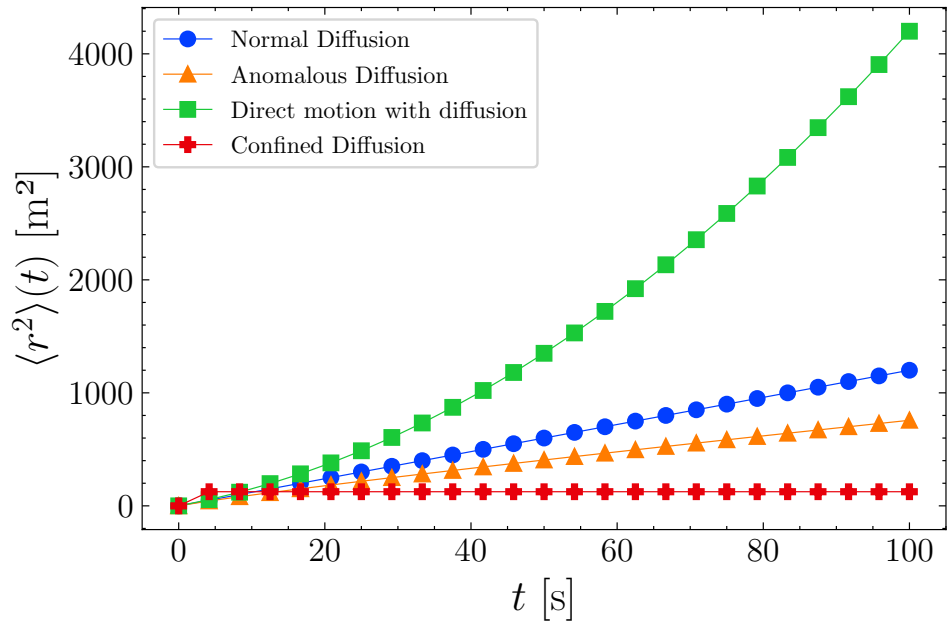


Figure 1.3 – A comparative plot of the four motion types. We defined the numerical values for each parameter as the following: $D = 2.0$, $n = 2$, $v = 0.3$, $A_1 = 2.0$, $A_2 = 3.0$, $r_c = 10.0$ and $\beta = 0.9$. As expected, the MSD for the Confined motion (red line) achieves a plateau, since the motion itself is limited (this will become clear later on). The Direct motion generates a MSD (green line) that grows fast for its quadratic time dependency, whereas the MSD for Normal diffusion (blue line) produces a straight line since the relation between the MSD and time is linear. Given that the exponent β must be less than 1, the MSD for this motion will always be smaller than either Normal diffusion or Direct motion with diffusion.

Fonte: The author.

We mentioned that β in Equation 1.10 is less than one. This fact was true in the context of study of Saxton e Jacobson (1997). However, as a general rule β assumes values that may be smaller or larger than one (OLIVEIRA *et al.*, 2019b). When $\beta < 1$, the diffusion regime is called Subdiffusion. On the other hand, if $1 < \beta < 2$, the regime is referred to as Superdiffusion.

Subdiffusion is observed when the movement of particles in a system is slower, on average, than the one observed in Normal Diffusion. This implies that the values of the MSD for Subdiffusion are smaller across time. One interesting example of Anomalous Diffusion in the Subdiffusion regime is the one of molecules' diffusion through the plasma membrane of the human cell (KRAPF, 2015). This membrane is known for its crowded inner structure that causes restriction of the motion of macromolecules.

Superdiffusion, on the other hand, happens when the movement of particles is faster, on average, than the one observed in Normal Diffusion. In contrast with Subdiffusion, the values of the MSD across time are high when compared to the MSD for Normal Diffusion. A famous example of Superdiffusion is encountered in the so called *Lévy flights*, where a Random Walker remains in motion without changing direction for a random amount of time (ZABURDAEV; DENISOV; KLAFTER, 2015). In the present work, we will use “Anomalous diffusion” as a simple way to refer to both subdiffusion and superdiffusion in the context we just described.

In systems where the MSD presents a quadratic time dependency, the active transport of matter is observed (WU; LIBCHABER, 2000). By active transport we mean that the substance that presents diffusion - perhaps Normal diffusion - is immersed in a fluid that moves over time. The Direct motion with diffusion, sometimes called Ballistic diffusion is observed, for instance, in the movement of bacteria (CASPI; GRANEK; ELBAUM, 2002). In the present work, we will use “Direct motion with diffusion” as a simple way to refer to situations where the MSD presents a squared time dependency.

Lastly, let us discuss the Confined diffusion. This motion type is observed when the particles of the substance present Normal diffusion while confined in a certain region (BURADA *et al.*, 2009). This region may have any geometric shape, as long as it is able to contain the substance trapped. Confined diffusion is observed, for instance, when spherical tracers are put inside Periodic Porous Nanostructures (RACCIS *et al.*, 2011).

Over the past decades, multiple computational models have been proposed by researchers in order to improve the overall understanding of motion. Theoretical models that aim to mimic single and multiple cell dynamics in both two dimensions (FANG *et al.*, 2020) and (HECK *et al.*, 2020) and three dimensions (CAMPBELL; BAGCHI, 2017) have been vastly proposed. From the perspective of trajectory analysis of three-dimensional Molecular Dynamics Simulations, two computational models among many are proposed by Michaud-Agrawal *et al.* (2011) and Roe e Cheatham (2013). Regarding the analysis of cell interactions, Jin *et al.* (2021) designed a package called CellChat. In the context of Machine Learning being used, Li (2022) developed scTour, a package that uses Deep Learning to model cell dynamics. In addition, the trajectory analysis of experimental data has also been discussed in the literature. For an approach that uses Machine Learning in sets of data generated from hand tremors related to Parkinson’s Disease, see San-Segundo *et al.* (2020). Lastly, Wagner *et al.* (2017) utilizes Machine Learning in combination with trajectory analysis and object tracking techniques in order to model single cell dynamics and diffusion classification.

1.3 The problem of diffusion classification and a proposed solution

In the last section, we have described the fundamental physical concepts related to diffusion, the MSD and its time dependency. In this section, we will show that a problem immediately rises if one wishes to classify the diffusion as one of the four motion types. In addition, we will propose a solution to this problem, where we construct an alternative approach to trajectory analysis and diffusion classification.

Let us begin by describing the problem we previously mentioned. As we established in Section 1.2, we may use the time dependency of the MSD as a way to classify the diffusion as one of the four basic motion types: Normal, Anomalous, Direct motion with diffusion and Confined diffusion. For now, we will use β to indicate the value of the time exponent related to each MSD time dependency. That means that for Normal diffusion, $\beta = 1$, for Direct motion with diffusion, $\beta = 2$, for Confined diffusion, $\beta < 1$ and for Anomalous diffusion⁴, $\beta \neq 1$.

⁴ We must recall that for Anomalous diffusion, β must also be less than two.

If we simulate many different trajectories for each of the motion types⁵, compute the MSD for each trajectory and then extract the value of the time exponent β , we should get approximate values of β that match the expected values for each class of diffusion⁶. However, as we can see in Figure 1.4, the distributions of the values of β do not match the expected values.

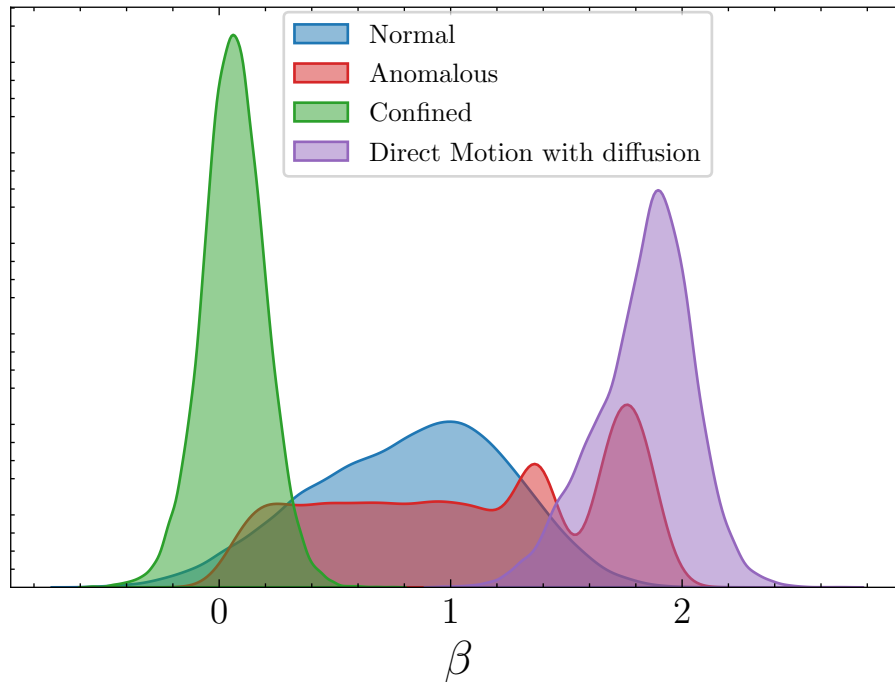


Figure 1.4 – This graph provides the distributions of β for each motion type: Normal, Confined, Anomalous and Direct motion with diffusion. These distributions show that when β is computed from the MSD for trajectories simulated based on the four motion types, we do not obtain the expected results for β . This, in turn, implies that we should not use β as the only parameter to classify the diffusion of trajectories, since we observe overlaps of β for different motion types.

Fonte: The author.

Such a problem may rise from underlying complexities that have not been taken into account in the literature when the physical model we discussed in Section 1.2 was proposed. In addition, as Hubicka e Janczura (2020) have stated, the trajectory of a cell, protein or any other object might present a shift regarding the time dependency of the MSD as the system evolves over time. A combination of such complexities and this shift in the time dependency might be a more general reason the diffusion classification problem is observed.

As a possible solution to this problem, a Python package called TrajPy was proposed by Moreira-Soares (2020) based on the ideas of Wagner *et al.* (2017). The overall objective of TrajPy is to be a Physics based computational tool for three-dimensional trajectory analysis for both data generated from simulations and experiments. The objective of the current dissertation is to present, discuss and improve TrajPy. In addition, we wish to perform trajectory analysis and diffusion

⁵ We will discuss how these simulations are done based on the approach presented by Wagner *et al.* (2017) in Section 2.2.

⁶ By class of diffusion, we mean Normal, Anomalous, Confined and Direct motion with diffusion. We will, from now on, use “motion types” and “diffusion class” interchangeably.

classification of a physical system of our choosing in order to show why TrajPy was proposed as a solution to the problem we have discussed in this section.

In Chapter 2, we will discuss TrajPy's main goals and its approach to the problem of diffusion classification. In Sections 2.1 through 2.3 we discuss the three sections of TrajPy that are used so that it achieves its purposes, along with the contributions we have made to the package.

In Chapter 3, we will use TrajPy to perform trajectory analysis and diffusion classification of a physical system we chose for demonstration purposes. In Section 3.1, we describe the system itself along with its applications in Physics. In Section 3.2, we provide the results of the trajectory analysis of the system. In Section 3.3 we present the diffusion classification process and the results we have gathered.

Lastly, in Chapter 4 we will review the main concepts and results we discussed throughout the current dissertation. In addition, we designate possible new steps we might take in the future regarding the development of TrajPy.

2 PHYSICS-BASED FEATURE ENGINEERING

In this chapter, we will discuss the Physics based computational tool called TrajPy, a general framework developed to perform trajectory analysis and diffusion classification. In Section 1.1, we established the importance of trajectory analysis. We must now discuss TrajPy's approach to the problem of diffusion classification we presented in Section 1.3.

We propose two workflows for data analysis that are independent and complementary. The first branch depicts the development of a classification model for diffusion modes: confined, normal, anomalous and direct motion. We generate synthetic trajectories by employing 4 independent simulation engines that provide trajectories on each of the 4 labels. We vary the space of parameters¹ for these simulations and obtain a range of different trajectories that obey the same diffusion regime. Then we apply feature engineering to quantify these trajectories with the proposed attributes in this dissertation. The data generated with features and the labels are used to train a classifier that can be used later for classifying unseen data generated from simulations or experiments. We provide a dataset of synthetic data that can be used to train new models (Moreira-Soares, 2022). The second workflow regards to statistical analysis of experimental (unlabeled) raw trajectory data. We perform the same feature engineering process on the experimental data, obtaining the same attributes used to train our classifier. Therefore, if deemed relevant, the analyst can apply the classifier to the experimental data and obtain the diffusion modes. The features can be useful to quantify different systems of interest across many areas and statistical inference can be performed to draw novel insights about the systems' nature. In addition, new classifiers can be trained based on other labels that may be interesting in other fields.

In Section 2.1, we present the physical and statistical attributes that can be used to perform trajectory analysis. Specifically, Subsection 2.1.1 contains the quantities implemented in the software by Moreira-Soares (2020). In Subsection 2.1.2 we discuss the new attributes added to TrajPy throughout the current research project. In Section 2.2 we present how the motion types we used to describe the diffusion classification problem were simulated and how we will use them. Lastly, in Section 2.3 we present both graphical-user interfaces TrajPy provides to the user. The first is used to analyse trajectories by computing the attributes discussed in Section 2.1 in a way that the user does not need to be familiar with programming. The second graphical-user interface is responsible for performing macroscopic object tracking, so that we are able to analyse the trajectory of bodies in the context of experiments. This graphical-user interface is another contribution we have made to TrajPy. For completeness sake, we have provided the algorithms we developed for each concept discussed in this chapter in Code A.1 of Attachment A.

As an easy way to summarise all the attributes TrajPy offers to the process of trajectory analysis we will discuss shortly, we provide a table that briefly describes each attribute. Table 2.1 contains the name of each attribute, what they measure, their smallest and largest values in parenthesis - when

¹ These parameters are, for instance, the initial position, the number of time steps and the number of dimensions. We will fully discuss these parameters in Section 2.2

Table 2.1 – Descriptive table of the attributes.

Attribute	Measurement	Possible Values
MSD by Time average	Average quadratic displacement	(0,-)
MSD by Ensemble average	Average quadratic displacement	(0,-)
MSD Ratio	Ratio between two MSD's	$\left\{ \begin{array}{l} \text{MSD Ratio} = 0 \rightarrow \text{Normal diffusion,} \\ \text{Positive MSD Ratio} \rightarrow \text{Confined or Anomalous diffusion} \\ \text{Negative MSD Ratio} \rightarrow \text{Direct motion with diffusion} \end{array} \right.$
Anomalous Exponent	Time dependency of the MSD	(0,-)
Fractal Dimension	Trajectory's irregularity	$\left\{ \begin{array}{l} \text{Fractal Dimension} = 1 \rightarrow \text{Trajectory is a straight line,} \\ \text{Fractal Dimension} \approx 2 \rightarrow \text{Trajectory resembles a Random Walker,} \\ \text{Fractal Dimension} > 2 \rightarrow \text{Trajectory undergoes physical limitations.} \end{array} \right.$
Gyration Radius	Description of the shape of a curve	(-,)
Asymmetry	Existence of preferred direction of motion	(0, ∞)
Anisotropy	Symmetry of the positions' distribution	(0,1)
Straightness	Similarity between the trajectory and a straight line	(0,1)
Positions' Kurtosis	Shape of the positions' distribution	$\left\{ \begin{array}{l} K < 3 \rightarrow \text{Flat peak and long tails,} \\ K = 3 \rightarrow \text{Gaussian Distribution,} \\ K > 3 \rightarrow \text{Sharp peak and short tails.} \end{array} \right.$
Gaussianity	Similarity between the positions' distribution and a Gaussian distribution	(0,-)
Efficiency	How effective the trajectory is	(0,1)
Velocity	Displacement per unit of time	(-,)
Velocity Autocorrelation Function	Similarity of the particle's velocity at different times	(-,)
Green-Kubo Relation	Diffusion coefficient	(0,-)
Velocity Description	Central tendency and spread of the velocity distribution	(-,)
Frequency Spectrum	Underlying frequencies via Fourier Transform	(0,-)

Fonte: The author.

that applies².

2.1 Computing attributes of the trajectory

Given that we have established the main idea of TrajPy and its approach to diffusion classification, it is time to present every physical and statistical quantity³ Trajpy provides so that we can perform trajectory analysis. To do that, we will discuss the features implemented by Moreira-Soares (2020) in Subsections 2.1.1 and our contributions in Subsection 2.1.2.

2.1.1 Previous features

The first attribute we will discuss is the **MSD**. In Section 1.2 we have defined the MSD to be the deviation of a particle's position with respect to a fixed starting point and provided the MSD models for each motion type. Now, let us discuss how we can compute the MSD of a trajectory composed of discrete data points.

There are two mathematically equivalent ways to compute the MSD, according to the Ergodic Hypothesis (ALLEN; TILDESLEY, 1989). We can either compute the MSD by Time Average or Ensemble Average⁴, where the former is the usual time average of the position of a single trajectory and the latter is an average of the position taken over the total number of single particle trajectories.

The **MSD by Ensemble Average** is defined as

$$\langle \vec{r}^2 \rangle(t) = \frac{1}{N} \sum_{n=1}^N |\vec{r}_n(t) - \vec{r}_n(0)|^2, \quad (2.1)$$

where N is the total number of trajectories, $\vec{r}_n(t)$ is the position of the n -th particle at time t and $\vec{r}_n(0)$ is the initial position of the same particle. In order to compute this feature, we take the difference between the position of the n -th particle at time t and its initial position. Then we square that difference so that the positive and negative values do not cancel out and move on to the next particle. Once we have squared all the differences using the same value for t , we add all of them and divide the result by the total number of particles N . This is the result for the MSD by Ensemble average for that value of t . The next step is similar to the previous one, but now we use the next available value for t and so on.

The **MSD by Time average** differs from the previous feature due to the fact that it takes in as input a single trajectory. The MSD by Time average is defined by Wagner *et al.* (2017) as

$$\langle \vec{r}_\tau^2 \rangle = \frac{1}{T - \tau} \sum_{t=1}^{T-\tau} |\vec{r}(t + \tau) - \vec{r}(t)|^2, \quad (2.2)$$

² When there are no smallest or largest values, we will inform what specific values indicate.

³ From now on, we will use “quantity”, “attribute” and “feature” to refer to the same concept.

⁴ An ensemble of particles can be interpreted in two ways: a set of particles that differ in their initial positions but are under the same influences in the system or, more generally, repeating the same experiment multiple times where, in each time, the particles present the same initial positions.

where τ is the time interval⁵, also called the time lag, between the two positions and T is the total trajectory time. To compute this feature, we interpret $\vec{r}(t + \tau)$ as the position of the particle at the $(t + \tau)$ -th time step of the trajectory⁶. By setting $\tau = 1$, since $\tau = 0$ would simply yield zero as result, and $t = 1$, we take the difference between the particle's position at the $(1 + \tau)$ -th and first time steps. Then we square this difference so that positive and negatives values do not cancel out and move on to the next value of t , in this case, $t = 2$. The next step is to plug in $t = 2, 3, \dots, T - \tau$ in Equation (2.2), add the results and divide the final result by $T - \tau$. The numerical output we get is the value of the MSD by Time average for $\tau = 1$. To extract information about the particle's displacement, we can set $\tau = 0, 1, 2, \dots$ (when analysing discrete trajectories) and compute the MSD by Time average for each value of τ . It is worth mentioning that, since we are always working with finite data, we cannot set a value for τ that is bigger than the data itself, given the definition of τ in this context.

The next feature TrajPy provides is the **MSD Ratio**, a feature that allows the user to characterize the shape of the MSD curve by using two different values for the time lag τ of the MSD by Time Average. The MSD Ratio is defined as:

$$\langle \vec{r}^2 \rangle_{\tau_1, \tau_2} = \frac{\langle \vec{r}_{\tau_1}^2 \rangle}{\langle \vec{r}_{\tau_2}^2 \rangle} - \frac{\tau_1}{\tau_2}, \quad (2.3)$$

where $\tau_1 < \tau_2$ and $\langle \vec{r}_{\tau_i}^2 \rangle$ is the Mean Squared Displacement by Time average. When we insert Equations (1.9) - (1.12) in Equation (2.3), we see that the MSD Ratio has the following values for each of the four motion types (WAGNER *et al.*, 2017):

$$\begin{cases} \text{MSD Ratio} = 0 & \text{for Normal diffusion,} \\ \text{MSD Ratio} > 0 & \text{for Confined and Anomalous diffusion,} \\ \text{MSD Ratio} < 0 & \text{for Direct motion with diffusion.} \end{cases}$$

That tells us that we can obtain the motion type simply by computing the MSD Ratio or looking at the motion graph. However, if we want to calculate the diffusion constant itself, we may extract this information by either fitting a MSD curve through our data, as we have already established, or computing the so called Green-Kubo's relation for diffusion (LEE, 2000). This relation will be explored in Section 2.1.2.

Now that we have established the four basic motion types and discussed the importance of the MSD Ratio, we may present the **Anomalous exponent** feature present in TrajPy. We have seen that the time dependency plays an important role in the MSD, so we would expect that the exponent who dictates this dependency is also crucial.

Let us consider the relation between the MSD and the Anomalous diffusion provided by Equation (1.10):

$$\langle \vec{r}^2 \rangle(t) = 2Dnt^\beta.$$

⁵ When dealing with a trajectory formed by discrete data points - which is the case for TrajPy, the values for τ must, by definition, be a non-negative integer. Negative values would make us compute the difference between the previous position and the next, not the other way around (which is what we want). Non-integer values do not make sense in the context of discrete trajectories, i.e, there is no position at a between the first and second positions.

⁶ Keep in mind that a trajectory is a list of positions with regular time steps, so $t = 1$ means the first time step and so on.

To compute the exponent β , we apply the logarithm function to both sides of this equation and make use of the logarithmic rules to obtain:

$$\log(\langle \bar{r}^2 \rangle(t)) = \log(2nD) + \beta \log(t). \quad (2.4)$$

Then, we take the partial derivative with respect to $\log(t)$ on both sides of Equation (2.4) to end up with:

$$\beta = \frac{\partial \log(\langle \bar{r}^2 \rangle(t))}{\partial \log(t)}, \quad (2.5)$$

since $\log(2nD)$ does not change with time⁷. Equation (2.5) provides a simple way to compute the exponent β , we must simply calculate the angular coefficient of the straight line in a log-log graph of the MSD as a function of time.

Figure 2.1 is a log-log plot of the MSD by Time Average as a function of τ . To construct this plot, we used the log of Equation (1.10) and set the possible parameters as follows: $D = 2.0$, $n = 2$, $\beta = 0.9$. We can easily check that the angular coefficient of the straight line if Figure 2.1 is precisely equal to 0.9, so Equation (2.5) does in fact provide the correct answer.

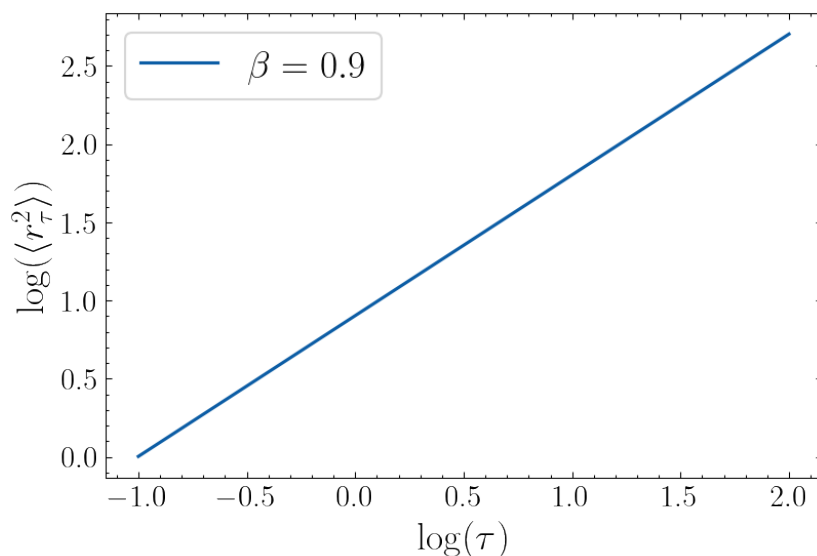


Figure 2.1 – A log-log graph of the MSD by Time Average as a function of τ . The angular coefficient of the straight line produced is the value of the anomalous exponent β .

Fonte: The author.

Another useful feature in TrajPy is the **Fractal Dimension**. This concept is connected to Kirkby (1983) and the idea of Fractal Geometry. Fractals are useful in many fields of study, including medicine (XINGYUAN; CHAO; JUAN, 2009), social sciences (BANASZAK *et al.*, 2015) and more. For a complete discussion about the use of fractals emerging from chaos and their applications in Science, see Strogatz (2014).

In TrajPy, we compute the Fractal Dimension D_f of curves proposed by Katz e George (1985), a direct adaptation of the same concept in the Fractal Geometry context to the analysis of curves in a

⁷ The diffusion coefficient D may fluctuate around a fixed value over time, but these fluctuations are minimal so we may not consider them.

plane. The Fractal Dimension that interests us is mathematically defined as

$$D_f = \frac{\log n}{\log(ndL^{-1})}, \quad (2.6)$$

where n is the number of steps taken to form the curve, d is the largest distance between two points of the curve and L is the total length of the curve.

In a sense, D_f tells us how irregular the trajectory is. If the curve is a straight line, then $D_f = 1$ - which means the trajectory is not irregular at all. However, if $D_f \rightarrow 2$, the trajectory can be associated to Brownian motion modeled by Normal Diffusion. And, finally, if a curve is erratic to the point where the motion described by the curve is undergoing some physical limitation - as in the Anomalous or Confined diffusion, for instance - the Fractal Dimension takes values of $D_f > 2$.

Figure 2.2 shows the value of the Fractal Dimension for each of the four motion types: Normal diffusion, Confined Diffusion, Anomalous diffusion and Direct motion with diffusion. As we can clearly see, the Direct motion with diffusion represented by the green line is a straight line and has a Fractal Dimension of $D_f = 1$. The other three motion types have a Fractal Dimension $D_f > 1$ and the more erratic the motion is, the higher the value of D_f . In addition, the more the curve crosses over itself, the higher the value of D_f .

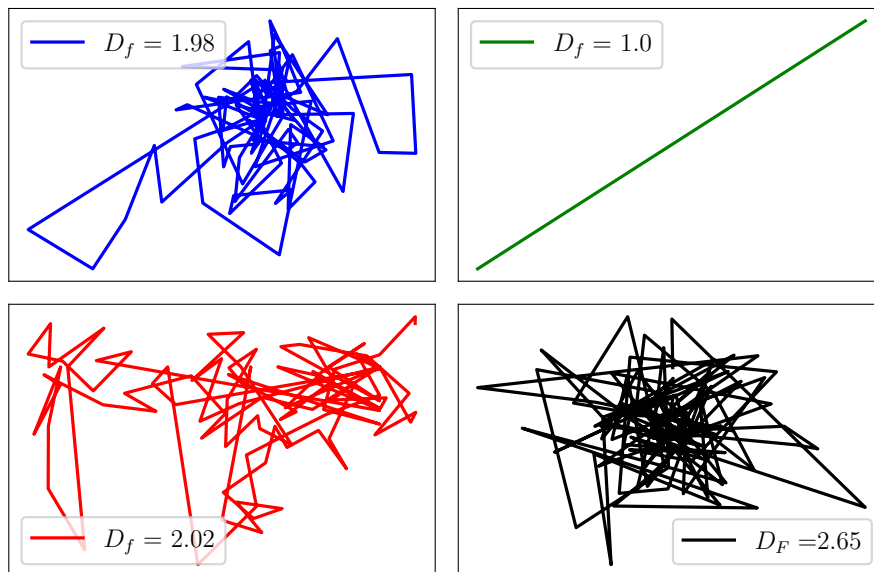


Figure 2.2 – A comparison for different values of the Fractal Dimension D . The blue, green, red and black lines refer to the four basic motion types: Confined Diffusion, Direct motion with diffusion, Normal Diffusion and Anomalous Diffusion.

Fonte: The author.

The next feature we will discuss is the **Radius of Gyration Tensor**, sometimes called Gyration Tensor. There is a close relation between this tensor and the Moment of Inertia Tensor, where both can be used to describe the shape of an object using its distribution of mass and particles. While the Inertia Tensor makes use of the masses of each particle, the Gyration Tensor uses the particles' positions as input to describe the macroscopic object (VYMĚTAL; VONDRÁŠEK, 2011).

To describe a distribution of mass, for example, the Gyration Tensor makes use of its eigenvalues, where the approximate shape is obtained from ensembles averages of each eigenvalue of the

tensor in question (RUDNICK; GASPARI, 1987). This is where the Gyration Tensor becomes very useful, since we can use its eigenvalues to describe the shape of the trajectory used in TrajPy and obtain certain properties such as the asymmetry and kurtosis of the trajectory.

The Gyration Tensor is defined as

$$\overleftrightarrow{T} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} \quad (2.7)$$

where each element, according to vSolc (1971), is defined by

$$R_{ij} = \frac{1}{N} \sum_{k=1}^N x_i^{(k)} x_j^{(k)} - \frac{1}{N^2} \sum_{k=1}^N x_i^{(k)} \sum_{k=1}^N x_j^{(k)}. \quad (2.8)$$

Computing the eigenvalues of the gyration tensor is very useful in many fields of study such as polymer physics, where the Gyration Tensor can describe the dimensions of a polymer chain (FIXMAN, 1962).

Once we compute the eigenvalues and eigenvectors of the diagonalized Gyration Tensor, we may proceed to the next features implemented in TrajPy. The **asymmetry** makes use of R_i , which is the square root of the i -th eigenvalue λ , in such a way that it allows us to verify if there is any tendency for a preferred direction. This means we can analyse any given trajectory and quantify the amount of symmetry or lack thereof across the entire movement. The **asymmetry** was first mathematically defined by Huet *et al.* (2006) as

$$\gamma = -\log \left(1 - \frac{(R_1^2 - R_2^2)^2 + (R_1^2 - R_3^2)^2 + (R_2^2 - R_3^2)^2}{2(R_1^2 + R_2^2 + R_3^2)^2} \right). \quad (2.9)$$

Given the already established meaning of asymmetry, we can use this concept to evaluate the shape of the trajectory curve. A perfectly symmetric trajectory would yield $\gamma = 0$ whereas a trajectory in a straight line along the main axis would be completely asymmetric - since a straight line is formed by consecutive steps in the same direction - and, therefore, $\gamma \rightarrow \infty$. Figure 2.3 shows a direct comparison of different trajectories and their respective values for γ .

The **Anisotropy** makes use of the eigenvalues of the Gyration Tensor. Defined by Olgar e Janke (2013) as

$$k^2 = 1 - 3 \frac{\lambda_1 \lambda_2 + \lambda_2 \lambda_3 + \lambda_3 \lambda_1}{(\lambda_1 + \lambda_2 + \lambda_3)^2}, \quad (2.10)$$

the anisotropy contains information about the symmetry and dimensionality of the system. In simple terms, the anisotropy is the characteristic of a physical property that changes its value according to the direction it is measured Colin *et al.* (2019). One simple example of anisotropy would be the mass anisotropy, where the amount of matter changes according to the direction we look. In Equation (2.10), k^2 is limited to values between 0 and 1 where $k^2 = 0$ indicates the distribution of positions is symmetric in relation to the origin and, therefore, all eigenvalues are the same. If $k^2 = 1$, that same distribution is not symmetric in relation to the origin and implies that at least two eigenvalues are zero. This happens because the positions may be aligned as a linear chain and there is only one eigenvector that could represent it (FIXMAN, 1962). Figure 2.4 provides a comparison of the anisotropy for four different trajectories.

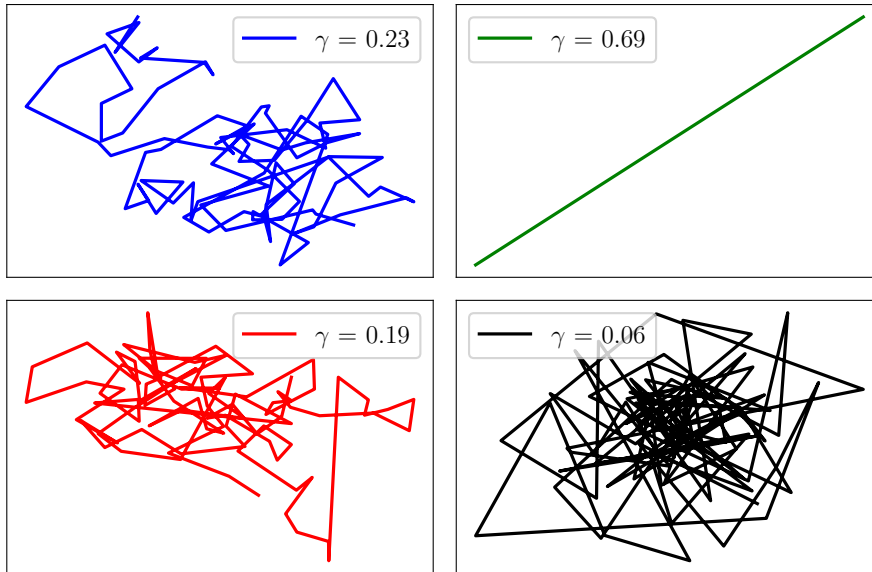


Figure 2.3 – This image shows a direct comparison for different values of the asymmetry γ . The blue, green, black and red lines refer to the four basic motion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion. As expected, the straight green line provides the highest value of γ .

Fonte: The author.

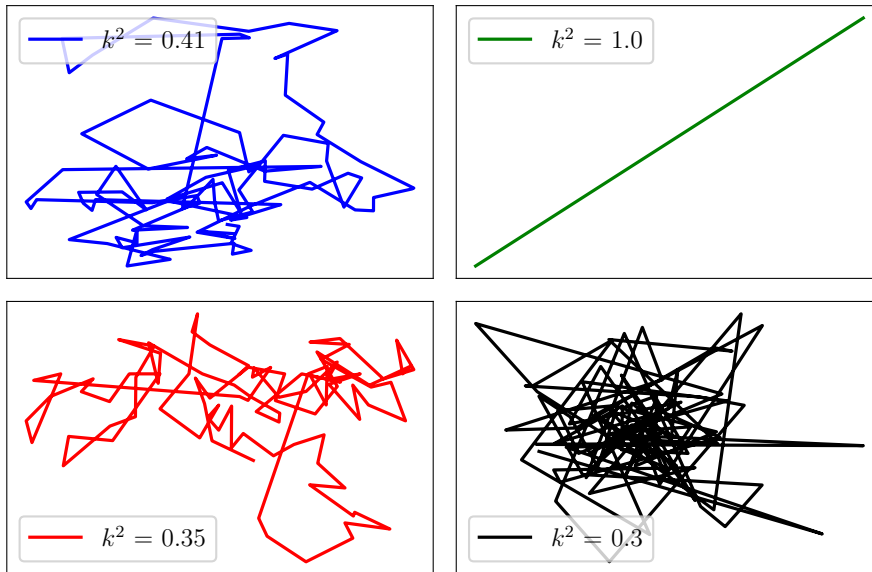


Figure 2.4 – This image shows a direct comparison for different values of the anisotropy k^2 . The blue, green, black and red lines refer to the four basic motion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion. As expected, the straight green line provides the highest value of k^2 .

Fonte: The author.

Another simple and yet useful quantity provided by TrajPy is the **Straightness** of the trajectory. The straightness measures the similarity between the actual trajectory and a straight line. This quantity, according to Benhamou (2004), is defined as

$$S = \frac{|\vec{r}_{N-1} - \vec{r}_0|}{\sum_{i=1}^{N-1} |\vec{r}_i - \vec{r}_{i-1}|}, \quad (2.11)$$

where the numerator is the distance, measured as a straight line, between the initial and second to last positions \vec{r}_0 and \vec{r}_{N-1} respectively. The denominator is the length of the trajectory measured as the sum of the individual distances between each pair of points of the trajectory.

If the trajectory takes the form of a straight line, then both terms of the fraction in Equation (2.11) are equal and, therefore, $S = 1$. However, as the trajectory gets less similar to a straight line, the larger is the length of the trajectory and, by extension, the larger the denominator becomes. In such cases, $S \approx 0$ as the trajectory increases in length. Figure 2.5 shows a comparison of different trajectories and their respective values of S .

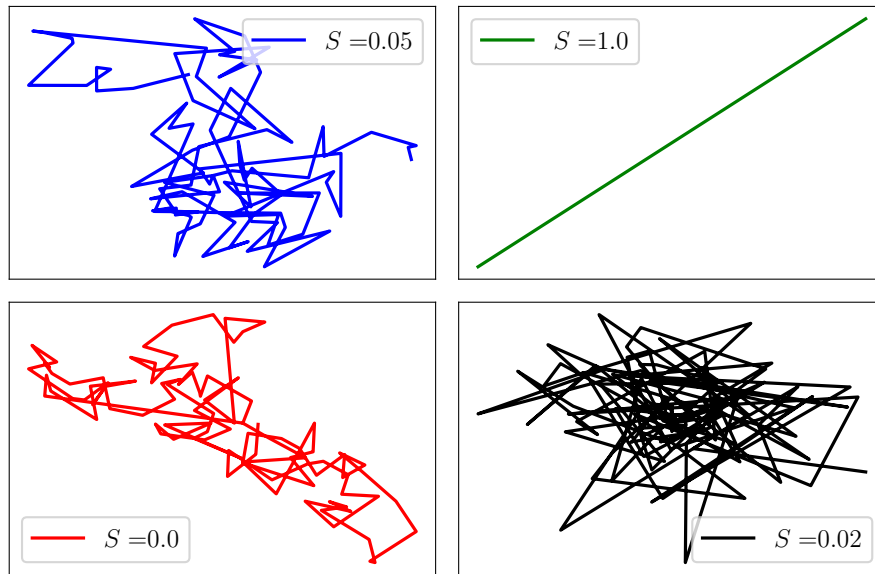


Figure 2.5 – This image shows a direct comparison for different values of the straightness S . The blue, green, black and red lines refer to trajectories presenting the four basic diffusion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion. As we expected, the straight green line presents $S = 1$.

Fonte: The author.

The **efficiency** of the trajectory, another quantity present in TrajPy, relies on a similar principle that of the **straightness**. Mathematically defined as

$$E_{ff} = \frac{|\vec{r}_{N-1} - \vec{r}_0|^2}{\sum_{i=1}^{N-1} |\vec{r}_i - \vec{r}_{i-1}|^2}, \quad (2.12)$$

the **efficiency** relates the square of the net displacement with the square of the trajectory length. According to Wagner *et al.* (2017), if the particle's initial and final positions r_0 and r_{N-1} are the same, the **efficiency** $E_{ff} = 0$ for any trajectory length. On the other hand, for a given net displacement - the numerator in Equation (2.12) - the more irregular the trajectory is, the smaller the value for E_{ff} in relation to a straight line. Figure 2.6 shows a comparison for the values of E_{ff} for trajectories of different diffusion types.

The next quantity present in TrajPy is the **gaussianity** of the trajectory. Simply put, this quantity measures how similar to a Gaussian Distribution the trajectory is. The **gaussianity** is defined

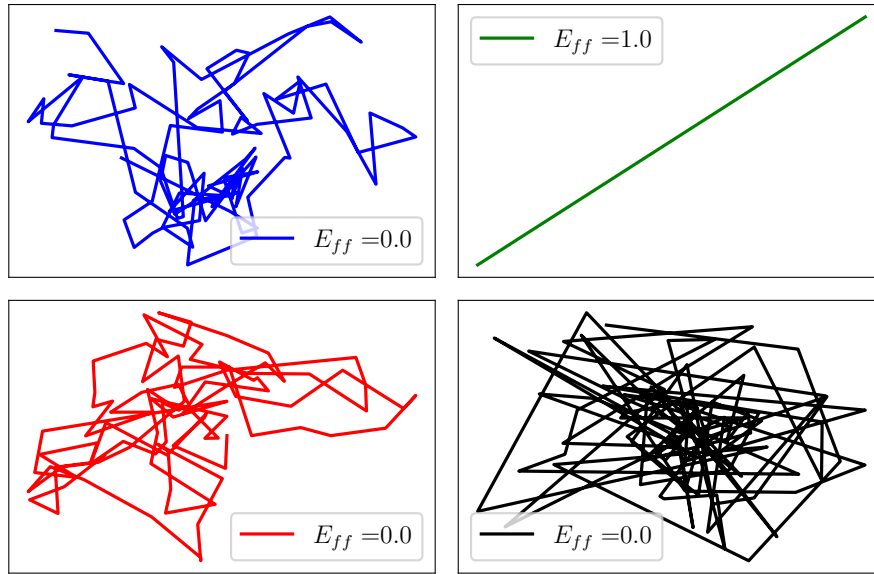


Figure 2.6 – This image shows a comparison for different values of the efficiency E_{eff} . The blue, green, black and red lines refer to trajectories presenting the four basic diffusion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion. As we can see, E_{eff} achieves the highest value for a straight line and rapidly goes to zero as the trajectory becomes more irregular.

Fonte: The author.

by Ernst, Köhler e Weiss (2014) as

$$G(\tau) = \frac{2\langle \bar{r}_\tau^4 \rangle}{3\langle \bar{r}_\tau^2 \rangle^2} - 1, \quad (2.13)$$

where $\langle \bar{r}_\tau^n \rangle$ is the n -th moment of the distribution computed by

$$\langle \bar{r}_\tau^n \rangle = \frac{1}{T - \tau} \sum_{t=1}^{T-\tau} |\bar{r}(t + \tau) - \bar{r}(t)|^n. \quad (2.14)$$

In Equation (2.14), T is the total number of steps and τ is the time lag. The **gaussianity** is useful when one wants to have an idea about the kind of diffusion a specific particle presents. For normal diffusion, we expect G to be zero and for the other diffusion types, G will present values different than zero (WAGNER *et al.*, 2017). Figure 2.7 shows a direct comparison for values of the **gaussianity** for different diffusion kinds.

The next feature we will discuss is the **kurtosis**. It measures the tailedness of the distribution of positions once we project this distribution along the eigenvector corresponding to the highest eigenvalue or the **gyration tensor** (HELMUTH *et al.*, 2007). To compute this feature, we take the scalar product between each position \vec{r}_i and the main eigenvector \vec{e}_1 : $r_i^p = \vec{r}_i \cdot \vec{e}_1$ and calculate the quartic moment of r_i^p given by

$$K = \frac{1}{N} \sum_{i=1}^N \frac{(r_i^p - \langle r^p \rangle)^4}{\sigma_{r^p}^4}. \quad (2.15)$$

In equation (2.15), $\langle r^p \rangle$ is the mean position of the projected trajectory and $\sigma_{r^p}^4 = (\sigma_{r^p}^2)^2$ is the variance of r_i^p . To put it in simple terms, the **kurtosis** measures how long the tails of the position

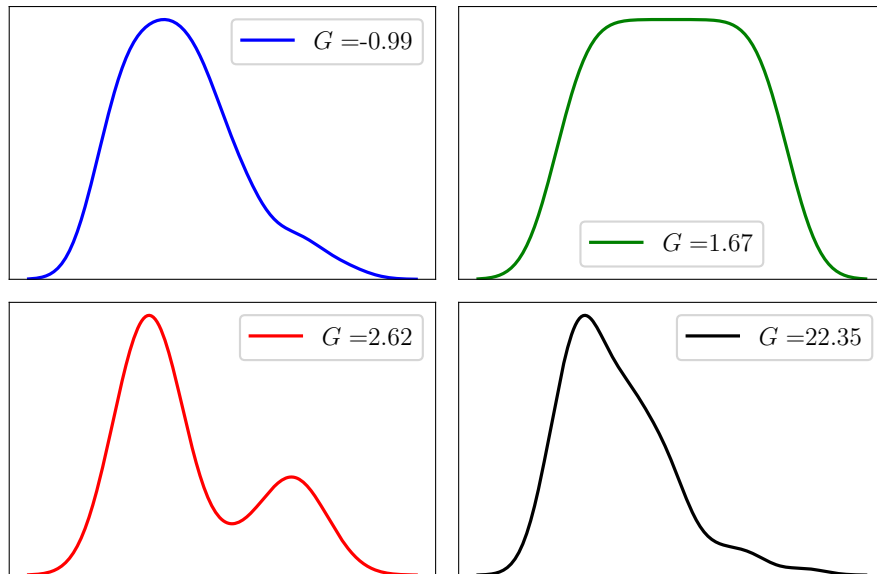


Figure 2.7 – A comparison for different values of the efficiency G . The blue, green, black and red lines refer to trajectories presenting the four basic diffusion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion.

Fonte: The author.

distribution are, it deals with the outliers - the data points far away from the expected value we usually call the mean. The higher the value of the **kurtosis**, the longer the tails are, and, by extension, the larger the number of outliers. Figure 2.8 provides a comparison for the values of kurtosis of four different trajectories when projected along their corresponding main eigenvector.

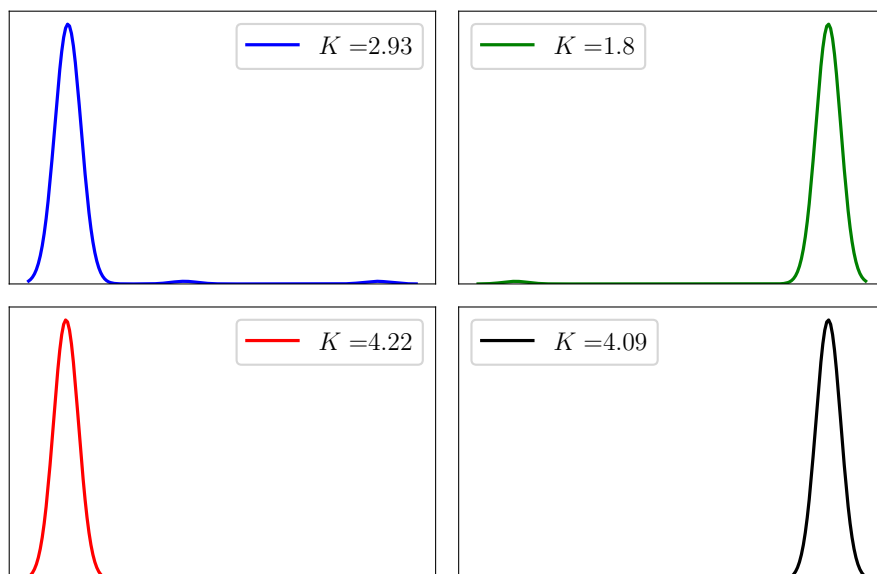


Figure 2.8 – This image shows a comparison for different values of the efficiency K . The blue, green, black and red lines refer to trajectories presenting the four basic diffusion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion when projected along their corresponding main eigenvector. The longer the tails, the larger the value of the kurtosis K

Fonte: The author.

The last feature of this subsection is the **velocity**. Given that, in TrajPy, we analyse trajectories that are composed of discrete data points, we compute the approximate velocity as the ratio between the displacement $\vec{r}_{i+1} - \vec{r}_i$ and the time step $t_{i+1} - t_i$.

2.1.2 New Features

In the previous section, we discussed the attributes that were implemented in TrajPy by Moreira-Soares (2020). In this section, we will discuss the ones implemented in the software throughout the current research project.

We begin by discussing the **Velocity Autocorrelation Function** (VACF), a quantity that measures the correlation between velocities at different times (ALDER; WAINWRIGHT, 1970). According to Despósito e Viñales (2009), the VACF is defined as

$$C_v(t) = \frac{1}{N} \sum_{i=1}^N (\vec{v}_i(t) \cdot \vec{v}_i(t=0)) = \langle v(t)v(0) \rangle, \quad (2.16)$$

where $\vec{v}_i(t)$ is the velocity of the i -th particle at time t and N is the total number of particles of the system. If we were to describe Equation (2.16) in simple terms, we could say that we take the scalar product of the velocity vectors of the particle i at different times in order to calculate how similar both vectors are⁸.

We care about this similarity - this **correlation** - because the way the velocity vectors differ from one another gives us information about both the system's thermodynamical equilibrium and dynamics. Which means the **Velocity Autocorrelation Function** contains information about the dynamical nature of the molecular process (LEVESQUE; ASHURST, 1974).

For forces with different magnitudes applied to different systems, such as solids and liquids, the VACF presents different behaviors. For instance, we would expect it to be a straight line if there are no forces acting on the system, since the momentum of each particle would remain the same over time. On the other hand, for weak forces, the VACF decays exponentially over time because a weak force gradually changes the momentum of the particles and, by extension their velocities became gradually different⁹. For strong forces we may observe a few - or even several - oscillations¹⁰ of the VACF between positive and negative values. This happens because strong forces make the particles seek the position where the net force is as small as possible. Once the particles reach such state, they oscillate back and forth around this equilibrium and these oscillations produce the oscillations in the VACF (WILLIAMS *et al.*, 2006). Figure 2.9 provides a comparison for the behavior of the VACF for $t = 100$ time steps for four different trajectories.

As we can see in Figure 2.9, three of the four trajectories's velocities' decorrelate over time whereas the green trajectory does precisely the opposite. This is observed because the trajectory of the particle with Direct motion with diffusion was simulated in such a way that its velocities are dependent of one another.

⁸ Notice that we compute the scalar product for different velocities for all N particles.

⁹ In Statistical Mechanics, we say the velocity **decorrelates** over time.

¹⁰ These oscillations are very similar to the ones we may observe in a Damped Harmonic Oscillator.

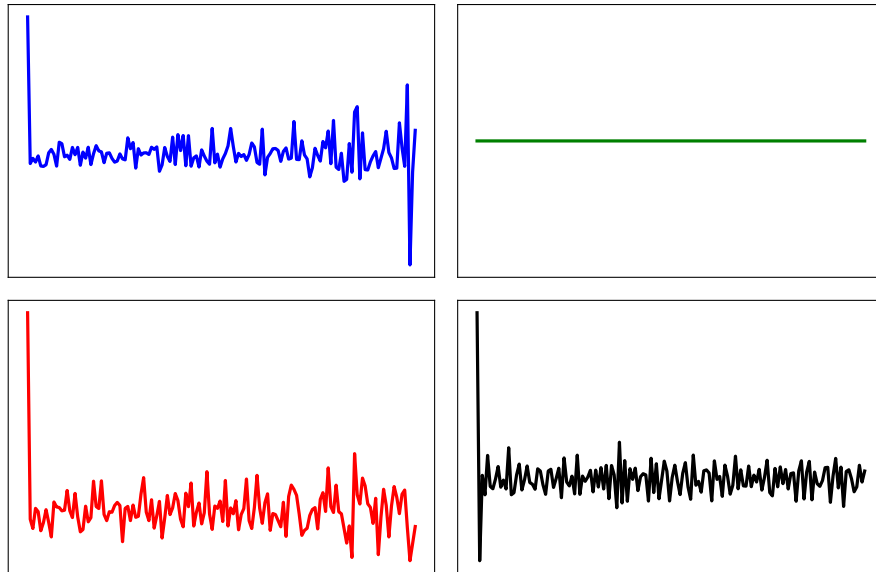


Figure 2.9 – This image shows a comparison for different behavior of the Velocity Autocorrelation Function. The blue, green, black and red lines refer to trajectories presenting the four basic diffusion types: Confined diffusion, Direct motion with diffusion, Anomalous diffusion and Normal diffusion.

Fonte: The author.

We can make use of the VACF to discuss the next feature present in TrajPy, the **Green-Kubo Relation** for the diffusion coefficient D . According to Frenkel e Smit (2002), this relation is defined as

$$D = \int_0^{\infty} d\tau \langle v(\tau)v(0) \rangle, \quad (2.17)$$

where $\langle v(\tau)v(0) \rangle$ is the VACF.

Equation (2.17) is the **Green-Kubo Relation** for the diffusion coefficient D . It connects the macroscopic concept of diffusion with the microscopic movement of the particles. There are Green-Kubo relations for multiple **transport coefficients**, such as diffusion. Simply put, transport coefficients measure how quickly a perturbed system returns to equilibrium (ALLEN; TILDESLEY, 1989). So in the case of the diffusion D , the system might be in equilibrium and a perturbation such as a drop of ink in a glass of water will make the particles of the system rearrange themselves so that the equilibrium is once again achieved¹¹.

The next feature recently implemented in TrajPy is a **statistical description of the velocity**. This feature takes as input the values of the velocity of each particle over time and computes the mean, median, mode, variance, standard deviation, the distance between the largest and smallest value of the velocity (called range), the kurtosis and skewness of the distribution - for a complete discussion about these concepts and the equations we are about to provide, see Agarwal (2006). Although these calculations are quite simple, they allow us to describe the velocity distribution using some basic concepts from Statistics.

¹¹ The movement of particles towards the opposite direction of the concentration gradient described in Subsection 1.2 is a general way to say that the system will always seek equilibrium.

The mean $\langle v^{(i)} \rangle$ of the velocity is the usual expected value defined as

$$\langle v^{(i)} \rangle = \frac{1}{n} \sum_{k=1}^n v_k^{(i)}, \quad (2.18)$$

where n is the total number of data points¹² for the velocity, $i = x, y, z$ and k indicates the k -th value of the velocity. From this point, we will omit the vector notation for simplicity.

The median is the data point where one half of the data points lies above it and the other half below it, when put in ascending order. The median $m_v^{(i)}$ of the velocity is mathematically defined as

$$m_v^{(i)} = \begin{cases} v_{\frac{n+1}{2}}^{(i)} & \text{if } n \text{ is odd,} \\ \left[v_{\frac{n}{2}}^{(i)} + v_{\frac{n}{2}+1}^{(i)} \right] / 2 & \text{if } n \text{ is even.} \end{cases} \quad (2.19)$$

where $v_{(n+1)/2}^{(i)}$ is the $(n+1)/2$ -th velocity data point in the i -th direction.

The mode is the velocity value that is repeated the most along the list of values for the velocity in each direction- in cases where there is more than one mode, we consider all of them. To compute this quantity, we used a Python module called Statsmodels (SEABOLD; PERKTOLD, 2010).

The velocity variance $var_v^{(i)}$ describes the average squared distance between the velocity data points and the mean $\langle v^{(i)} \rangle$ and is mathematically defined as

$$var_v^{(i)} = \frac{1}{n} \sum_{k=1}^n (v_k^{(i)} - \langle v^{(i)} \rangle)^2. \quad (2.20)$$

The standard deviation $std_v^{(i)}$ of the velocity, on the other hand, is the square root of the variance so that the average distance described in Equation (2.20) can be written in the same units as the velocity itself. The standard deviation calculated for the velocity data points in the i -th direction is defined as

$$std_v^{(i)} = \sqrt{var_v^{(i)}}. \quad (2.21)$$

The range of the velocity $ran_v^{(i)}$ is defined to be the absolute value of the difference between the largest and the lowest observed values for the velocity in the i -th direction. It is simply defined as

$$ran_v^{(i)} = |\max_v^{(i)} - \min_v^{(i)}|. \quad (2.22)$$

The skewness $S^{(i)}$ measures the lack of symmetry of the velocity probability distribution function when the center point is considered. The more different the left and right sides of the distribution are, the higher the skewness. For a Gaussian distribution, the skewness is null, since perfect symmetry is observed. In the case of a distribution with a short left tail and a long right one, the skewness will be positive. If the opposite is observed, the skewness will be negative. The skewness is defined as

$$S^{(i)} = \frac{(var_v^{(i)})^{3/2}}{std_v^{(i)}}. \quad (2.23)$$

¹² In this case, we are considering that the velocity is composed of discrete data points.

The kurtosis measures the shape of the peak and tails of the velocity probability distribution function. In TrajPy, we use what is called excess kurtosis, where the calculations are done in such a way that a Gaussian Distribution has a kurtosis of zero - hence the value of three being subtracted in Equation (2.24). When looking at a distribution, if its peak is sharper and its tails contain more data than the ones of a Normal Distribution, the kurtosis will have a positive value. On the other hand, if the distribution presents a flat peak with long tails, its associated kurtosis will be negative. The excess kurtosis $K^{(i)}$ is defined as

$$K^{(i)} = \frac{(\text{var}_v^{(i)})^{3/2}}{(\text{std}_v^{(i)})^4} - 3. \quad (2.24)$$

Figure 2.10 provides an illustration for the behavior of probabilities distribution functions and their respective values for the skewness and kurtosis. Notice that both of these measurements give us insight about the overall behavior of the velocity.

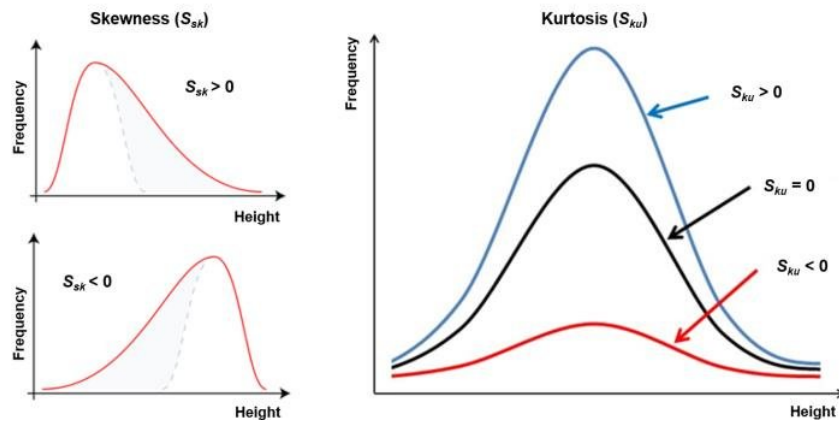


Figure 2.10 – This figure provides the possible outcomes for both the skewness and kurtosis of the intensity - here referred to as “height” - distribution function of pixels in a topographic image. The velocity distribution function follows the same idea regarding the values of kurtosis and skewness.

Fonte: (BONYÁR, 2015).

The last attribute implemented in TrajPy is the **frequency spectrum** performed by the Fourier Transform - for a thorough discussion regarding the Fourier Transform, see Arfken (1985). The integral transformation called Fourier Transform is one of the many of its kind and produces relevant information about any function $f(t)$, as long as the function is differentiable everywhere in its time domain. The Fourier Transform is defined as

$$g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{i\omega t} dt, \quad (2.25)$$

where $g(\omega)$ is the analogous of $f(t)$ in the frequency domain ω .

We are particularly interested in the fact that the Fourier Transform reveals what are the trigonometric functions $\sin(t)$ and $\cos(t)$ present in $f(t)$ if represent such function as linear combinations of both trigonometric functions. For an application of the Fourier Transform in the analysis of hand tremors, see San-Segundo *et al.* (2020).

In TrajPy, we use this linear combination of trigonometric functions to get access to the underlying frequencies of vibration of the system along with its amplitude. To do that, we use

the Python numerical library Numpy (HARRIS *et al.*, 2020), specifically the Fast Fourier Transform algorithm (MUQRI; WILSON; SHAKIB, 2015). By using this mathematical tool, we are able to provide the trajectory of any particle, access the trigonometric linear combination and compute the following attributes: the highest frequency and its associated amplitude, the mean frequency, the frequencies above a given threshold specified by the user along with their amplitudes, the frequency spectrum and amplitudes.

To compute these features, we apply the Fast Fourier Transform algorithm to every data point of the trajectory and store the information. The dominant frequency μ is the highest frequency and, by extension, the dominant amplitude is the amplitude of the wave with frequency μ . The mean frequency is computed by taking the mean of all frequencies, whereas the main frequencies are calculated once we provide a threshold and select the frequencies that are higher than or equal to this threshold along with their corresponding amplitudes. The frequency spectrum is a list of all frequencies calculated for every trajectory data point and the amplitudes are the absolute value of each frequency in the spectrum. We take the absolute value of the frequencies so that we can get the amplitudes. Figure 2.11 shows an application of the Fast Fourier Transform algorithm.

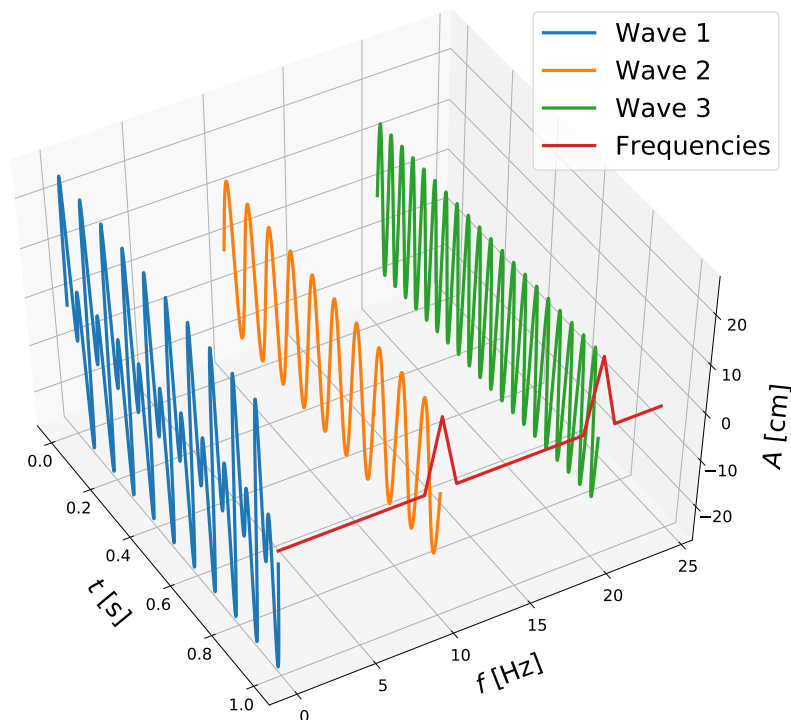


Figure 2.11 – This figure shows a wave (blue) that is formed by the sum of two different waves (green and yellow). The red line presenting two peaks is a visual representation of the effects of applying the Fast Fourier Transform algorithm to the blue wave, once the units of measurement of the red line is Hz, the unit of frequency. The corresponding amplitudes of each wave can be verified by looking at the Amplitude axis.

Fonte: The author.

2.2 Simulating trajectories for the four basic motion types

In this section, we will present the way we simulate the four motion types we have discussed in Section 1.2. The algorithms used to simulate each motion type can be seen in Code A.2 of Attachment A.

Let us begin by describing the simulation procedure for Normal Diffusion, as proposed by Michalet (2010). In order to simulate a particle in ϕ dimensions whose trajectory is modeled by Normal Diffusion, we simulate ϕ one-dimensional brownian motions and return the time steps and associated one-dimensional trajectories. The ϕ one-dimensional brownian motions are then combined as spatial components x, y and z , where each component is scaled to have unitary length.

In simple terms, for each of the one-dimensional brownian motions, we draw a random number a out of a standard Gaussian distribution with unitary mean and standard deviation. Then we set $u = (0.5 - a)dx$, where dx is the maximum step length the particle is allowed to take in any direction, and check whether a is greater than or equal to the absolute value of $C(u)$, where C is the Concentration we have defined in Equation 1.6. If any of the previous conditions is true, then the particle takes a step of length u in a random direction. This process is repeated for all the given time steps for all ϕ one-dimensional brownian motions.

As stated in the beginning of this chapter, we may change a set of parameters for each simulation engine for the four basic motion types. In the case of Normal Diffusion, we may change the number of displacements the particle will present, the number of dimensions¹³, the maximum step length dx , the initial position, the diffusion coefficient D and the time step t that will be used in Equation 1.6.

The next motion type we will discuss is the Confined Diffusion. To simulate it, we specify the number of displacements, the number of dimensions, dx , the initial position, D and the time step. For each displacement out of the total number of displacements determined as a parameter, the particle will present Normal Diffusion behavior for 100 displacements. If the particle, after those 100 displacements, remains inside a sphere whose radius is also defined as a parameter, the new position of the particle will be the position associated with the 100th displacement. Regarding the radius of confinement of the sphere, we suggest using positive values up to 0.5, since the larger the radius, the more similar to Normal Diffusion the Confinement Diffusion becomes.

The third motion type is the Direct Motion with Diffusion, which is a combination of a simple integration of a constant velocity v over time and Normal Diffusion. The direct motion with diffusion is given by

$$\vec{r}(t) = \vec{\Gamma}(t) + \vec{r}_0 + \int_{t_0}^T \vec{v} dt, \quad (2.26)$$

where $\vec{\Gamma}(t)$ is the position generated from Normal Diffusion at time t , \vec{r}_0 is the initial position of the particle determined as a parameter in the algorithm used to simulate Direct motion with diffusion and \vec{v} is the constant velocity also determined as a parameter. The remaining parameters are the number

¹³ We may set the number of dimensions as the product between the number of particles we wish to simulate and the number of dimensions for each particle or loop over the number of particles simulating one at a time.

of displacements, the number of dimensions and the time step.

The combination of Direct Motion and Normal Diffusion yields different behaviors, depending on the magnitude of the velocity v in Equation 2.26. For $v \approx 0.01$, the overall diffusion resembles a Normal Diffusion. For $v > 1.0$, the same overall diffusion becomes more similar to the Direct Motion modeled by Equation 2.26. Figure 2.12 provides a simple comparison of four different trajectories presenting the behavior of Direct Motion with Diffusion for four different values of the velocity v .

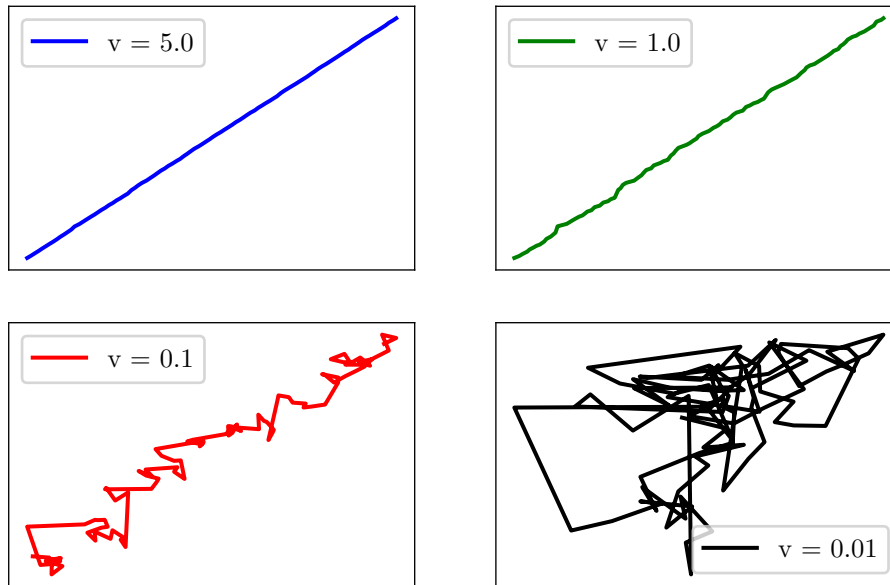


Figure 2.12 – This figure presents the effect of the velocity's magnitude on the overall behavior of the Direct Motion with Diffusion. The smaller the velocity, the more similar to a Random Walker the trajectory becomes.

Fonte: The author.

In the literature, there are different methods to simulate Anomalous Diffusion, such as obstruction of the particle's path, fractional brownian motion and continuous-time random walk (SANDEV; METZLER; CHECHKIN, 2018). In TrajPy, we have chosen to simulate this diffusion by means of the Weierstrass-Mandelbrot function¹⁴ (BERRY; LEWIS; NYE, 1980) for the simplicity associated with its computational calculation.

Put in simple terms, the Weierstrass-Mandelbrot function is continuous in its entire domain, differentiable nowhere and mathematically defined as

$$W(t) = \sum_{n=-\infty}^{\infty} \frac{\cos(\phi_n) - \cos(\gamma^n t^* + \phi_n)}{\gamma^{n\beta/2}}. \quad (2.27)$$

In Equation 2.27, ϕ_n is a phase randomly chosen from the interval $[0, 2\pi]$ for every value of the summation index n , $\gamma = \pi^{1/2}$ and $t^* = 2\pi t/N$, where N is the total number of steps the particle may take in a simulation.

Every individual position of the trajectory is obtained by performing the summation in Equation 2.27 from $n = -8$ to $n = 48$, since, as determined by Saxton (2001), adding more positive and

¹⁴ In the present moment, we are solely interested in the practical aspect of such function from a computational perspective. The Weierstrass-Mandelbrot function presents a connection with fractals and fractional brownian motion, but its origins and meaning are beyond the scope of the current research project.

negative values has little to no effect in the position, i.e., the numerical value of the position does not change much if the summation is performed over a larger interval than $[-8, 48]$ for n .

Regarding the parameters, they are the number of displacements, the number of dimensions, the time step and the exponent β . We advise the user to use values of β within the interval $(0, 2]$, where we must exclude the value of $\beta = 1$, since this specific value would yield a Normal Diffusion, as we have discussed previously.

In the next section, we present the two graphical-user interfaces TrajPy offers to the user. The first allows the user to upload a file containing the trajectory of interest and compute the attributes discussed in Subsections 2.1.1 and 2.1.2. The second uses the idea of Computer Vision to perform object tracking on video to extract the $x - y$ coordinates of any moving object present in a video. As we will discuss in details, this graphical-user interface was developed to be a first step to a general solution to a technological bottleneck in the process of drug discovery.

2.3 Using Graphical-User Interfaces - attributes calculations and animal tracking for drug discovery

The first graphical-user interface was implemented by Moreira-Soares (2020) and its objective is to assist users who are interested in computing the attributes discussed in Subsections 2.1.1 and 2.1.2 but are not familiar with Python programming. Figure 2.13 shows what this interface looks like.

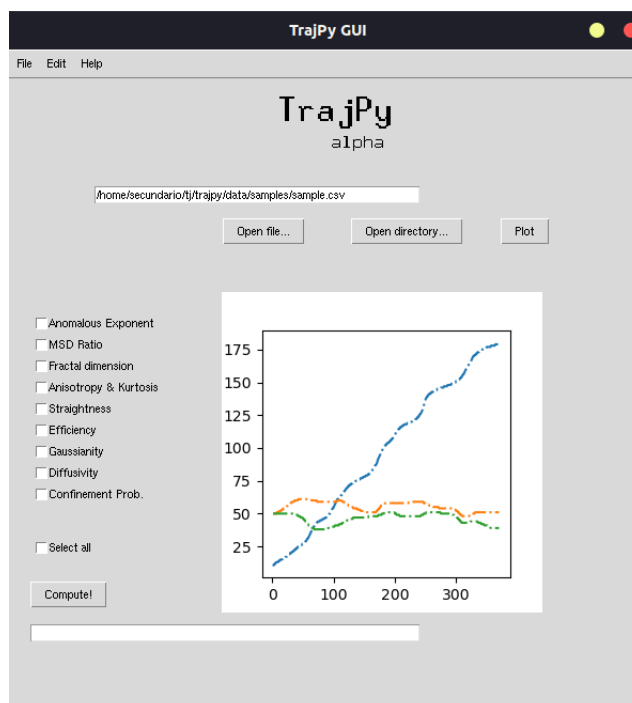


Figure 2.13 – This figure presents the first graphical-user interface. The graph shown in this figure is not generated automatically, the user must generate it by using the “plot” button. We will discuss the function of each button shortly.

Fonte: The author.

Let us present the function of each button shown in Figure 2.13. The “Open file...” and “Open

directory...” buttons allow the user to load the file that contains the trajectory the user wishes to analyse. Once the user clicks either of these buttons, a window will pop up on the screen and the user should use this screen to select the trajectory file. The “Plot” button, once the trajectory file is loaded, allows the user to make a graph of each coordinate of the trajectory, separated by colors - making this graph is optional.

As soon as the trajectory file is loaded, the user may check the boxes corresponding to each attribute to be computed and press the “Compute!” button. The graphical-user interface will then use the attributes informed in the check boxes and compute the attributes. The numerical results will be shown in the white box below the “Compute!” button. The order the user checks each box does not matter, the results will be shown in the order they appear from the top to the bottom of the check boxes. The “Select all” button allows the user to select all check boxes at once.

The second graphical-user interface, another contribution we have made to TrajPy, was developed to be used as an object tracking computational tool, where we have used the Python library called OpenCV to implement important ideas of Computer Vision (BRADSKI; KAEHLER, 2008). Our main goal for this graphical-user interface is to provide to the scientific community, specially to scientists involved in drug discovery research, a possible solution to the significant bottleneck regarding effective chemobehavioral screening in drug discovery and neurotoxicology (HENRY; WLODKOWIC, 2020). The algorithm we developed to perform single object tracking can be seen in Code A.3 of Attachment A.

Before we delve on into the functionality of the second graphical-user interface, let us briefly discuss the bottleneck we previously mentioned, so that the reason we proposed this interface becomes clear. According to Berdigaliyev e Aljofan (2020), the experimental use of animals for *in vivo* testing is one of the steps a specific drug must take before being put on the market to be sold as medication. The *in vivo* stage basically consists of analysing the overall behavior of an animal under the treatment of a drug in order to assess how the animal is affected - according to Bhargava, Pullaguri e Bhargava (2022), living cells are also used in a similar way. Depending on the outcomes of the assessment, i.e., if the observed effect was desired or not, scientists involved in the process either take the next step and move on to the clinical trials or go declare that the drug failed.

In order to analyse the behavior of the animal, specifically its trajectory, object tracking techniques have been used as a tool to assist the analysis process. The bottleneck emerges, according to Wlodkowic (2022), from the fact that there is a lack of automated video-based multiple animal tracking techniques that can aid in multi-dimensional behavioral analysis. With this in mind, we proposed the second graphical-user interface in combination with the attributes discussed in Section 2.1 as a possible solution to this bottleneck.

The tracking algorithm we propose was set up to perform single object tracking only. We are currently working on a generalized algorithm that allows the user to track multiple objects at once. If we achieve such a goal, TrajPy will become a general framework for trajectory analysis in the broadest sense of the word.

To perform the technique of object tracking, the user must provide a video feed¹⁵ of the object in motion. The graphic-user interface will use such video to extract the x and y coordinates of the object and store them on an external trajectory file along with the time t .

Figure 2.14 shows the second graphic-user interface.

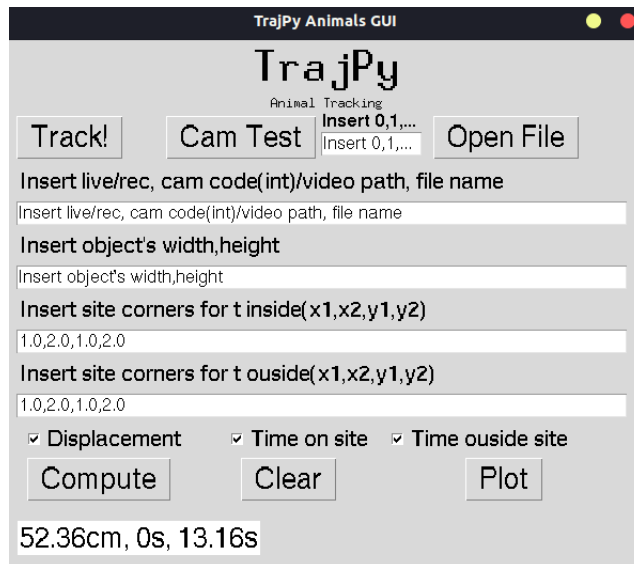


Figure 2.14 – This figure presents the second graphic-user interface where the user is able to perform the technique of object tracking.

Fonte: The author.

In Figure 2.14, there are five entry fields where the user must provide the relevant information in order to use this interface. The first, to the right of the “Cam Test” button, allows the user to discover the code of the camera¹⁶ that will be used to perform the tracking of the object on a live video. The user must provide integer numbers starting from zero and pressing the “Cam Test” button. If a live video pops up on the screen showing the video being captured by the camera, the integer number previously informed is the “cam code” the user should use to perform live object tracking. In order to close such window, the user must press the key “q” on the keyboard. If the user wishes to provide a previously recorded video, there is no need to use the “cam code”.

The second entry field is located below the “Track!”, “Cam Test” and “Open File” buttons. In this specific entry field, the user must inform whether the object tracking will be performed on a live video (by writing the word “live”) or on a previously recorded video (by writing the word “rec”). Then, the user must inform either the “cam code” we discussed earlier, when using a live video feed, or the “video path”, when providing a video that has been stored in the user’s computer. To inform the “video path”, the user must inform the full path to the folder where the video is located, e.g., **/home/videos/example.mp4**. For simplicity, all videos must be in the **.mp4** format. Lastly, the user must inform the name of the file where the trajectory of the object will be stored. This “file name” will be used to name both the **.txt** file containing the x and y coordinates along with the time t and a video

¹⁵ This video may be a live video feed or a previously recorded video.

¹⁶ This is particularly useful in situations where there are more than one camera involved. The user may need to use different cameras in different situations, so mapping each camera to its “cam code” is important.

where the object tracking technique will be shown¹⁷. If a full path is provided for the “file name”, both the **.txt** file and the **.mp4** video will be saved in the specific location provided by the full path. For clarity, we provide this example where we have informed the algorithm we are going to track an object using a live video using a specific camera and we wish to save the trajectory file inside the “test” folder in a file named “example_file”: **live,0,/home/documents/test/example_file**. Notice that we have not used any of the **.txt** or **.mp4** extensions after informing the name of the file we wish to use to save the trajectory and the video.

The third entry box is located below the second. In this entry box, the user must inform the size of the object being tracked. The “width” must be the largest dimension of the object and the “height” the remaining dimension. These dimensions must be provided in centimeters separated by commas with no space between each dimension and the comma. For clarity, we provide the following example: **5.5,4.9**. Notice that we have not informed that the unit of each dimension is given in centimeters.

The fourth and fifth entry boxes are located below the third, one after the other, as we can see in Figure 2.14. In each of them, the user must provide the x and y coordinates of a squared or rectangular region. In the fourth entry box, the user must inform the corner coordinates of the region where the time spent by the object **inside** this region will be computed. In the fifth entry box, the coordinates of the corners of another region must be informed. However, the time the object spent **outside** this region will be computed. Notice that these two regions might be different from each other, since the user may want to compute the time spent at the center of the general area the object is moving in and on the borders of such general area. For clarity, Figure 2.15 provides an illustration of the two different regions along with the position of the origin the user must take into account when measuring the coordinates of the corners of each region.

In order to track the object of interest, the user must fill the second and third input fields with the required information and press the “Track!” button. Once the tracking technique starts, a new window will pop up on the user’s screen showing the object being tracked by a rectangle around it. The x and y coordinates of the center of this rectangle will be the approximate coordinates of the object that will be stored in the **.txt** file. To stop the object tracking, the user must press the letter “q” on the keyboard.

Once the user finishes tracking the object of interest, both the **.txt** and **.mp4** files containing the object’s trajectory and the tracking on video, respectively, will be saved on the informed folder. To open the trajectory file, the user must use the “Open File” button, locate the **.txt** file and open it. Once this file is loaded into the graphic-user interface, the user is able to compute the total displacement of the object, the time spent by the object inside the region bounded by the coordinates informed and the time outside a region bounded by its corners’ coordinates, as we previously discussed. To compute these features, the user must check the associated check boxes and press the “Compute” button. To clear the results, the “Clear” button should be used.

Lastly, the “Plot” button reads the already loaded trajectory file and makes two graphs. The

¹⁷ This video showing the tracking algorithm will show a rectangle around the moving object. The user may want to see this video in order to check if the tracking technique was performed correctly. If, for some reason, the algorithm failed, the user will be able to notice that the rectangle is not being drawn around the object at all times.

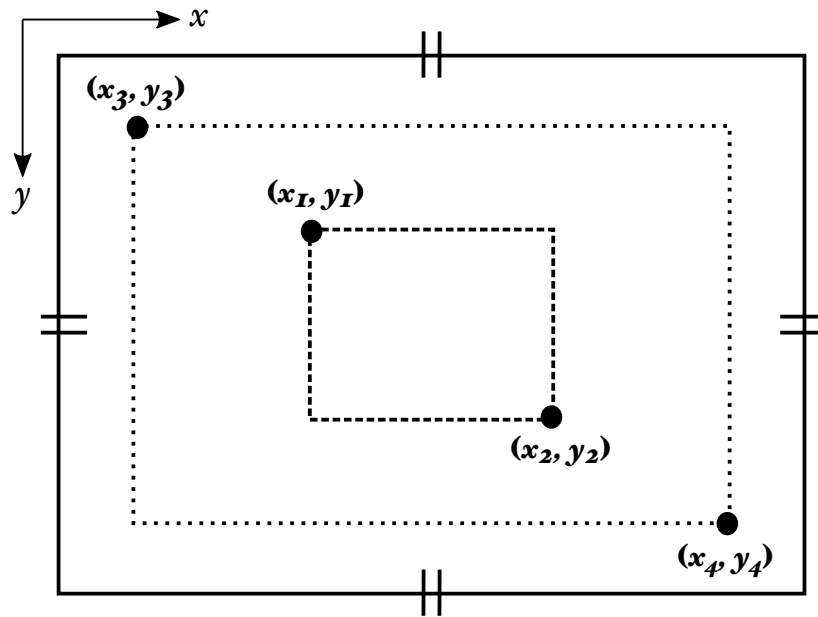


Figure 2.15 – This figure presents the two regions the second graphic-user interface uses in order to compute the time spent inside the region bounded by x_1, y_1, x_2 and y_2 and outside the region bounded by x_3, y_3, x_4 and y_4 . The largest region presenting two parallel lines on each side represents the arena the object is allowed to move in. Notice the x and y axis orientation on the top left corner.

Fonte: The author.

first is the time evolution of the object's position represented by a continuous line that changes colors according to the value of the time t . The second graph is a two-dimensional histogram that computes the amount of data points for each combination of x and y in the trajectory file. The amount of data points is indicated by a change in color in the histogram. In order to save these graphs, the user must click the button represented by a floppy disk in the top left corner of Figure 2.16.

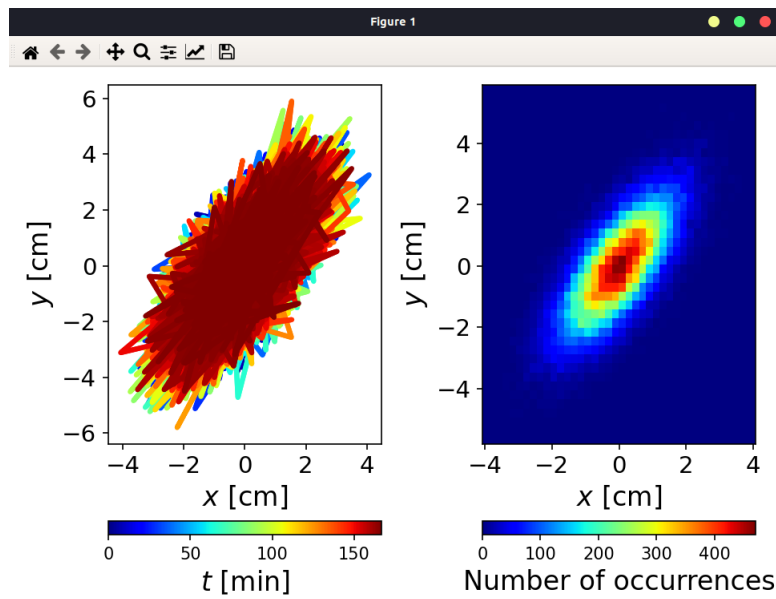


Figure 2.16 – This figure presents the two graphs provided by the second graphical-user interface, once the user presses the “Plot” button. The graph on the left shows the time evolution of the trajectory indicated by different colors. The right graph is a two-dimensional histogram that shows the number of data points for every combination of x and y . The buttons on the top left corner, from left to right, allow the user to: reset the graphs to their original size, in case the user changed it, change figures when more than one is generated (this is not the case for this graphic-user interface), move a specific graph to a different direction as the user pleases, zoom in in a specific region of one graph, change the overall figure’s settings (we advise the user to not do it), change different configurations of the graphs, such as colors, axis limits and so on (this is a great option to personalize your graphs) and, finally, save the graphs as a single figure.

Fonte: The author.

3 TRAJECTORY ANALYSIS AND DIFFUSION CLASSIFICATION OF TWO-DIMENSIONAL POLY-MER RINGS

In this chapter, we will present the physical aspects of the system we will analyse the dynamics of in Section 3.1. Then, we will provide and discuss the results for each attribute we have obtained in Section 3.2. Lastly, in Section 3.3 we offer a brief introduction to the idea of Machine Learning and discuss the results for the diffusion classification.

3.1 The physical system of interest

3.1.1 General Overview

Amorphous solids, as glasses, probably are the materials most known and used by humans. Obsidian volcanic glass was used for prehistoric tools and weapons. Now we can easily design glasses with desired mechanical or optical properties on an industrial scale. Yet, a microscopic understanding of the glassy state of matter remains a challenge for statistical physicists (BIROLI, 2007). Amorphous solids can be formed by two distinct process: cooling or jamming. In the first one, called glass transition, the temperature is lowered, leading the dynamics to slow down to the point where the system can no longer relax and becomes rigid. A similar state can be obtained when the density is increased - this is the second process, the jamming transition. This transition can be observed in hard spheres and in many other systems, such as foams, emulsions and granular matter. (LIU; NAGEL, 1998), (MAJMUDAR *et al.*, 2007), (LIU; NAGEL, 2010), (ALTIERI, 2019).

Amorphous materials jamming transition remains an extremely challenging problem in Solid State Physics (BIROLI,), (JAEGER, 2015), (CHARBONNEAU *et al.*, 2017), (BRITO; LERNER; WYART, 2018), although there are many technological applications – besides glass related technologies – that relays in the proper understanding of the jamming transition. For instance, we usually relate the word "jamming" with traffic jamming. In this sense, in Social Physics the jamming transition is of special interest for predict and prevent cars and airplanes jamming (NAGATANI, 1994), (NAGATANI, 1998), (HELBING, 2001), (LACASA; CEA; ZANIN, 2009), (IKEDA *et al.*, 2020), (RAMANA; SAI; JABARI, 2021), (JIANG *et al.*, 2022). Also, pedestrian jamming has been investigated by Social Physicists, including recent works on effects of corridors, wall-following (as in the case of blackout), disabilities and social distance (MURAMATSU; IRIE; NAGATANI, 1999), (KRAMER; WANG, 2021), (FU *et al.*, 2021), (ZHOU *et al.*, 2022), (THOMPSON *et al.*, 2022), (WANG *et al.*, 2022). Even information traffic jamming in wifi networks - and how to prevent it - are being studied (OHIRA; SAWATARI, 1998), (LU; WANG; WANG, 2014), (HAN *et al.*, 2020), (WANG *et al.*, 2021a), (DUY; HUONG; HONG, 2022). Granular jamming has been identified as a fundamental mechanism for the development of robotic grippers, soft robotics, wearable haptics and wearable robots, bioprinting, reversible construction and fabrics with tunable properties and exoskeleton (BROWN *et al.*, 2006), (STELTZ *et al.*, 2010), (AEJMELAEUS-LINDSTRÖM *et al.*, 2016), (BRANCADORO *et al.*, 2019), (PLOSZAJSKI *et al.*, 2019), (CHENG *et al.*, 2020), (FITZGERALD; DELANEY; HOWARD, 2020), (LIU *et al.*, 2021), (WANG *et al.*, 2021b), (GÖTZ *et al.*, 2022), (JADHAV *et al.*, 2022).

Therefore, jamming transitions have been widely studied in physics and material science. However, their importance in a number of biological processes, including embryo development, tissue homeostasis, wound healing, and disease progression, has only begun to be recognized in the past few years (SADATI *et al.*, 2014), (OSWALD *et al.*, 2017), (LENNE; TRIVEDI, 2022). Specifically, cell migration is an adaptive process that depends on and responds to physical and molecular triggers. Moving cells sense and respond to tissue mechanics and induce transient or permanent tissue modifications, including extracellular matrix stiffening, compression and deformation, protein unfolding, proteolytic remodelling and jamming transitions. The fact that biological systems can undergo rigidity/-jamming transitions is attractive, as it would allow these systems to change their material properties rapidly and strongly (HELVERT; STORM; FRIEDL, 2018). However, how they are being regulated, and what their physiological relevance might be is still being debated (HANNEZO; HEISENBERG, 2022). Therefore, a better understanding about the dynamics of biological system in jamming transitions are of special interest. The cell plasticity itself, including deformability and contractility, plays a key role in the cell migration process (FRIEDL; ALEXANDER, 2011), (HAEGER *et al.*, 2014), (BOEKHORST; PREZIOSI; FRIEDL, 2016), (KANG *et al.*, 2021), (ILINA *et al.*, 2020), (ALMAGRO *et al.*, 2022). In this sense, we will apply the TrajPy package to explore the behavior of a simple drop-like model for biological cells as the pressure is increased and it overcomes the jamming transition.

3.1.2 The Model and Simulation Details

3.1.2.1 The 2D drop-like model for deformable cells

Biological cells can deform and change its shape to overcome obstacles. In fact, the mechanical properties of the cells can play an major role in their mobility. In a recent work (MOREIRA-SOARES *et al.*, 2020), a simple drop-like model was proposed, and here we show the 2D version of this model. Briefly stated, each cell is represented by a bead-spring polymeric, where each bead is connected to a ghost particle located at the cell center of mass (CM) by a harmonic potential,

$$U^{\text{cm}} = \frac{1}{2}k_{\text{cm}}(R - r_{\text{b-cm}})^2. \quad (3.1)$$

Here, k_{cm} is the spring constant, R is the cell radius and $r_{\text{b-cm}}$ is the distance between the polymer bead and the cell CM. Two neighbour beads in the external ring are also connected by a harmonic potential,

$$U^{\text{ring}} = \frac{1}{2}k_{\text{ring}}(r_{ij} - \sigma)^2, \quad (3.2)$$

where r_{ij} is the distance between the two neighbour beads i and j , σ is the bead diameter and $k_{\text{ring}} = 30.0\epsilon/\sigma^2$ is the spring constant, and $\epsilon = k_B T$ is the thermal energy.

Two beads i and j that do not belong to the same ring will interact by the purely repulsive Weeks-Chandler-Andersen (WCA) potential (WEEKS; CHANDLER; ANDERSEN, 1971),

$$U^{\text{WCA}} = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 + \frac{1}{4} \right], & r_{ij} \leq r_c \\ 0, & r_{ij} > r_c. \end{cases} \quad (3.3)$$

which is nothing but a standard 6-12 Lennard-Jones (LJ) potential shifted by ε and cut at its minimum, $r_c^{WCA} = 2^{1/6}\sigma$. The total interaction for the i th bead is, therefore,

$$U = U^{\text{cm}} + U^{\text{ring}} + U^{\text{WCA}} . \quad (3.4)$$

In this model, the cell deformation is ruled by the value of k_{cm} . In this sense, we can see how the cell dynamics in the jamming transition will be affected by its deformation just by varying k_{cm} . In Figure 3.1 we show a depiction of our model, and snapshots from the trajectories for low and high values of k_{cm} .

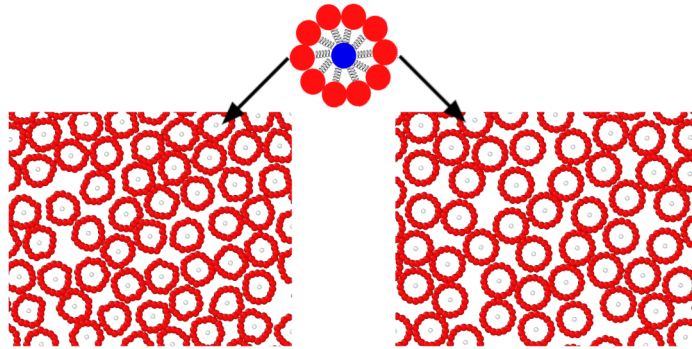


Figure 3.1 – Upper: Schematic depiction of the bead spring ring model.
Lower: system snapshots at low and high values of k_{cm} .

Fonte: The author.

In this work, all the quantities are computed and presented in the standard Lennard Jones (LJ) reduced units (ALLEN; TILDESLEY, 2017),

$$r^* \equiv \frac{r}{\sigma}, \quad \rho^* \equiv \rho\sigma^3, \quad \text{and} \quad t^* \equiv t \left(\frac{\varepsilon}{m\sigma^2} \right)^{1/2}, \quad (3.5)$$

for distance, density of particles and time, respectively, and

$$p^* \equiv \frac{p\sigma^3}{\varepsilon} \quad \text{and} \quad T^* \equiv \frac{k_B T}{\varepsilon} \quad (3.6)$$

for the pressure and temperature, respectively, where $\sigma = 1.0 \mu\text{m}$.

3.1.2.2 Simulations Method and Details

Our initial system consists of $n_{\text{cell}} = 400$ cells randomly displaced in a square box with initial size $L = 210$. A distance criteria is used to avoid overlaps in the cell insertions, and the initial velocities are obtained from a Gaussian distribution at temperature $T = 0.5$. To keep the initial temperature constant and to mimic local hydrodynamic effects, we perform Langevin Dynamics (LD) simulations, a special case of classical Molecular Dynamics (MD) simulations (ALLEN; TILDESLEY, 2017).

To use MD based techniques, we have to solve the classical equations of motion for conservative systems,

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = -\vec{\nabla}_{\vec{r}_i} U \quad (3.7)$$

where m_i is the mass of the i -th particle and the right side of the equation is the force applied to the atom due to interactions with the other particles. Therefore, MD consists in solving the differential equation 3.7 for each particle of the studied system. Once the initial configuration has been defined, the program is ready to perform a simulation step. Given a final simulation time, the program will perform a series of calculations while performing the steps until reaching the final time. Typically, the program calculates the resulting force on each particle of the system. This step depends on the interaction potentials chosen for the system. Next, we integrate numerically the classical equations of motion, thus obtaining the new positions and speeds of the particles in a time $t + \delta t$, where δt is the time increment for each simulation step. After integrated the equations, the program calculates properties of interests in the system under study, such as energy, pressure, temperature, diffusion, radial distribution function and so on. Once the time limit has been reached, the program stops. The trajectory can be then directly obtained at each time step, or at multiples time steps.

While the MD method is conservative, i.e., keeps the number of particles N , the system area A (once we are analyzing a 2D system) and energy E fixed. On the other hand, in the LD method, at each simulation step the particles of the system suffer the action of a viscosity force proportional to their speeds and the coefficient of friction of the system, γ and random forces originated from a white noise term, which simulate the effect of a continuous series of collisions between the particles of the system and the particles of the solvent. Thus, we insert these two terms into the calculation of the resulting force on each particle,

$$\vec{F}_R = -\vec{\nabla}U - m\gamma\vec{v}_i + \vec{W}_i(t) \quad (3.8)$$

where W is the random force originated from a white noise term (ALLEN; TILDESLEY, 2017), that from the fluctuation-dissipation theorem is related to the system temperature through the second moment of distribution,

$$\langle W_i(t)W_j(t') \rangle = \delta_{ij}\delta(t-t')6k_bT\gamma \quad (3.9)$$

where δ_{ij} is the Kronecker delta, $\delta(t-t')$ is the Dirac delta, k_b is the Boltzmann constant, T is the temperature and γ is the coefficient of friction between the system and the thermostat. This expression indicates to us that random forces are completely uncorrelated at different times. The first term of the equation 3.8 corresponds to the forces derived from the interaction potential between the particles in the system. This extra forces control the system temperature by allowing the energy exchange. Then, LD simulations are in the NVT , or Canonical, Ensemble.

To simulate the jamming transition the system particle density, $\rho = n_{\text{cell}}/A$ has to be increased along a isotherm. This can be done by increasing n_{cell} or decreasing A in the NAT ensemble. Alternatively, we can increase the system pressure, what would lead to a decrease in A and a increase in ρ . To do so, simulations were performed at constant pressure p by employing the Andersen barostat (ANDERSEN, 1980). Briefly stated, the barostat rescales the system size - and the particles coordinates - according to a coupling to external pistons that acts in the system with pressure p . With this, we will perform simulations in the NpT Ensemble for temperatures ranging from $p = 0.10$ to $p = 0.25$, with interval of $\delta p = 0.01$.

While would be possible to write a LD computer code from the scratch, in this work we perform the simulations using the ESPREesSo¹ package version 3.3.1 (LIMBACH *et al.*, 2006),(ARNOLD *et al.*, 2013). The main code of this version is written in C, and a TCL script is necessary as input for the executable. The script employed to perform this simulations is shown in Appendix 2. Each simulations consists in to run a isotherm along the pressure interval for a cell with a value of $k_{\text{cm}} = 10.0, 25.0, 50.0, 75.0, 100.0, 150.0, 250.0, 300.0,$ and 500.0 . After the system been randomly created, we run 1×10^6 timesteps with $\delta t = 0.01$ to thermalize the system, and extra 1×10^6 timesteps for equilibration. After that, 1×10^7 timesteps were performed in the results production stage, in such way that the trajectory file consists in 401 frames. After that, we increase the system pressure by δp , run the equilibration and production steps, and repeat until we reach the highest pressure. The algorithm written using ESPREesSo can be seen in Code A.4 of Attachent A.

3.1.3 Results and Discussion

3.1.3.1 Thermodynamic and Structural Analysis

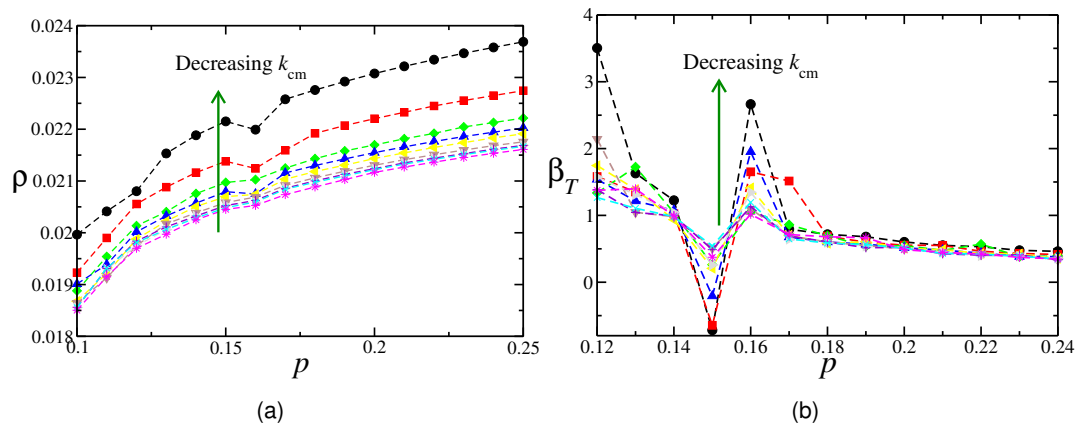


Figure 3.2 – (a) Density ρ as function of pressure p^* and (b) Isothermal compressibility β_T for all values of k_{cm} , ranging from $k_{\text{cm}} = 10.0$ (black spheres) to $k_{\text{cm}} = 500.0$ (magenta stars).

Fonte: The author.

We start our discussion showing how the system thermodynamic properties varies as rings with distinct k_{cm} are compressed. In Fig. 3.2(a) we show that deformable cell will have higher density compared to rigid rings - what is expected, once as the pressure increases the cell with lower k_{cm} will be compressed, leading to a smaller mean box size \bar{L} . Interestingly, there is a discontinuity at $p^* = 0.16$, that is more pronounced as we decrease k_{cm} . The derivative of density as function of pressure along a isotherm gives the isothermal compressibility,

$$\beta_T = \frac{1}{\rho} \left(\frac{\partial \rho}{\partial p} \right)_T, \quad (3.10)$$

¹ ESPResSo stands for *Extensible Simulation Package for Research on Soft Matter Systems*.

whose behavior is shown in Figure 3.2(b). It indicates that at this pressure there is a phase transition. To identify exactly the characteristics of the transition we must analyze the system structural and dynamical behavior.

Once the interactions are pairwise important quantities can be calculated explicitly as integrals involving the radial distribution function (RDF) $g(r)$ (HANSEN; MCDONALD, 2006). To check for long range translational ordering using the RDF as basis we evaluate the cumulative two-body entropy (KLUMOV; KHRAPAK, 2020)

$$C_{s2}(R) = -\pi \int_0^R [g(r_{ij}) \ln(g(r_{ij})) - g(r_{ij}) + 1] r_{ij} dr_{ij} . \quad (3.11)$$

Here r_{ij} is the distance and R is the upper integration limit. At this distance $|C_{s2}|$ converges for fluid or amorphous phases and diverges for the ordered phases. It is important to address that the two-body excess entropy is a structural order metric which connects thermodynamics and structure, not a thermodynamic property of the system. The translational ordering can also be characterized by the translational order parameter, defined as (ERRINGTON; DEBENEDETTI, 2001)

$$t_s \equiv \int_{\xi}^{\xi_c} |g(\xi) - 1| d\xi, \quad (3.12)$$

where $\xi = r\bar{\rho}^{1/2}$ is the interparticle distance r divided by the mean separation between pair of particles. $\xi_c = \bar{L}\bar{\rho}^{1/2}/2$ is the cutoff distance, with \bar{L} the mean box side and $\bar{\rho}$ the mean density. For an ideal gas, $g(\xi) = 1.0$ and t_s vanishes. For ordered systems, $g(\xi) \neq 1.0$ once there is translational long range order.

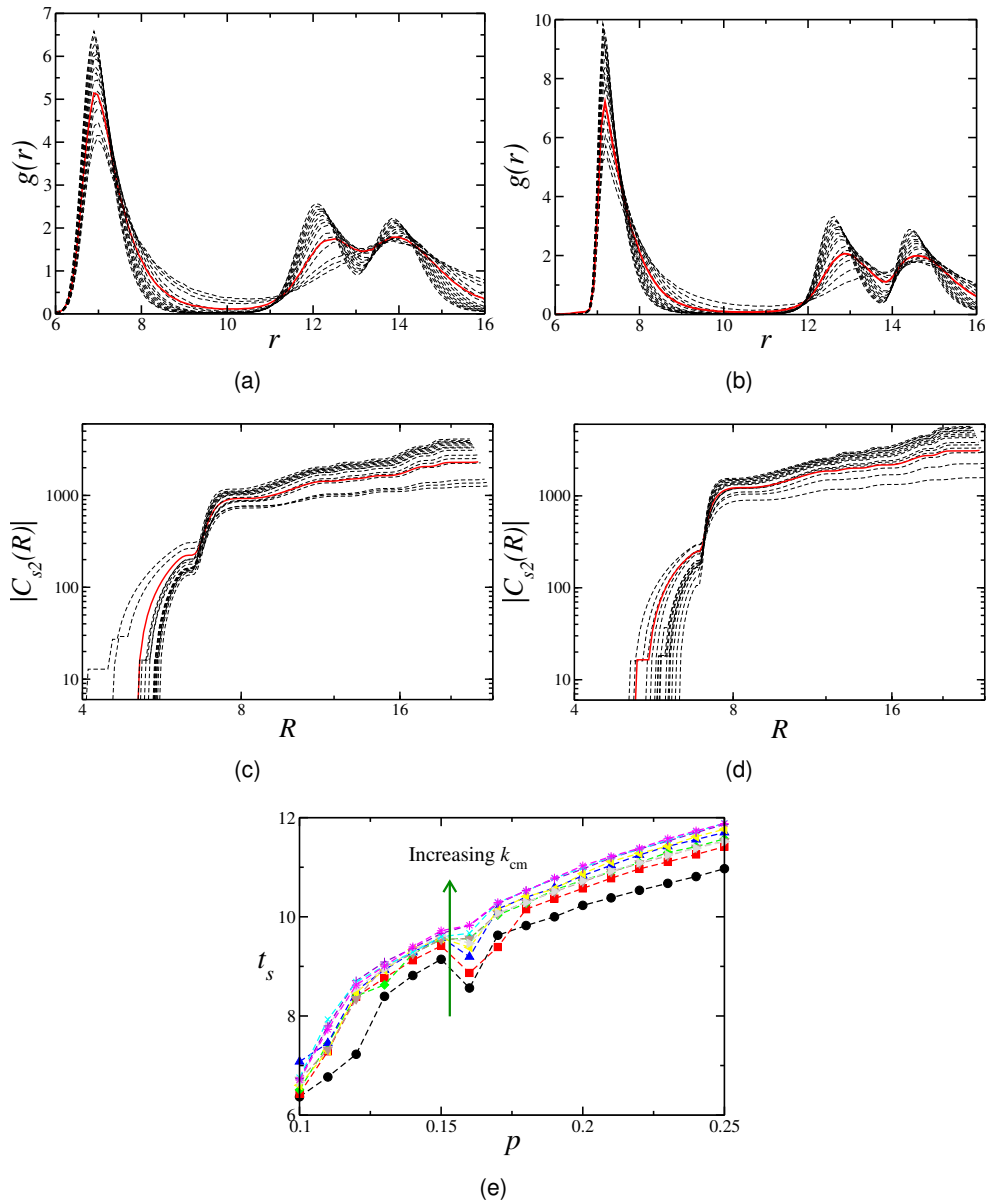


Figure 3.3 – Cell center of mass radial distribution functions for (a) $k_{cm} = 10.0$ and (b) $k_{cm} = 500.0$. Cumulative two-body entropy (c) $k_{cm} = 10.0$ and (d) $k_{cm} = 500.0$. The dashed black lines are the curves for all pressures from 0.10 to 0.25, except the red solid curve, that corresponds to the $p^* = 0.16$ point. (e) Translational order parameter for all values of k_{cm} , ranging from $k_{cm} = 10.0$ (black spheres) to $k_{cm} = 500.0$ (magenta stars).

Fonte: The author.

Is this sense, we show in Fig 3.3(a) and (b) the RDF between the cells center of mass for the models with $k_{cm} = 10.0$ and $k_{cm} = 500.0$, respectively. We can notice the deformation effect by looking to the shape of the first peak in the $g(r)$. Once the cell diameter is 6.0 and the bead diameter is 1.0, the hard-core effective separation between two cells center of mass is 7.0. For deformable cell, Fig 3.3(a), there is a clear penetration in this hard-core separation. This indicates the softness of the cells at low values of k_{cm} . As expected, rings with higher values of k_{cm} tends to keep the circular shape, behaving more like a hard-core disks. Fig 3.3(b) shows how there is no penetration between two cells, even at high pressures.

While the first coordination shell provided information about the cell softness, analyze the

longer range ordering can help to understand the system symmetry. To this end, we can look the cumulative two-body entropy behavior in Fig 3.3(c) and (d) for cell models with $k_{\text{cm}} = 10.0$ and $k_{\text{cm}} = 500.0$, respectively. For the $k_{\text{cm}} = 10.0$ case there is a clear separation between a disordered phase and a more ordered phase - the red line corresponds to the $p^* = 0.16$ point, where β_T indicated a transition. Therefore, there is a structural change at this pressure. Going back to the RDF curves, Figs 3.3(a) and (b), we can see that at $p^* = 0.16$ the second and third coordination shells have the same height. Then this transition corresponds to a long range ordering, what is corroborated by the increase in the translational order parameter, Fig. 3.3(e).

3.2 Trajectory analysis

Let us begin the trajectory analysis by discussing the way we will approach it. As we have mentioned in the previous section, each system is formed by 400 rings simulated in a combination of pressure p^* and spring constant k^* . For each polymer ring of every system, we have calculated all but one attribute² of those described in Subsections 2.1.1 and 2.1.2.

We will then use the mean value of every feature computed - except for the MSD by Ensemble average - to represent the average behavior of the system regarding the features themselves. Our main interest is to observe what happens to the numerical results of each feature for different values of p^* and k^* . The results will be presented and discussed in such a way that we wish to develop some intuition for both the meaning of each feature and the behavior of the system itself.

Let us begin by presenting the MSD by Ensemble average. As discussed in Subsection 2.1.1, the MSD by Ensemble average calculates the average quadratic distance from the origin at time t for the 400 polymer rings for each system. We can use such quantity to describe the average motion of the polymer rings and gain some insight about the overall behavior of the system.

If we recall Equation (2.1), to compute the MSD by Ensemble average for each timestep we sum the squared distances from the origin for each polymer ring and divide it by the number of polymer rings. By analysing this a bit more carefully, we see that the MSD can increase, decrease or maintain the same value from timestep to timestep. When it increases, it means that, on average, the polymer rings got further away from the origin. If the MSD decreases, the polymer rings - again, on average - get closer to the origin. And if it remains the same, it indicates that the polymer rings did not get either further away from or closer to the origin. For subsequential timesteps, the MSD by Ensemble average may oscillate around a specific value. When that happens, it means that the polymer rings, on average, move away from the origin and then get closer to it - and the process is repeated. We begin by analysing the MSD by Ensemble average for $p^* = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k^* = 10.0$ presented in Figure 3.4.

In Figure 3.4 we notice that the MSD, in each timestep, gets smaller as the pressure increases. This means that the polymer rings, on average, can no longer move away from the origin by a

² We have not computed the frequency spectrum for every ring, since we are interested in the dynamics of the central bead of each ring.

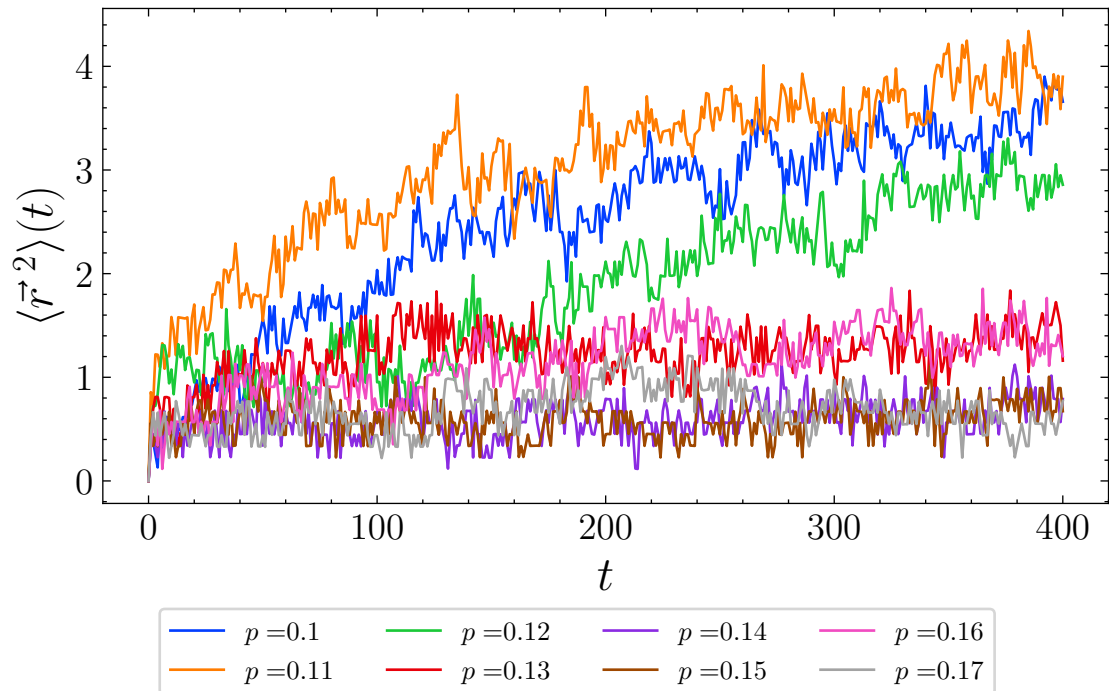


Figure 3.4 – This figure provides the MSD by Ensemble average for pressure values of $p^* = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k^* = 10.0$ for all 401 timesteps.

Fonte: The author.

significant amount once the pressure is high enough³. For $p^* = 0.1, 0.11$ and 0.12 , we see that the MSD increases overall, even though we observe eventual slight decreases in subsequential timesteps. For these three systems simulated the average quadratic distances at the last timestep are quite similar to one another: $3.65, 3.89$ and 2.85 , respectively. If we take the square root of these values and multiply the result by $\sigma = 1 \times 10^{-6} \text{m}$ to calculate the average squared distances given in meters, we get $1.91^{-6} \text{m}, 1.68^{-6} \text{m}$ and 1.91^{-6}m , respectively.

If we analyse the curves for the MSD for the other values of pressure in Figure 3.4 we see that the MSD increases a bit for the first 50 timesteps, approximately, and then it simply oscillates around specific values for each curve. As we previously stated, oscillations of this kind indicate that the polymer rings, on average, move away from the origin by a certain amount, get closer to it, and repeat the same process for the next timesteps. This implies that the polymer rings for the systems simulated for the remaining values of pressure shown in Figure 3.4 can not move away from a certain region of the system, i.e., they are confined in that region.

We now provide the result of the MSD for the systems simulated for the remaining values of pressure at $k^* = 10.0$. Given that the overall behavior of the MSD is very similar⁴ for $p^* = 0.18, 0.19, 0.2, 0.21, 0.22, 0.23, 0.24$ and 0.25 , we provide a simple curve - in this case, for $p = 0.25$ - to show the results in a way that the graph is visually comprehensible in Figure 3.5.

³ In the case of Figure 3.4, we might say that pressures larger than 0.12 are considered high enough to impose physical limitations on the system.

⁴ The MSD basically oscillates around a specific value and may change the value it oscillates around of, in some occasions. However, it presents virtually the same overall behavior given than the possible changes of values regarding the oscillations are minimal.

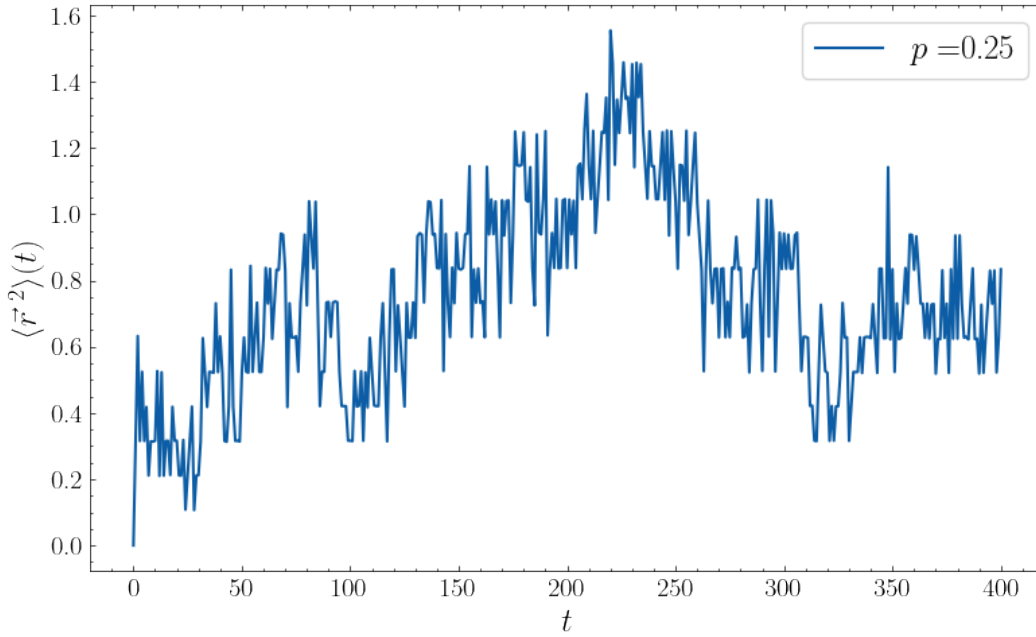


Figure 3.5 – This figure provides the MSD by Ensemble average for $p^* = 0.25$ and $k^* = 10.0$ for all the 401 time steps.

Fonte: The author.

Figure 3.5 shows that the MSD by Ensemble average for $p^* = 0.25$ oscillates around 0.4 and then 0.6 throughout the timesteps. This means that the polymer rings, on average, are confined in a given region of the system. We can say that, given the that the MSD oscillates around a certain value and then this value increases a bit, the polymer rings may have moved in a way that they became confined in a region that is a little further away from the origin. To visualize this, we have simulated Confined Motion with a radius of confinement $r_c = 0.5$ using the simulations we have discussed in Section 2.2. This way, we can demonstrate that the MSD by Ensemble average does in fact oscillate around a specific value and that the change from 0.4 to 0.6 we mentioned earlier indicates the movement we described. Figure 3.6 provides the comparison for the MSD presented in Figure 3.5 and the MSD calculated for the Confined Motion we have simulated. We provide the graphs for $p^* = 0.1$ through $p^* = 0.17$ for the remaining values of k in Attachment B. The MSD for higher pressures follows very similar behaviors to the one presented in Figure 3.5, so we will not provide the results for simplicity.

The second attribute we will discuss is the **MSD ratio**. As stated in Subsection 2.1.1, we expect the MSD ratio to be equal to zero for Normal Diffusion, positive for Confined or Anomalous Diffusion and negative for Direct motion with Diffusion. Figure 3.7 provides the mean value of the MSD ratio for each system. In every figure of this section, the points represent the mean of the attribute for every combination of p^* and k^* and the error bars represent the standard deviation associated with that mean.

As we can see in Figure 3.7, the MSD ratio increases as the pressure p^* gets larger for every value of k^* . Somewhat surprisingly, the average behavior of the MSD ratio does not seem to change a considerable amount for each value of k , which indicates that the effects of the pressure on the

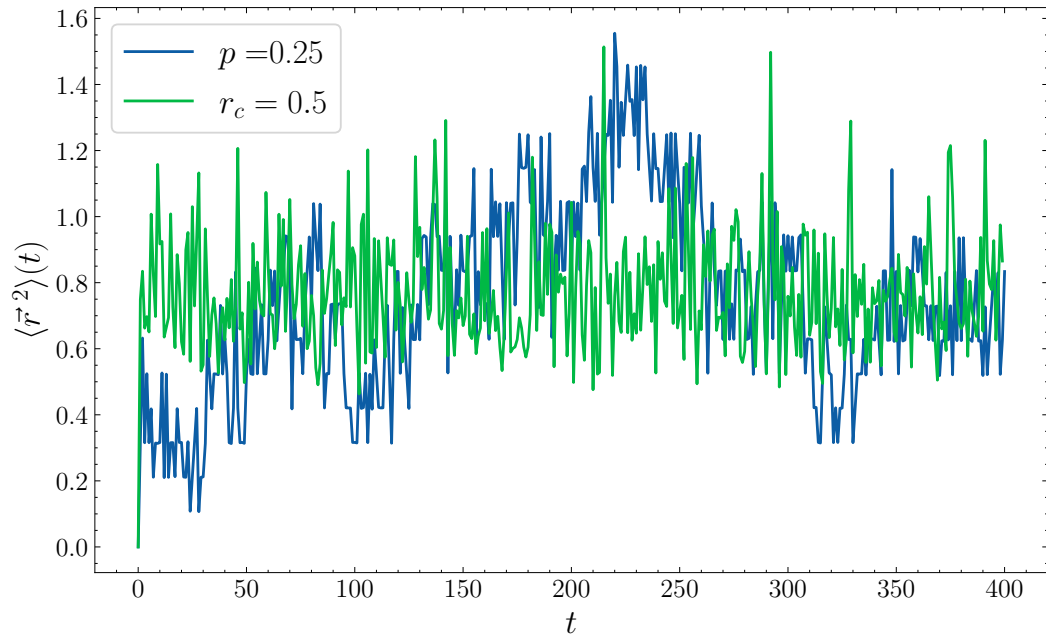


Figure 3.6 – This figure provides a comparison between the MSD by Ensemble average for $p^* = 0.25$ and $k^* = 10.0$ and the MSD calculated from simulated Confined Motion with a radius of confinement $r_c = 0.5$.

Fonte: The author.

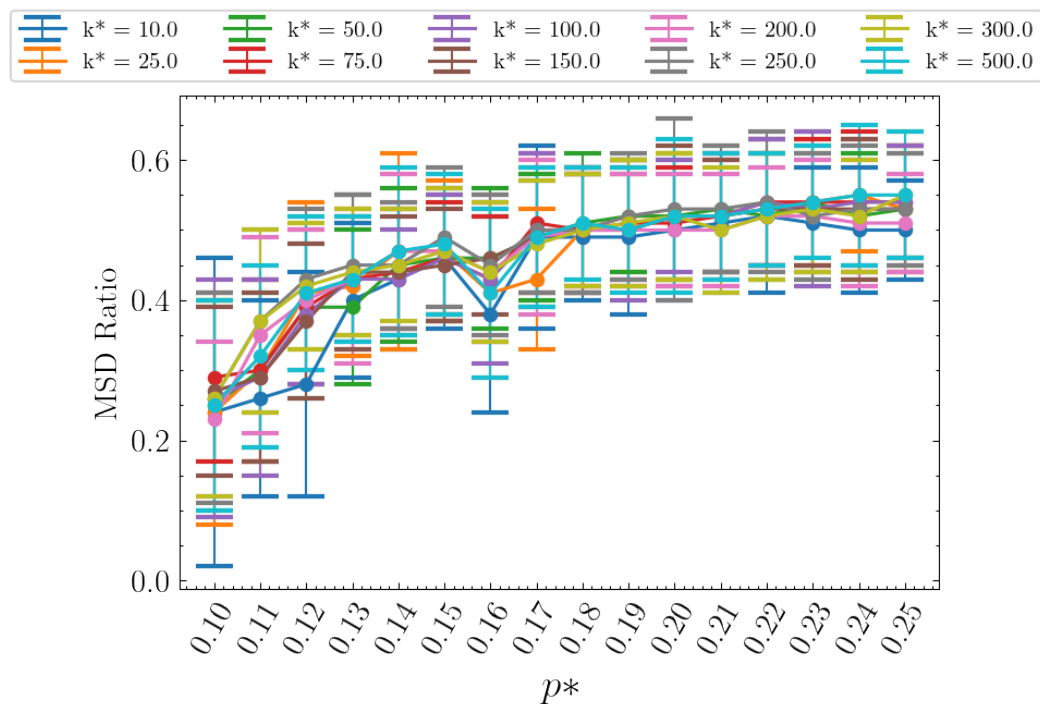


Figure 3.7 – This figure provides the average and the standard deviation of the MSD ratio for every system of interest. We compared those averages for every pressure p^* and spring constant k^* .

Fonte: The author.

system are greater than those of the springs' constants.

In addition, the overall increase of the attribute observed in Figure 3.7 indicates that the

particles of each system could be represented by Random Walkers⁵ for low p^* - which, in turn, are associated with Normal Diffusion - and by confined Random Walkers for high values of p^* - and, by extension, the diffusion would be classified as Confined or Anomalous Diffusion. For $p^* = 0.16$, we observe that the MSD ratio decreases momentarily. This is connected, in some way, to the phase transition we discussed in Subsection 3.1.3.1. To fully understand this, we will simulate the same systems for different temperatures in future works.

The next attribute we will discuss is the **Anomalous Exponent**. As discussed in Subsection 2.1.1, the Anomalous Exponent is calculated as the angular coefficient of a linear regression in a $\log - \log$ graph of the MSD as a function of time. The Anomalous Exponent β dictates the time dependency of the MSD $\langle r^2(t) \rangle \propto t^\beta$. If $\beta = 1$, the MSD is represented as a straight line and the diffusion is classified as Normal Diffusion. On the other hand, $\beta \neq 1$ represents either Subdiffusion or Superdiffusion, as we have stated in Subsection 2.2. Figure 3.8 shows the average behavior of the Anomalous exponent for every combination of p^* and k^* .

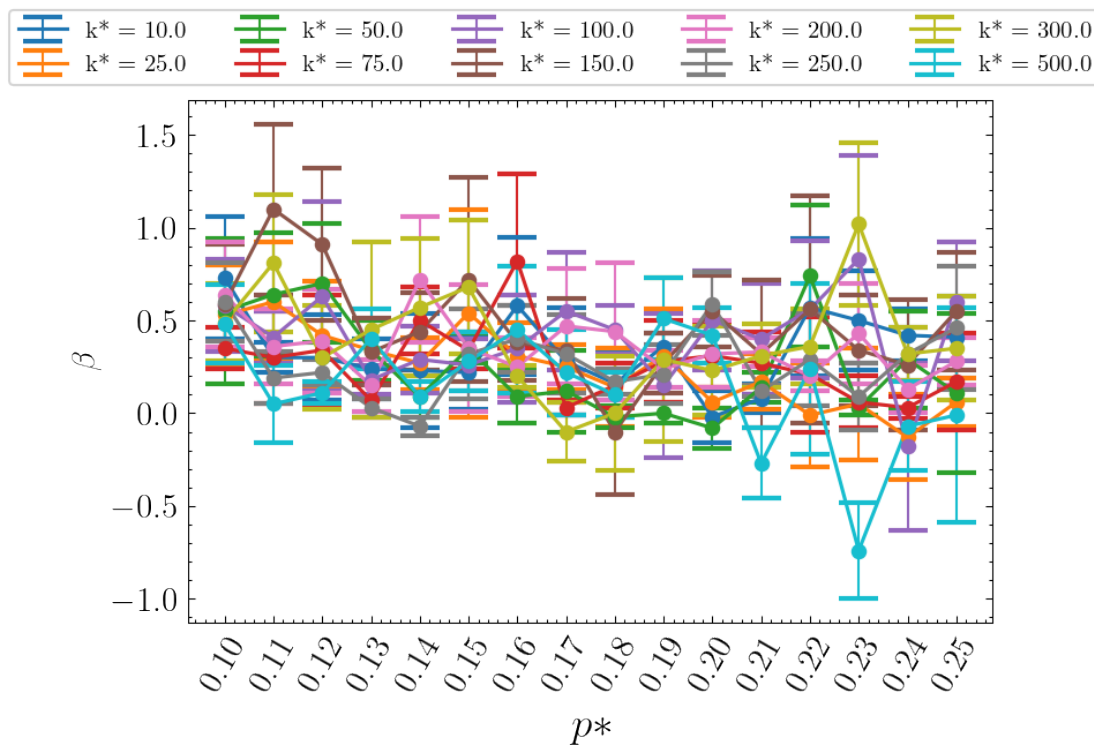


Figure 3.8 – This figure provides the average and standard deviation of the Anomalous Exponent for every system of interest.

Fonte: The author.

As we can see in Figure 3.8 the values of the Anomalous Exponent basically oscillate around 0.4 in the top graph and 0.5 in the bottom graph. In addition, the time dependency of the MSD lies within the Normal and Subdiffusion regimes. In fact, we see that, for low p^* , the highest value of β is usually achieved, which again indicates that the systems undergo a dynamical transition. Regarding the negative values of β we have obtained,

⁵ We will use “Random Walker” and “Brownian Motion” interchangeably from now on.

The next attribute to be discussed is the **Fractal Dimension**. As we have established in Subsection 2.1.1, the Fractal Dimension - in the context of TrajPy - provides a way for us to describe how irregular the trajectory is. If the trajectory is a straight line, its Fractal Dimension is equal to 1. If the trajectory resembles a Random Walker, the Fractal Dimension gets close to 2. Lastly, if the trajectory undergoes some physical constraint, the Fractal Dimension gets larger than 2. Figure 3.9 shows the results we have obtained for each system.

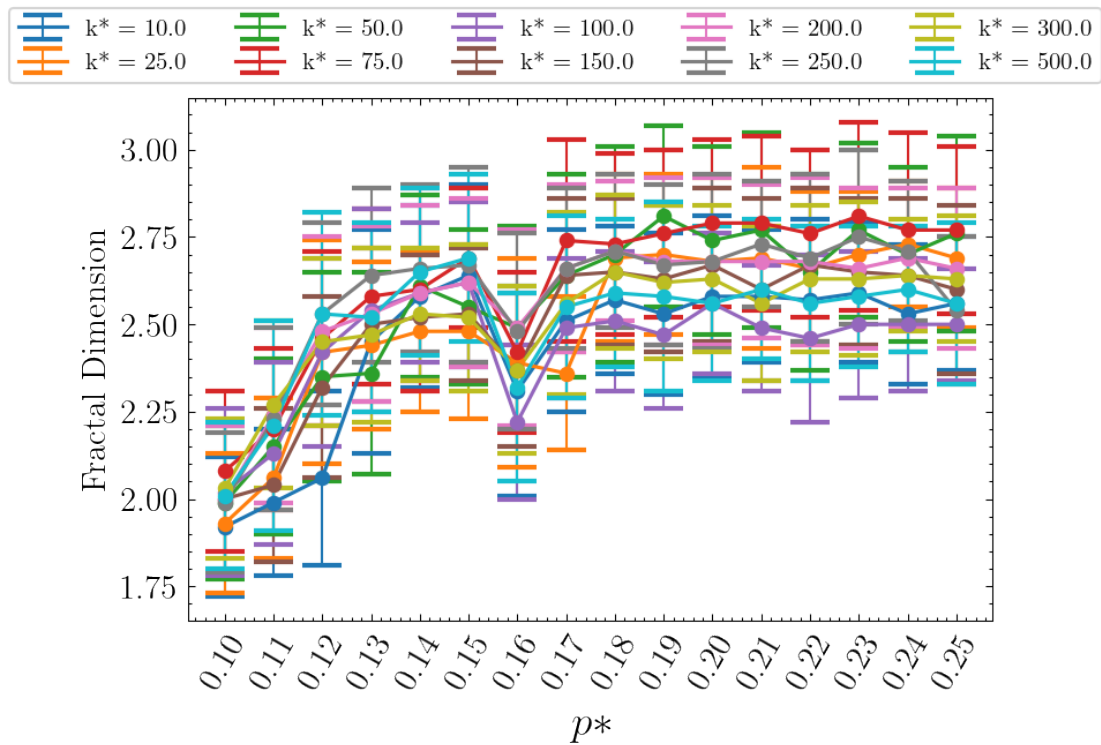


Figure 3.9 – This figure provides the average and standard deviation of the Fractal Dimension for every system of interest.

Fonte: The author.

In Figure 3.9 we can observe that for low p^* , the Fractal Dimension is equal to 2 and as the pressure gets higher, it increases to values larger than 2. This means the mean value of the Fractal Dimension for each system indicates a dynamical transition from Normal Diffusion to Confined or Anomalous Diffusion. Regarding the effects of the spring constant k^* , its effects on the systems are once again small compared to the effects of the pressure p^* . The decrease in the Fractal Dimension observed for $p = 0.16$ will be analysed separately in future works, since we must simulate the systems with different values of temperature in order to make sense of this decrease.

The next attribute we will discuss is the **asymmetry**. As we have stated in Subsection 2.1.1, the asymmetry measures the possible tendency for a preferred direction in any trajectory. When computed, the asymmetry for a symmetric trajectory is null, since there is no direction of preference - the Random Walker satisfies this property. The closer to one the asymmetry becomes, the closer to a straight line along an axis the trajectory becomes. Figure 3.10 provides the results we have obtained.

In Figure 3.10, we observe only small values for the asymmetry of each system. This means that, on average, the particles of each system can be described as Random Walkers. The dynamical

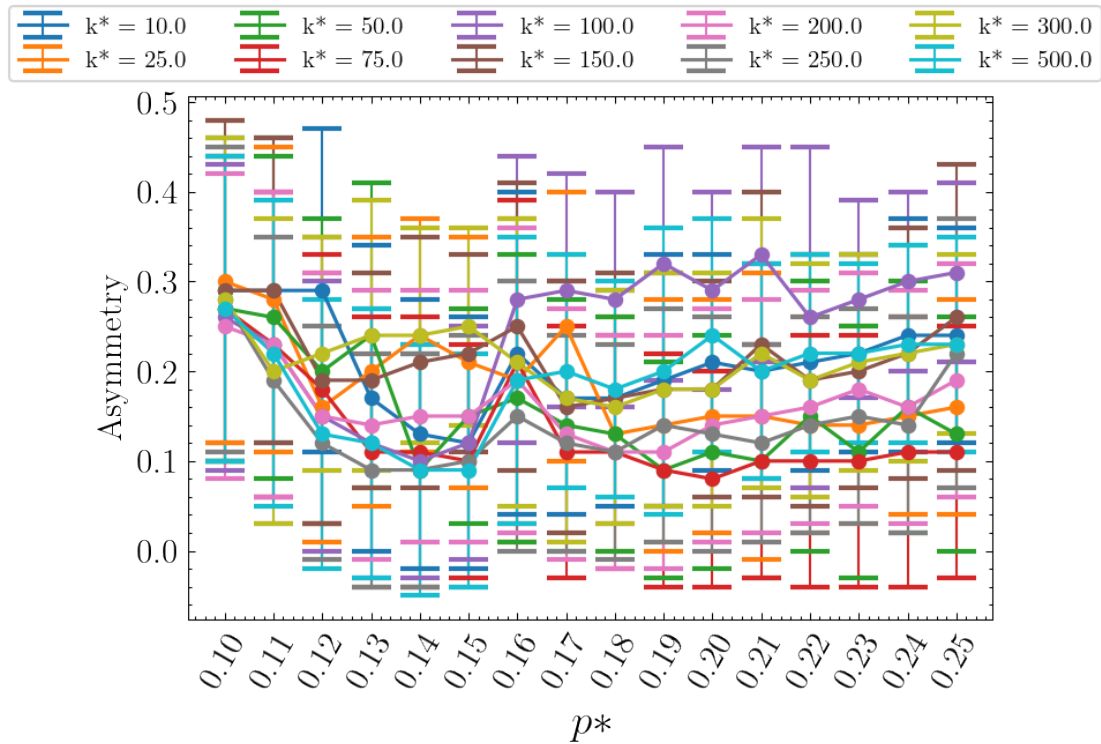


Figure 3.10 – This figure provides the average and standard deviation of the Asymmetry for every system of interest.

Fonte: The author.

transition we have been observing so far is not apparent in this case since Normal and Confined Diffusion both present similar values for the asymmetry. For reference, see Figure 2.3.

But once we recall this fact, we immediately realise we should use other attributes in combination with the asymmetry in order to once again confirm the transition from Normal to Confined or Anomalous Diffusion. This comes to show the importance of Feature Engineering, which is the basic idea of TrajPy from a Machine Learning perspective.

The next attribute we will discuss is the **anisotropy**. As we have discussed in Subsection 2.1.1, the anisotropy measures the distribution of positions regarding its symmetry to the origin. When computed, the anisotropy is null for a symmetric position distribution and yields 1 when the symmetry is not observed. Figure 3.11 shows the mean anisotropy for each system.

As we can see in Figure 3.11, the mean anisotropy for each system oscillates around 0.5. This means that the position distributions are somewhat symmetric in relation to the origin, as expected for Random Walkers. For reference, see Figure 2.4.

Although the dynamical transition we previously observed is not apparent, the Anisotropy will be useful once we decide to classify the diffusion of the systems using Machine Learning. Some of the attributes of TrajPy - the Anisotropy being one of them - provide relevant information about specific aspects of the systems in a particular way, which may be useful in certain situations. The fact that we cannot observe dynamical transitions in this case should not be a surprise, since the process of Feature Engineering allows us to analyse the systems from different perspectives, where some may be more useful than others when we are looking for specific information.

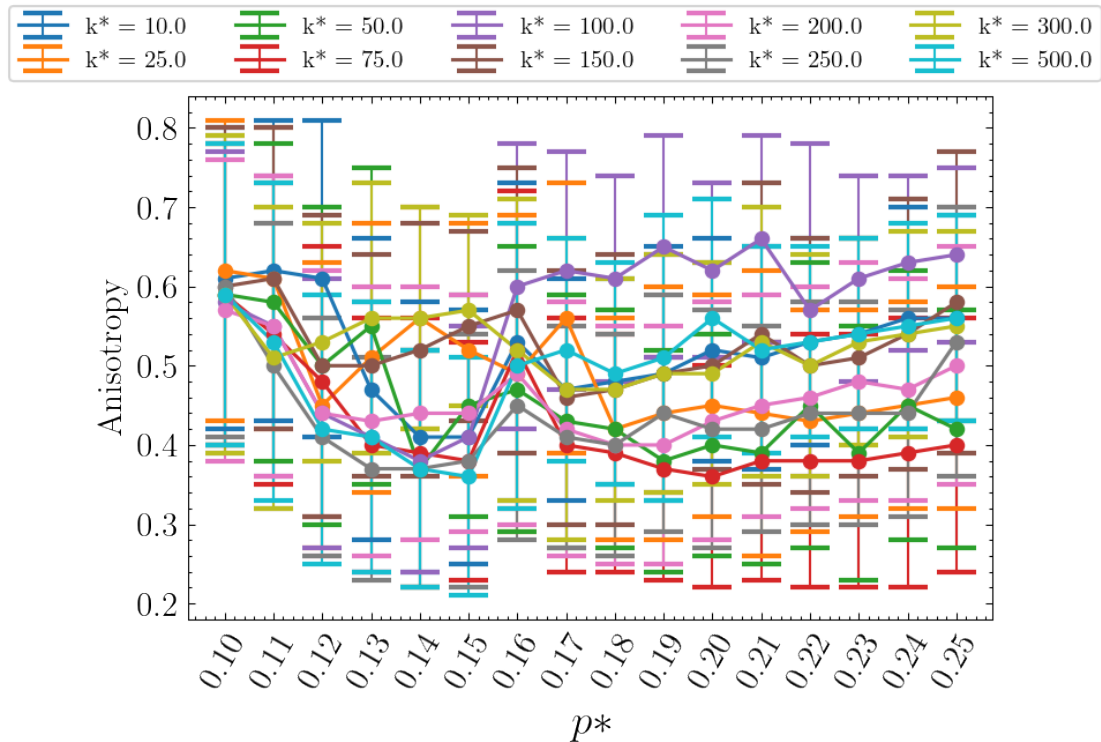


Figure 3.11 – This figure provides the average and standard deviation of the Anisotropy for every system of interest.

Fonte: The author.

The next attribute we will discuss is the **straightness**. In Subsection 2.1.1, we have established that the straightness measures the similarity between the trajectory and a straight line. If the straightness is equal to 1, the trajectory is a perfect straight line, whereas a null trajectory indicates that the trajectory is not straight whatsoever. Figure 3.12 provides the results of the straightness computed for every system.

As we expected from the results for the MSD ratio and the Fractal Dimension, Figure 3.12 shows that the mean straightness for each system is close to zero. This indicates that each trajectory may be represented as a Random Walker, which is precisely what we have been stating so far. If we had observed values for straightness close to 1 at some point, the diffusion of the system in question would be classified as Direct Motion with small deviations from a straight line. This would only happen for small p^* , since large pressures would compress the system to a point where the polymer rings would not be able to move much, let alone move in a straight line.

Interestingly enough, the largest values of straightness are observed for low values of pressure. This indicates that the polymer rings are able to move in a straight line for a very short period of time between collisions, which should be expected since the pressure is low.

The next attribute we will discuss is the **efficiency**. In Subsection 2.1.1, we have defined the efficiency of a trajectory to be a measurement that relates the length of the trajectory to its net displacement. Figure 3.13 provides the average efficiency for every system.

The null average efficiency indicates that either the net displacement for each polymer ring is very small or the trajectory length is very large. The fact that for every k^* the efficiency is the same

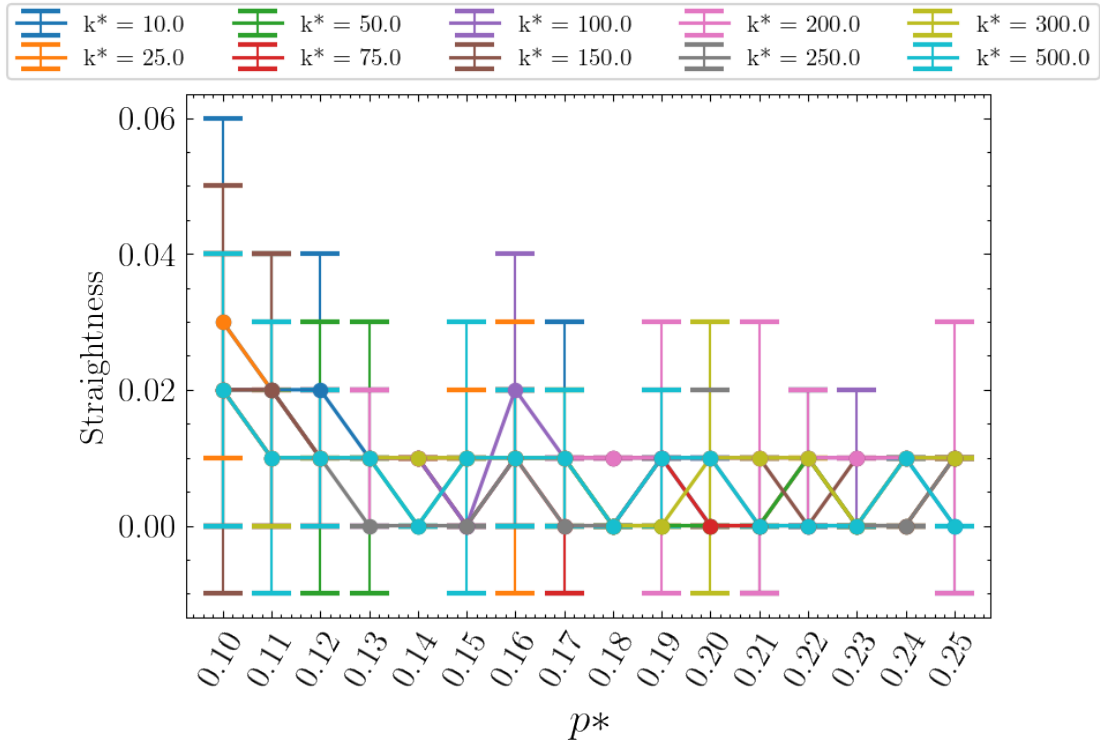


Figure 3.12 – This figure provides the average and standard deviation of the Straightness for every system of interest.

Fonte: The author.

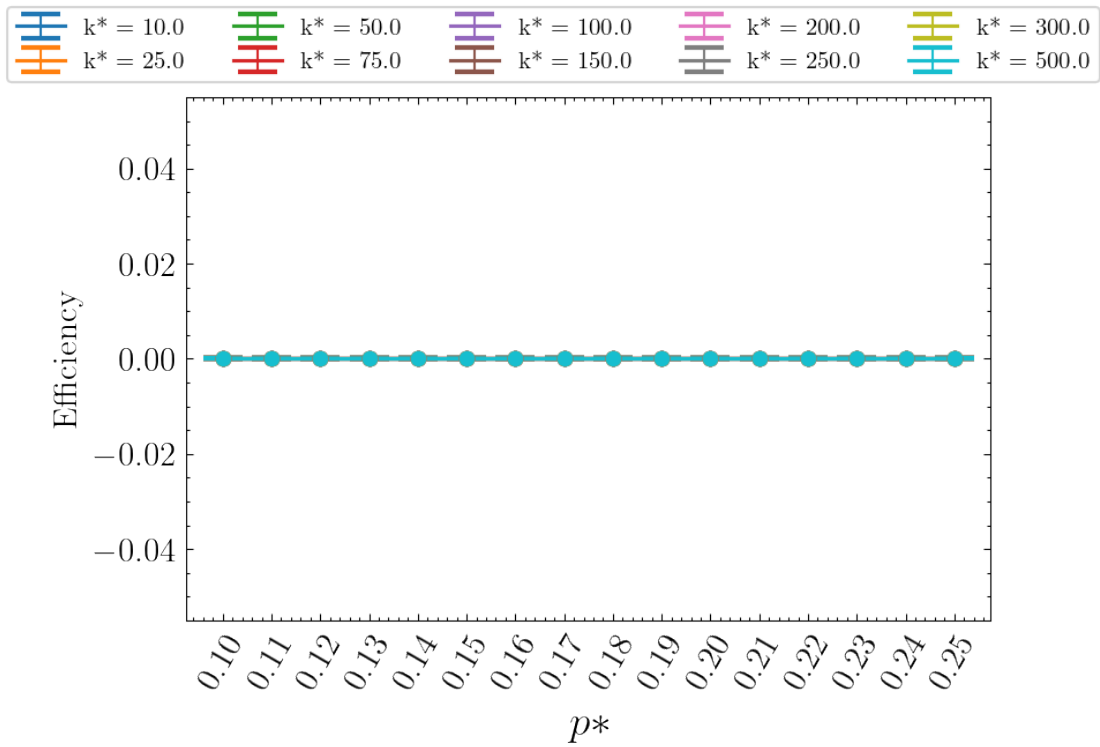


Figure 3.13 – This figure provides the average and standard deviation of the Efficiency for every system of interest.

Fonte: The author.

points out to what we had previously considered: the effects of the spring's constant on the system may be much smaller than the one of the pressure p^* .

The next attribute we will discuss is the **gaussianity**. As discussed in Subsection 2.1.1, the gaussianity measures the similarity between the positions and the Gaussian distributions. The more similar both distributions are, the closer to zero the gaussianity will be. Figure 3.14 shows the results of the average gaussianity computed for every system.

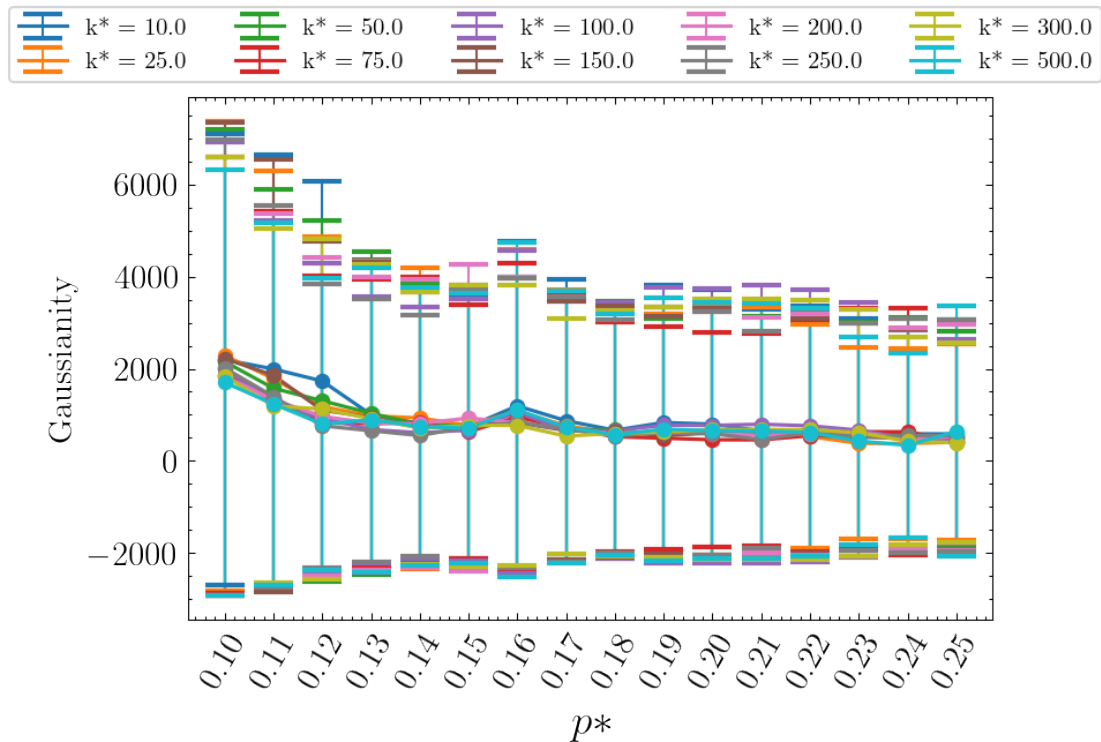


Figure 3.14 – This figure provides the average and standard deviation of the Gaussianity for every system of interest.

Fonte: The author.

When analysing Figure 3.14 we notice that both the average gaussianity and its standard deviation are quite large, specially the latter. Before we consider discussing the meaning of these results, let us analyse the value of the gaussianity for each polymer ring for each system so that we can make sense of the large averages and standard deviations. To do that, we will count the number of polymer rings, per system, whose gaussianity was measured to be larger than one. Figures 3.15 provides the results of the counting process, where for every value of pressure p^* and k^* , we counted the number of polymer rings that present a gaussianity larger than one.

As we can see in Figure 3.15, there is a certain amount of polymer rings per system whose gaussianity is larger than one. We notice that such amount decreases as the pressure p^* increases - except for $p^* = 0.16$, where there is a momentarily increase, which is observed when we computed the average gaussianity (and also the average of other attributes, where, for $p^* = 0.16$, there is a change in behavior for that average). We now take a step further and display histograms for the values of the gaussianity of those polymer rings whose gaussianity is larger than one. In Figure 3.16, we

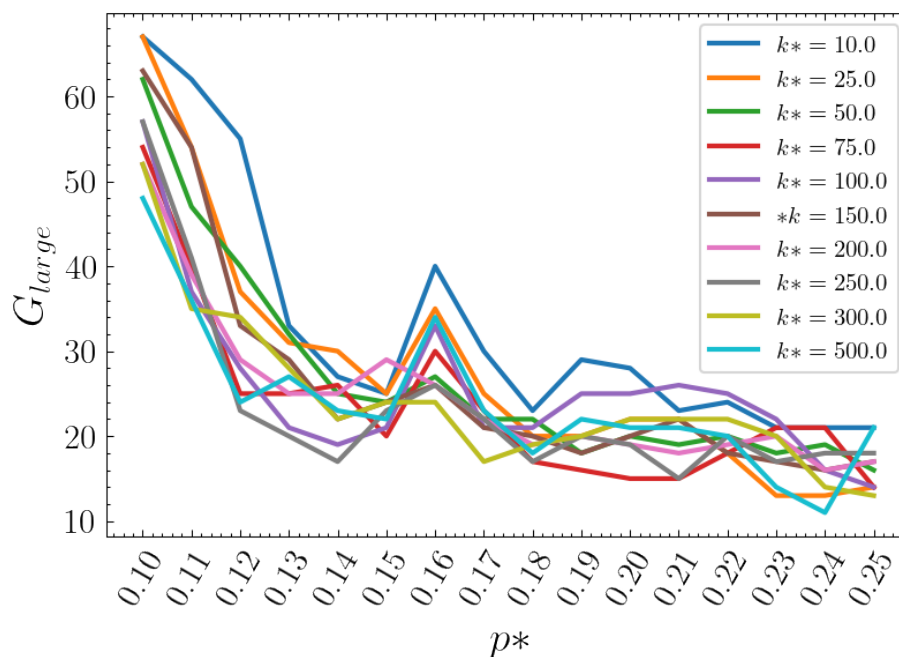


Figure 3.15 – This figure provides the number of polymer rings whose gaussianity was measured to be larger than one for all values of k^* and p^* .

Fonte: The author.

show those histograms for $k^* = 10.0$ for every value of p^* . We have attached graphs of the same kind for $k^* = 25.0, 50.0, 75.0, 100.0, 150.0, 200.0, 250.0, 300.0$ and 500.0 in Attachment B.

When analysing Figure 3.16, we notice that the values of gaussianity, when larger than one, are extremely large. According to our discussion on the meaning of the gaussianity, large values are expected when the positions' distribution does not resemble a Gaussian Distribution - for reference, see Ernst, Köhler e Weiss (2014). Let us now discuss the meaning of the results for the gaussianity presented in Figure 3.14. As we see, the gaussianity decreases significantly as the pressure increases for all values of k^* . This means that the positions' distributions get more similar to a Gaussian Distribution as the pressure rises, i.e., the polymer rings, on average, present motions that resembles more and more a Random Walker. Given that the large values of gaussianity have a significant effect on the average value computed, we have calculated the average gaussianity once more, where we have ignored those large values so that we can obtain the expected gaussianity for the majority of the polymer rings for each system. Figure 3.17 presents the average gaussianity for this case.

We can clearly see in Figure 3.17 that the average gaussianity - when we ignore the large values - decreases from values close to one to zero. This transition, once again, indicates that the polymer rings, on average, behave as Random Walkers. We are not able to point out if such transition implies that we have a dynamical transition from Normal to Confined Diffusion, once both motion types are quite similar, as discussed in Section 2.2. However, we do have a guarantee that the majority of the polymer rings behave as expected from the previous results discussed in this section.

The next feature we will discuss is the **kurtosis**. As stated in Subsection 2.1.1, the kurtosis measures the tailness of the positions distribution once we project it along the main eigenvector of the gyration tensor. Figure 3.18 provides the results we have obtained for the position kurtosis.

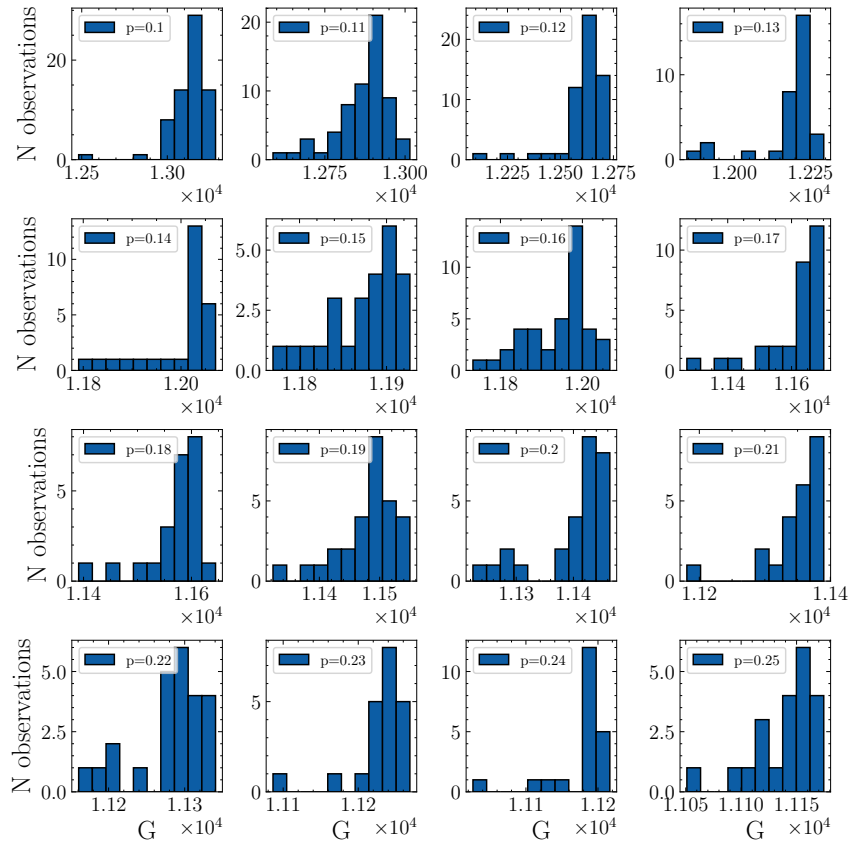


Figure 3.16 – This figure provides the histograms for the observed gaussianity larger than one for every value of p^* for $k^* = 10.0$.

Fonte: The author.

A kurtosis equal to 3 indicates the position distribution is a Gaussian Distribution, whereas a kurtosis lower and larger than 3 implies in a distribution with a sharp peak and short tails and a short peak with long tails, respectively. We can easily notice that, on average, the kurtosis somewhat oscillates around 8 for small p and, as the pressure increases, the oscillations present a smaller amplitude and begin oscillating around 6. In addition, we observe once again large values for the standard deviation. Let us investigate this a bit further.

As we see in Figure 2.8, a kurtosis of 4.5 would not be a surprise in the context we are discussing in this dissertation. With that in mind, we will approach the problem of the standard deviation in the same way as we did for the gaussianity. We will count the number of polymer rings for each system that present a kurtosis larger than 5 and consider them to be large values. Figure 3.19 provides the results for each value of the springs' constants k .

As we notice in Figures 3.19, the number of polymer rings that present a kurtosis larger than 5 are somewhat similar to the analogous case for the gaussianity. In addition, we once again observe a decrease in the number of polymer rings that obey the condition we have used as the pressure increases. The sharp increase for $p^* = 0.16$ is also noticed once more, as we did for the gaussianity. We now present the histograms for the observed values of the kurtosis larger than 5 for $k^* = 10.0$ for all values of p in Figure 3.20.

In Figure 3.20 we see that there are a few observations of a kurtosis that is actually large,

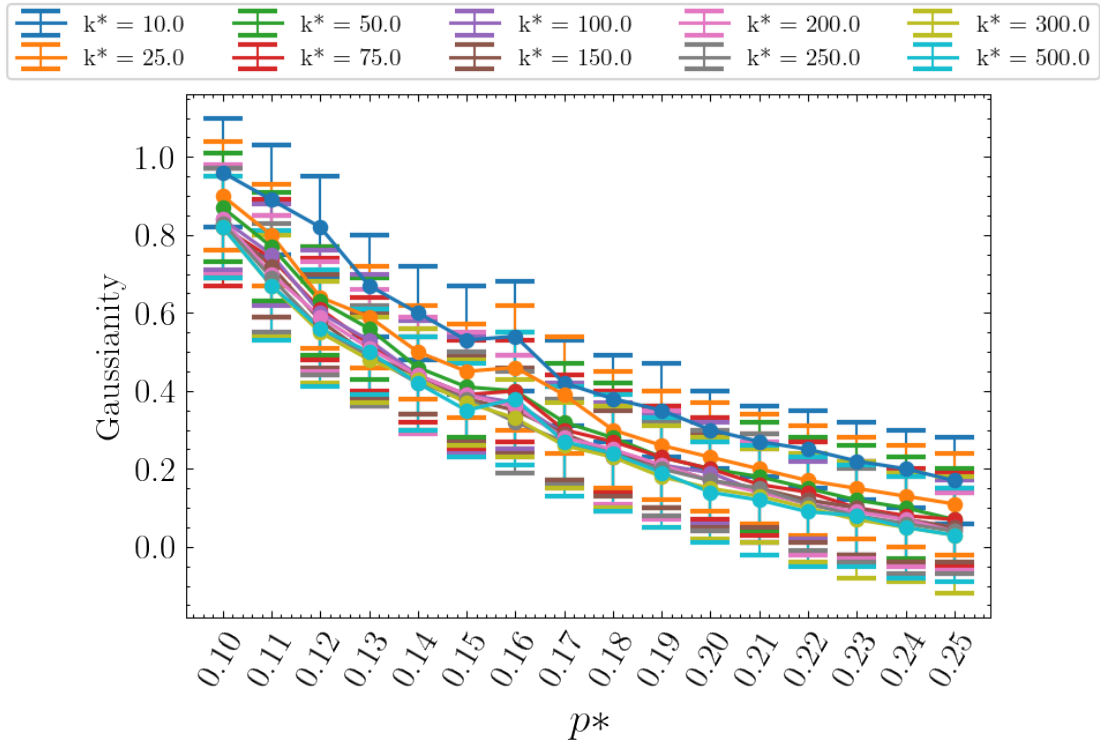


Figure 3.17 – This figure provides the average and standard deviation of the Gaussianity for every system of interest, once we ignore the large values we observe.

Fonte: The author.

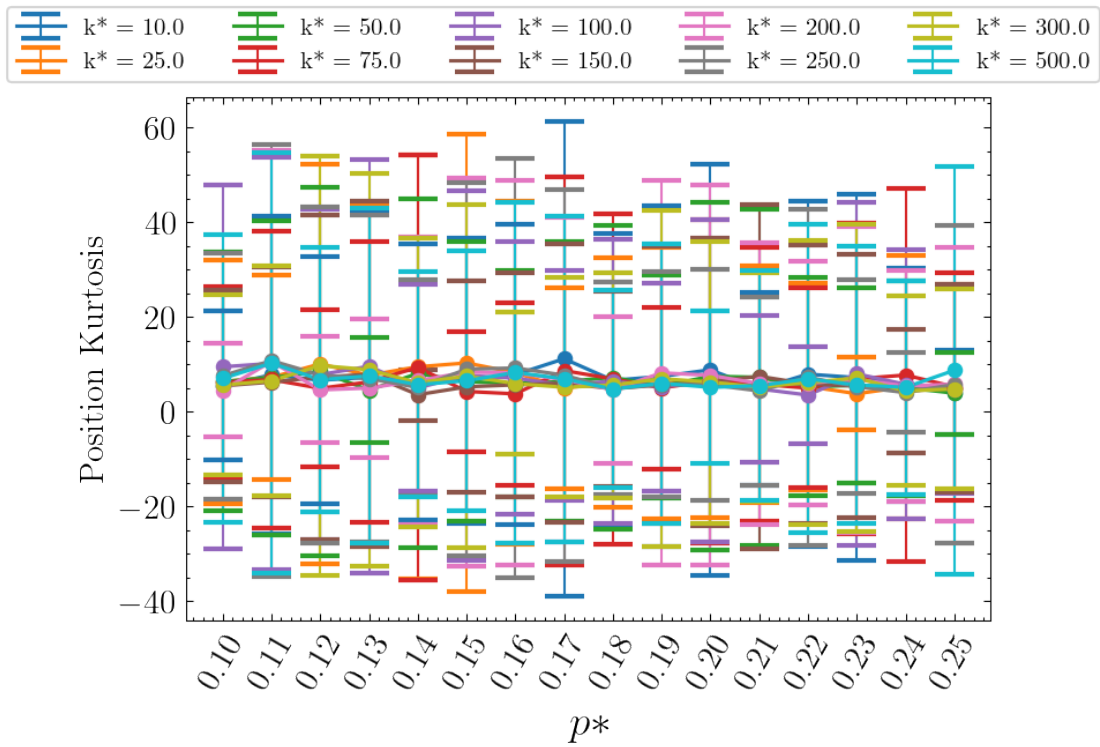


Figure 3.18 – This figure provides the average and standard deviation of the Kurtosis for every system of interest.

Fonte: The author.

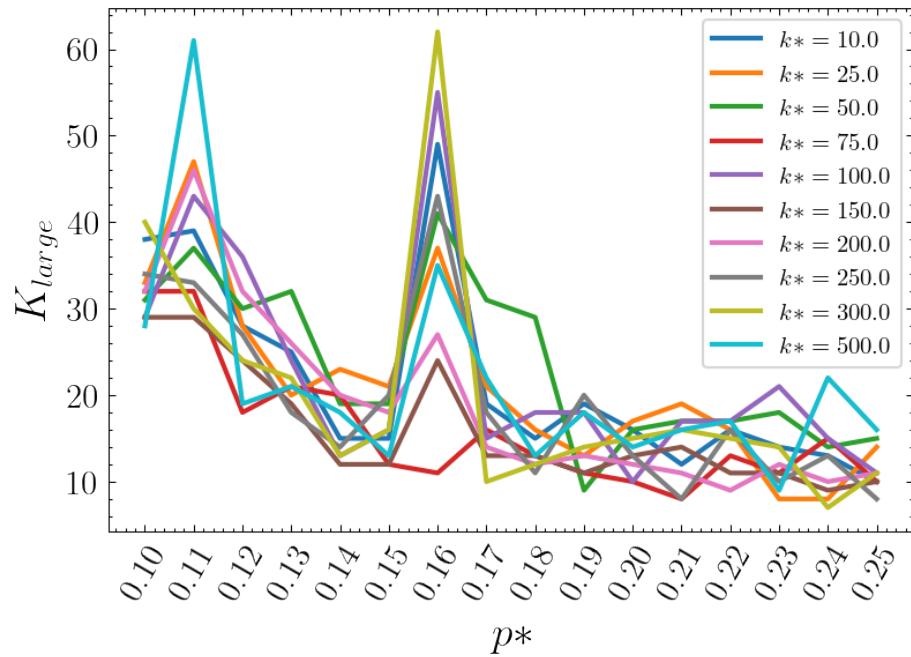


Figure 3.19 – This figure provides the number of polymer rings for all values of p^* and k^* whose kurtosis was measured to be larger than 5.

Fonte: The author.

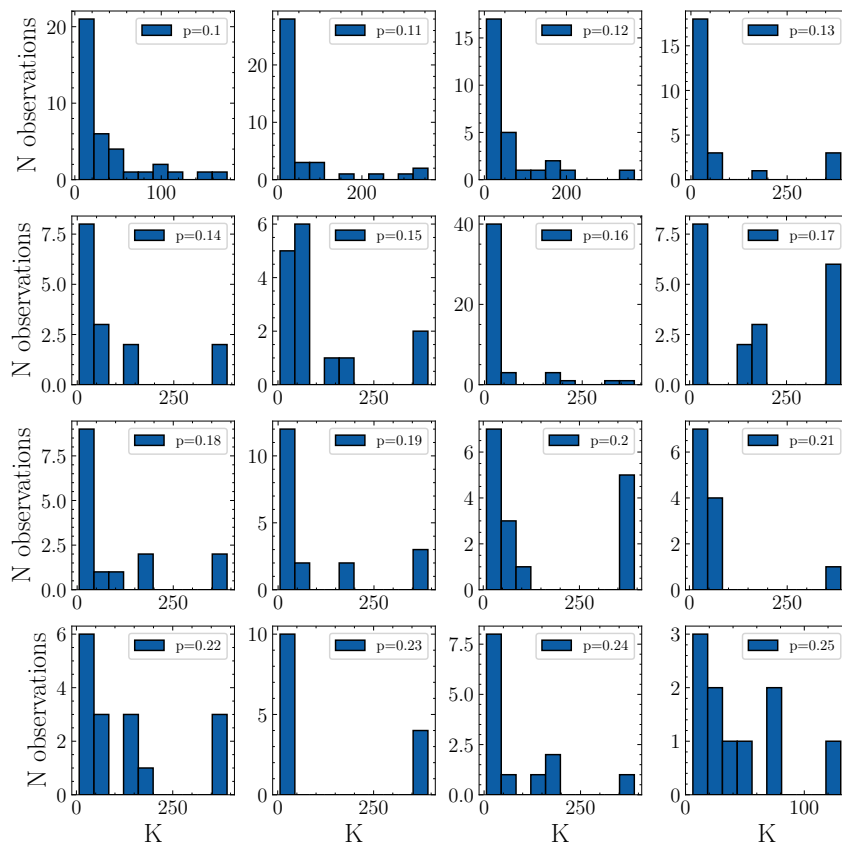


Figure 3.20 – This figure provides the histograms for the observed values of kurtosis larger than 5 for every value of p^* for $k^* = 10.0$.

Fonte: The author.

such as a kurtosis equal or close to 250. This is the reason we observe a large standard deviation for the kurtosis, where in this case, the standard deviation is not as near as large as in the case for the gaussianity. See Attachment B for the histograms for the remaining values of k^* for all pressures p^* . We now compute the average kurtosis for the case where we ignore the large values - the cases where the kurtosis is larger than 5. Figure 3.21 provides the results so that we can discuss them.

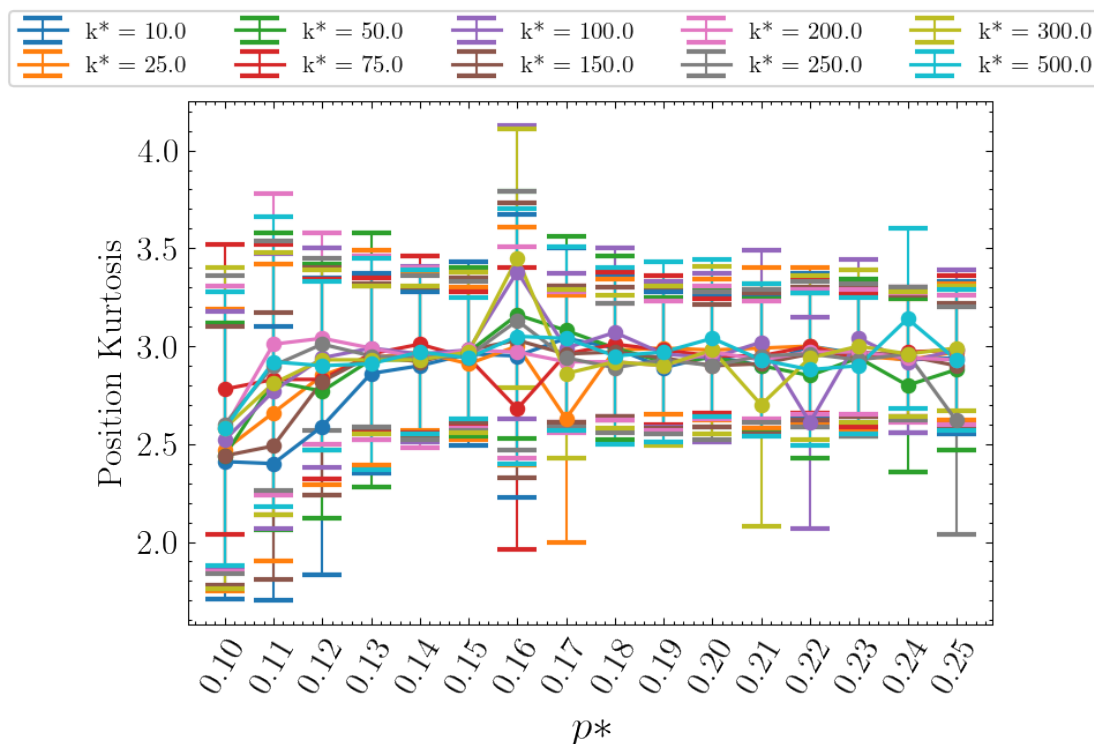


Figure 3.21 – This figure provides the average and standard deviation of the kurtosis for every system of interest, once we ignore the large values.

Fonte: The author.

When we analyse Figure 3.21, we see that the standard deviation has decreases significantly and that there is a transition from values that are a bit smaller than 3 to 3 itself. This transitions indicates that the polymer rings possess positions' distributions that resembles a Gaussian Distributions as p increases. That means the polymer rings, on average begin to behave as Random Walkers, which is once again what we have been observing. This implies that the polymer rings follow the motion type of Normal or Confined Diffusion. In order to specify which one, we will have to rely on the results of the previous attributes as well.

The next attribute we will discuss is the **velocity kurtosis**. As we have established in Subsection 2.1.2, the velocity kurtosis measures the shape of the velocity distribution in the same way the **kurtosis** does. However, there is a slight difference regarding the values of both kurtosis. In the case of the velocity kurtosis, a null kurtosis implies in a Gaussian distribution, whereas a positive and negative kurtosis indicate the same shape of distributions as the kurtosis larger and smaller than 3. Figure 3.22 shows the results for the velocity kurtosis.

When analysing Figure 3.22, we notice that the velocity kurtosis decreases significantly as the pressure increases. We also notice, once again, large standard deviations. We will approach this

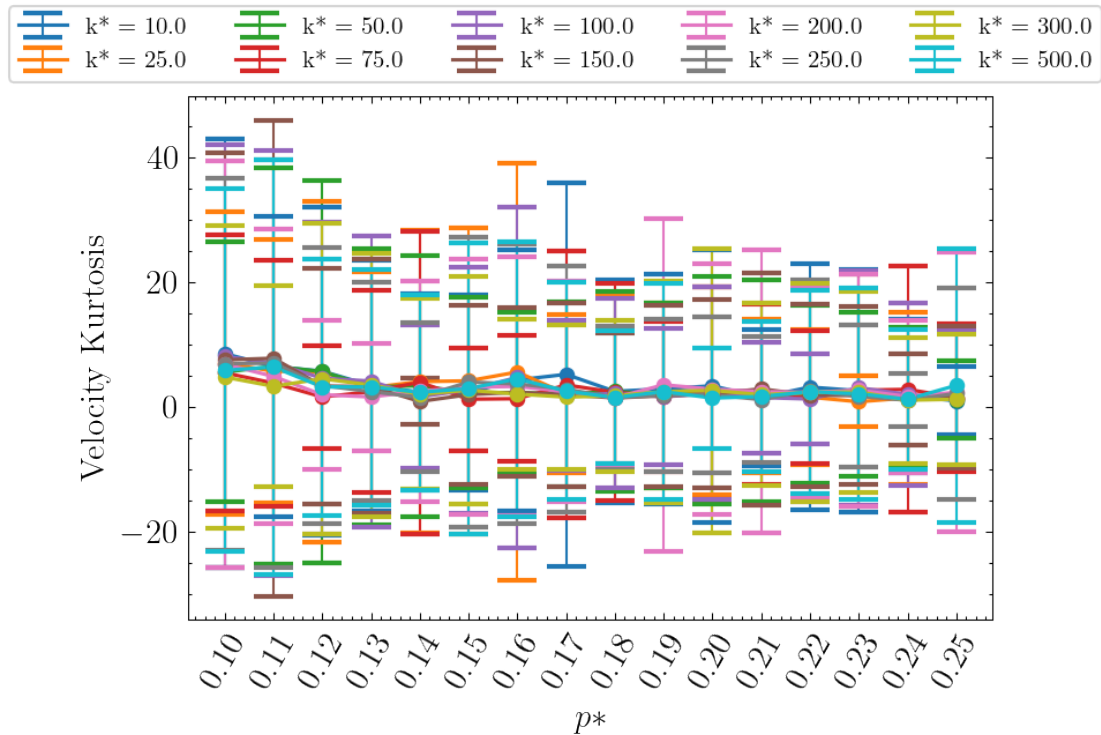


Figure 3.22 – This figure provides the average and standard deviation of the Velocity Kurtosis for every system of interest.

Fonte: The author.

problem in the same way as before. First, we count the number of polymer rings, per system, that present a velocity kurtosis larger than 2 - given that a velocity kurtosis may present positive values, we will choose the value of 2 to be considered large values. Figure 3.23 presents the number of polymer rings per system that obey this condition.

In Figure 3.23 we observe the same trend as before: the number of polymer rings that present a velocity kurtosis larger than 2 decreases as the pressure increases and. In addition, the sharp increase in the same number appears for $p = 0.16$. We now present the histograms for the outliers of the velocity kurtosis for all pressures p^* and $k^* = 10.0$ in Figure 3.24. The histograms for the outliers of the velocity kurtosis for the remaining values of k can be seen in Attachment B.

Lastly, we provide the results for the average velocity kurtosis, once the outliers are ignored. Figure 3.25 shows these results. When we analyse Figure 3.25, we see that the velocity kurtosis decreases as the pressure p^* increases. This decrease indicates that the polymer rings, on average, present a velocity distribution that becomes more similar to a Gaussian Distribution as the pressure rises.

The next attribute we will discuss is the **velocity skewness**. In Subsection 2.1.2, we have defined the velocity skewness to be the measurement of symmetry of the distribution when its center point is considered. A Gaussian distribution of velocities yields a null skewness, since it is symmetric. A velocity distribution with positive skewness indicates that this distribution presents a short left tail and a long right tail, whereas a negative skewness implies that the opposite is true. Figure 3.26 provides the results for the average velocity skewness for every system.

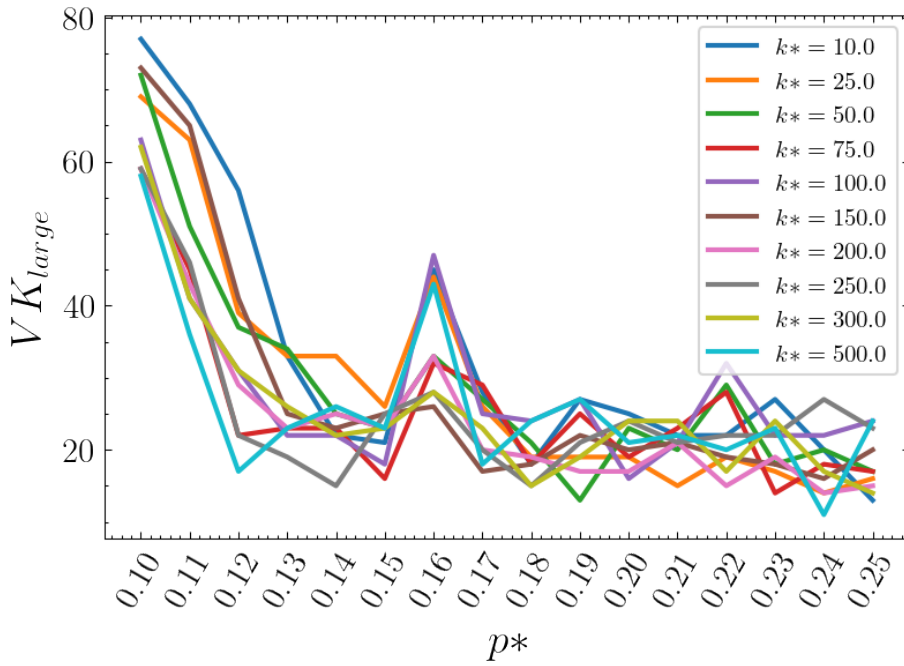


Figure 3.23 – This figure provides, for all values of k^* and p^* , the number of polymer rings whose velocity kurtosis was measured to be larger than 2.

Fonte: The author.

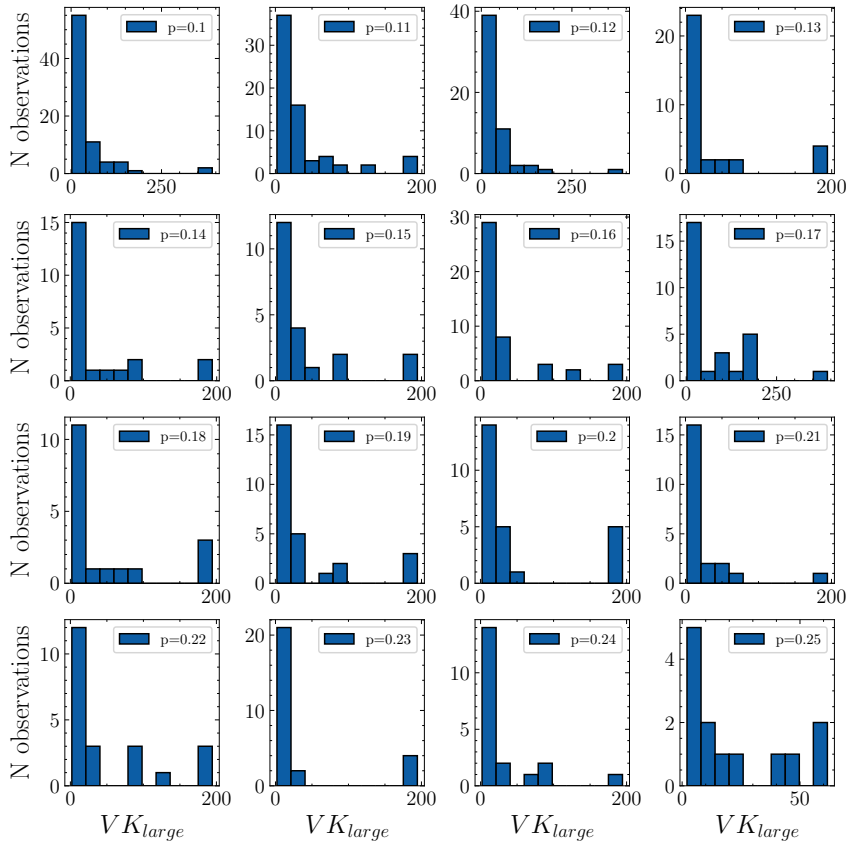


Figure 3.24 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p^* for $k^* = 10.0$.

Fonte: The author.

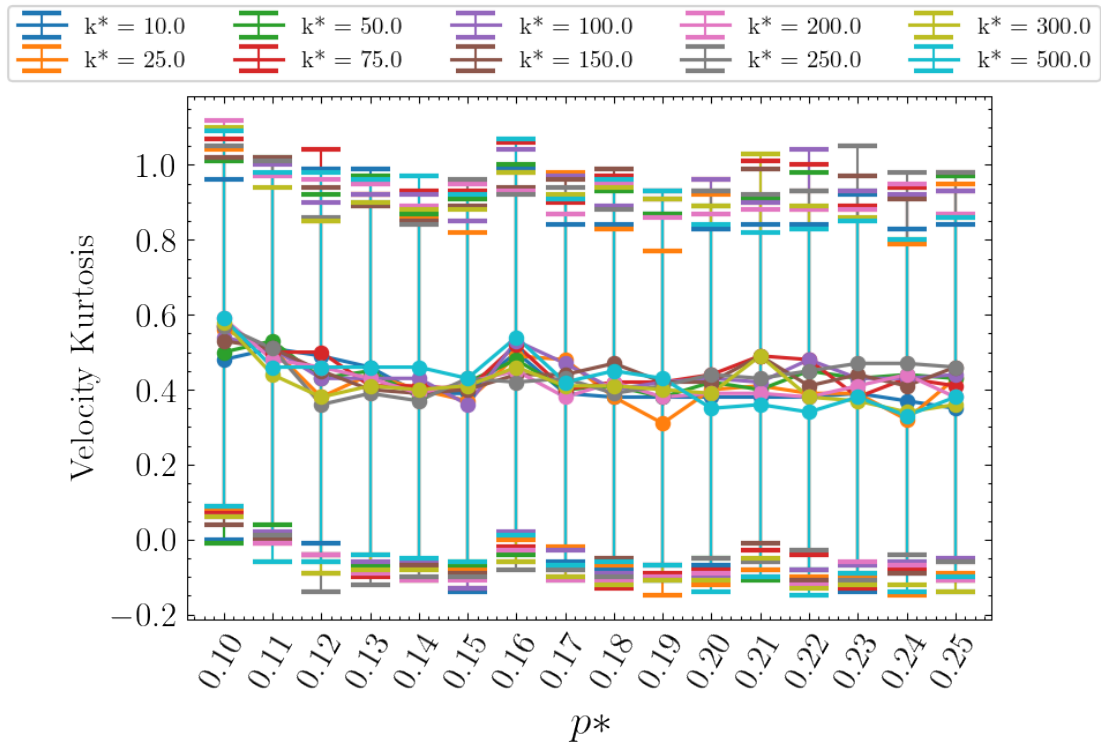


Figure 3.25 – This figure provides the average and standard deviation of the velocity kurtosis for every system of interest, once we ignore the large values.

Fonte: The author.

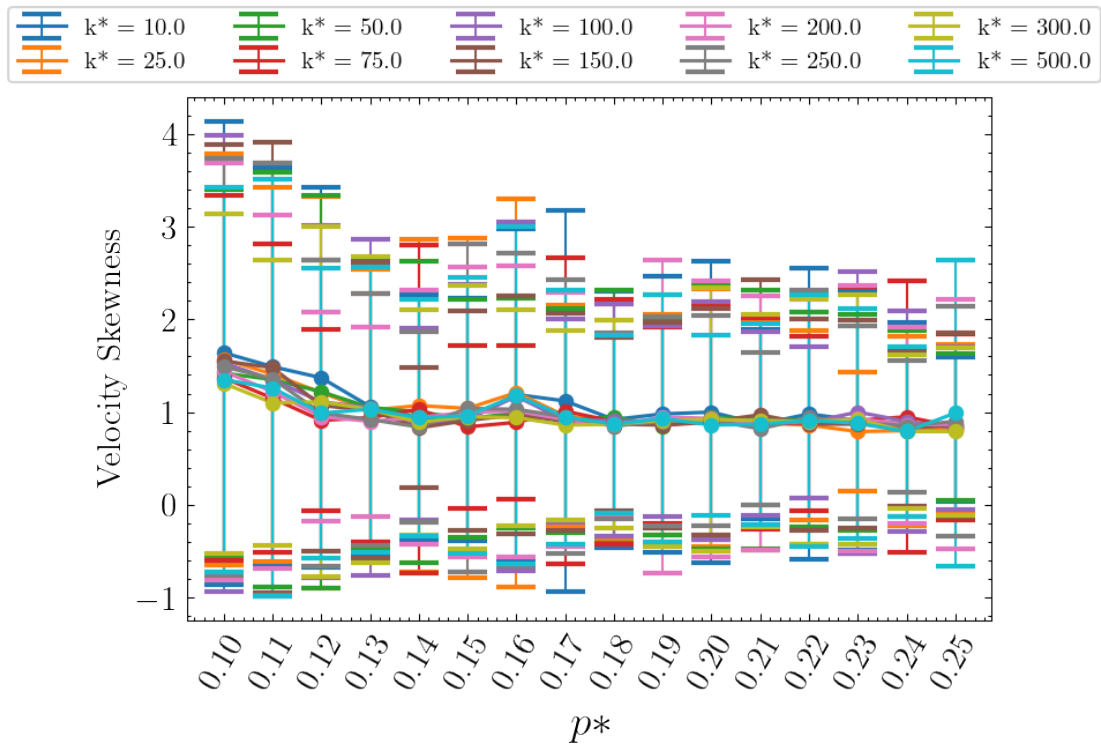


Figure 3.26 – This figure provides the average and standard deviation of the Velocity Skewness for every system of interest.

Fonte: The author.

As we can see in Figure 3.26, the average skewness for each system decreases as the pressure increases. This means that the velocity distributions become more similar to a Gaussian distribution. For low p^* , the average positive skewness implies that the distributions present a long right tail, which, in turn, indicates that we are more likely to encounter a particle with small velocities. At this point, we observe that the increase of the quantity being measured for $p^* = 0.16$ is once again present.

The last attribute we will discuss is the **diffusion coefficient**. As discussed in Subsection 2.1.2, we use the **Green-kubo Relation** to compute the diffusion coefficient for each particle. Such coefficient indicates how fast each particle can swipe out a unit of area. Figure 3.27 shows the average diffusion coefficient we have computed for each system.

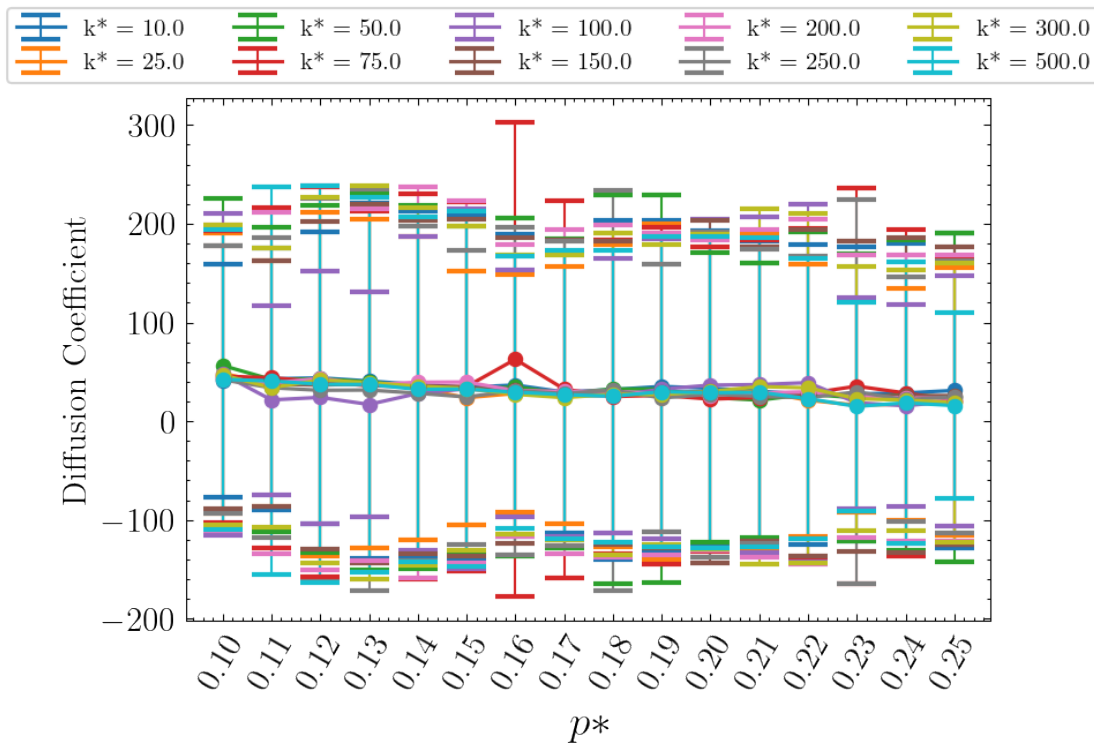


Figure 3.27 – This figure provides the average and standard deviation of the Diffusion Coefficient for every system of interest.

Fonte: The author.

As we can see in Figure 3.27, the average diffusion coefficient decreases as the pressure p^* increases. In addition, we observe a large standard deviation once more. To analyse the values for the diffusion coefficient, we can no longer look for large values as before, since there are no base values we can use for our analysis. To work around this fact, we have generated histograms for all values of the Diffusion Coefficient D for every system and came to the conclusion that every value of $D > 5.0$ may be considered to be large, given that approximately 350 values for D were calculated to be less than 5. Considering this threshold, we may proceed as before. Figure 3.28 provides the number of polymer rings per system that present a diffusion coefficient larger than 5.

When we analyse Figures 3.28, we observe the same pattern we have been noticing. The number of polymer rings that present a diffusion coefficient larger than 5 decreases as the pressure

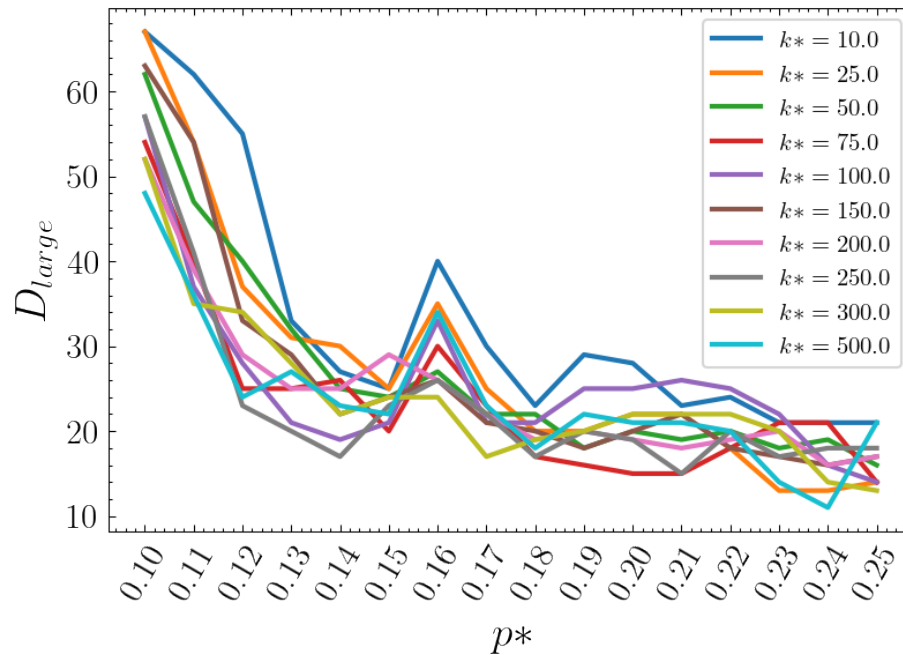


Figure 3.28 – This figure provides , for value of p^* and k^* , the number of polymer rings whose diffusion coefficient was measured to be larger than 5.

Fonte: The author.

increases. In addition, we see that this number increases momentarily for $p = 0.16$ as well. We now present the histograms for the values of the diffusion coefficient considered to be outliers for all values of p for $k = 0.10$ in Figure 3.29. The histograms for the other values of k can be seen in Attachment B.

Lastly, let us present the average diffusion coefficient once we ignore the large values. Figure 3.30 presents the results. It is clear that the diffusion coefficient decreases as the pressure increases, which comes to show that, for high pressures, the polymer rings can no longer move as much as they did when the pressure was low. We may use this analysis to point out that the polymer rings are probably confined in a small space, which points to the direction of Confine Diffusion.

We have now finished discussing the results of each attribute. As discussed, there is an apparent dynamical transition from Normal Diffusion to Confined or Anomalous Diffusion. We must now classify the diffusion for each polymer ring of each system. The classification procedure involving Machine Learning and the results we have obtained will be discussed in the next section.

3.3 Machine Learning and diffusion classification

In this section we will discuss the idea of Machine Learning and present the Diffusion Classification results. In Subsection 3.3.1 we provide a brief introduction to the three main classes of algorithms in Machine Learning. Lastly, in Subsection 3.3.2 we discuss the diffusion classification results we have obtained.

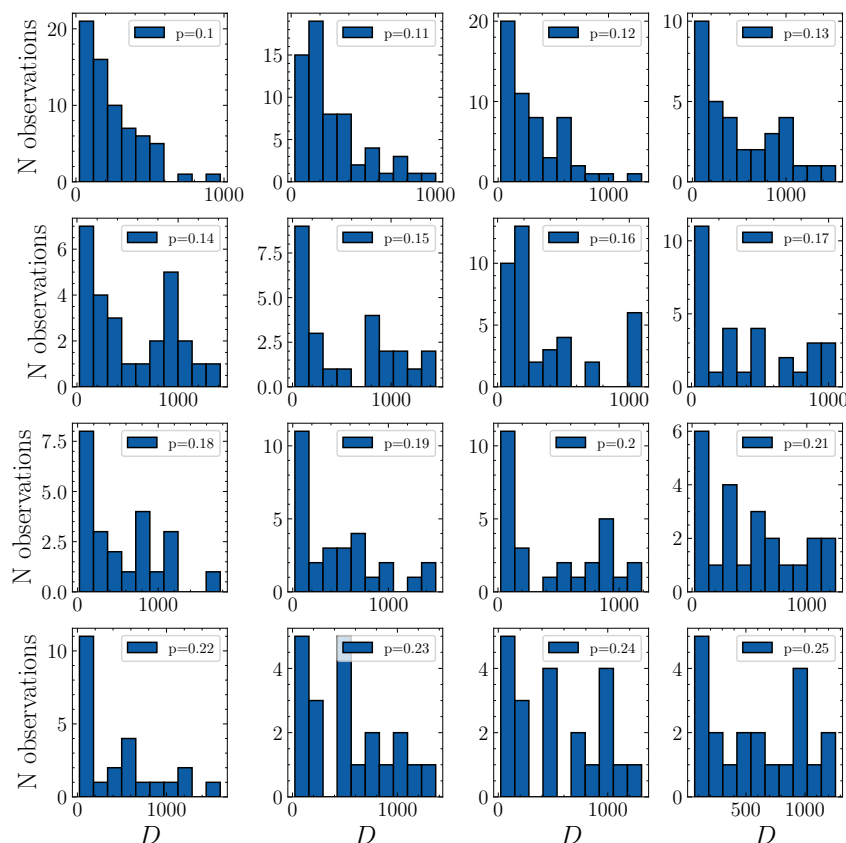


Figure 3.29 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 10.0$.

Fonte: The author.

3.3.1 A general description of Machine Learning

Machine Learning algorithms are a great computational tool to deal with massive amounts of data. Due to the technological advance in the last decades, regular computers are now able to perform the statistical computations involved in such algorithms. This has made the Machine Learning usage a common topic of discussion in our everyday lives and specially in Science. A quick search online will reveal that many scientific papers have now adopted the topic of this subsection in their methodology (MJOLSNESS; DECOSTE, 2001). Machine Learning can be used in any context, such as Material Science (WEI *et al.*, 2019) and molecular biophysics (MATEUS; SAVITSKI; PIAZZA, 2021), which comes to show how versatile and useful these algorithms are.

In order to classify the diffusion of each polymer ring, we must first explain what is meant by “classification” in the context of Machine Learning. To do that, we will describe in general terms the three main classes of Machine Learning algorithms - Supervised, Unsupervised and Reinforcement Learning - and move on to a simple explanation of Classification and Regression algorithms.

Simply put, Supervised Learning is a class of algorithms that seek the correct answer to a question by means of performing a fit using the data to known answers of that question, e.g., a company that needs to predict the probability of a certain client buying their product given that the client is fifty years old. The data in question would be the client’s age and the known answers

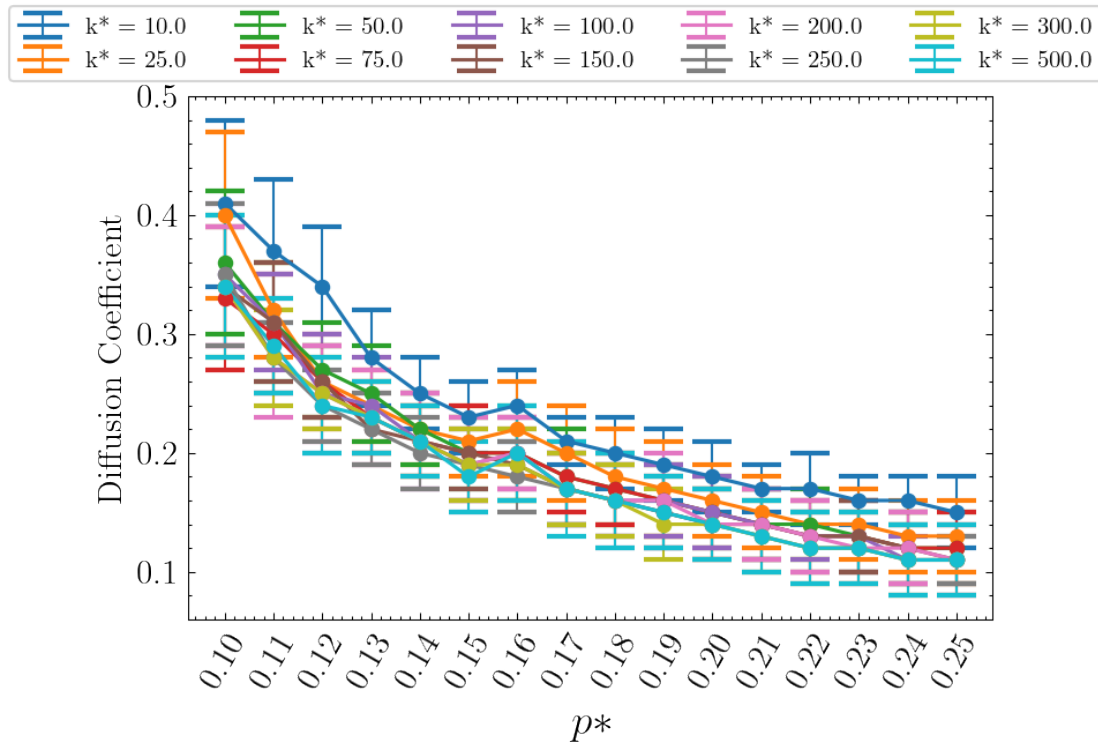


Figure 3.30 – This figure provides the average and standard deviation of the diffusion coefficient for every system of interest, once we ignore the large values.

Fonte: The author.

would be a large list of clients that bought the company's product and their associated age. The fit previously mentioned may be the attempt to fit a certain function, say $f(t)$, to the data in order to predict the outcome of a certain situation. This mathematical function may take different forms, such as the classical Linear Regression performed in Statistics (SU; YAN; TSAI, 2012) or the Logistic Regression (NICK; CAMPBELL, 2007). On the other hand, the fit may take a different form, such as the Ensemble Method (DIETTERICH, 2000), which will be the method used to classify the diffusion of all polymer rings.

The Unsupervised Learning, on the other hand, is a class of algorithms that also seek the answer to a question, perhaps that of the needs of the company, but without making use of known answer- possibly because there are no known answers. In order to use the data as input to make predictions, these algorithms look for similarities between the data points. A famous way to acquire such similarities is the Nearest Neighbors approach (GHAHRAMANI, 2004), where the algorithm looks for similar information using a given number of data points in the vicinities of a specific data point. This approach is also known as Data Clustering.

Lastly, let us briefly discuss the main idea of the third class of machine Learning algorithms. Reinforcement Learning is a class of algorithms that, given the proper context, seek the best action that leads a so called agent to a state with the highest reward (SUTTON; BARTO, 2018). These techniques are widely used in state of the art technologies such as chat bots that guide us through a company's website or self-driving cars. The agent in question could be interpreted as a fictitious man and the state is the situation he's encountered in any given environment. The action he takes will

produce a consequence we call a reward. The goal of Reinforcement Learning is to maximize this reward by taking multiple actions over multiple processes and deciding which set of actions, taken in a specific order, optimizes the reward.

Now that we have provided a brief description of the three main Machine Learning algorithm classes, it is time for us to discuss the Supervised Learning in more details, for the classification algorithm we are going to use belongs to this specific class.

In Supervised Learning, there are two kinds of algorithms called Regression and Classification⁶. While Regression algorithms look for numerical answers to a problem such as that probability of a specific client buying a company's product we previously discussed, Classification algorithms seek categorical answers - in the client and company example, the categorical answer could be the gender of the client that is most likely to buy the product.

3.3.2 Diffusion classification

Among the many algorithms for classification, we have decided to use the Random Forest algorithm. For a thorough discussion of its mathematical formulation, see (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

Put in simple terms, the diffusion classification process relies on three steps. First, we compute the attributes such as the Gaussianity and Fractal Dimension for the trajectories we wish to classify the diffusion of. We then simulate many different trajectories for each diffusion class. Once the simulations are performed, we compute the same attributes we used in the first step.

Lastly, we use Machine Learning algorithms - in this case we will use Supervised Learning - in order to classify the diffusion of each polymer ring of each system. In this particular step, a fit⁷ is performed so that the Random Forest algorithm finds the pattern within the data we provide.

In the previous sections, we have discussed both the physical system of interest and each the results of the attribute we have computed using TrajPy. Now we must briefly discuss the data we have provided to the classification algorithm so that it can seek the pattern we previously mentioned.

The dataset⁸ consists of 9 attributes computed for 106584 two dimensional particles, where 26520 were simulated with Normal Diffusion, 34270 with Anomalous Diffusion, 19998 with Confined Diffusion and 25796 with Direct motion with diffusion. The attributes in this dataset are the **Anomalous Exponent, MSD Ratio, Fractal Dimension, Anisotropy, Kurtosis, Straightness, Gaussianity, Diffusion Coefficient** and the **Efficiency**. Figure 3.31 shows a scatter matrix, a simple way to compare such attributes at once in order to visualize the clustering of data points according to the diffusion class.

In order to provide this data to the Random Forest algorithm, we separated the dataset in two parts, where 70% of it was used to find the pattern and the remaining 30% was used to test

⁶ If we were to be precise, we should refer to both kinds of algorithms as "Regression". This separation in two different types is usually introduced in textbooks for simplicity.

⁷ This fit may be interpreted in the same way as a statistical fit, since Machine Learning algorithms are simply statistical approaches used in a computer.

⁸ This dataset is currently online and opened to whoever wishes to use it. To access it, see Moreira-Soares (2022).

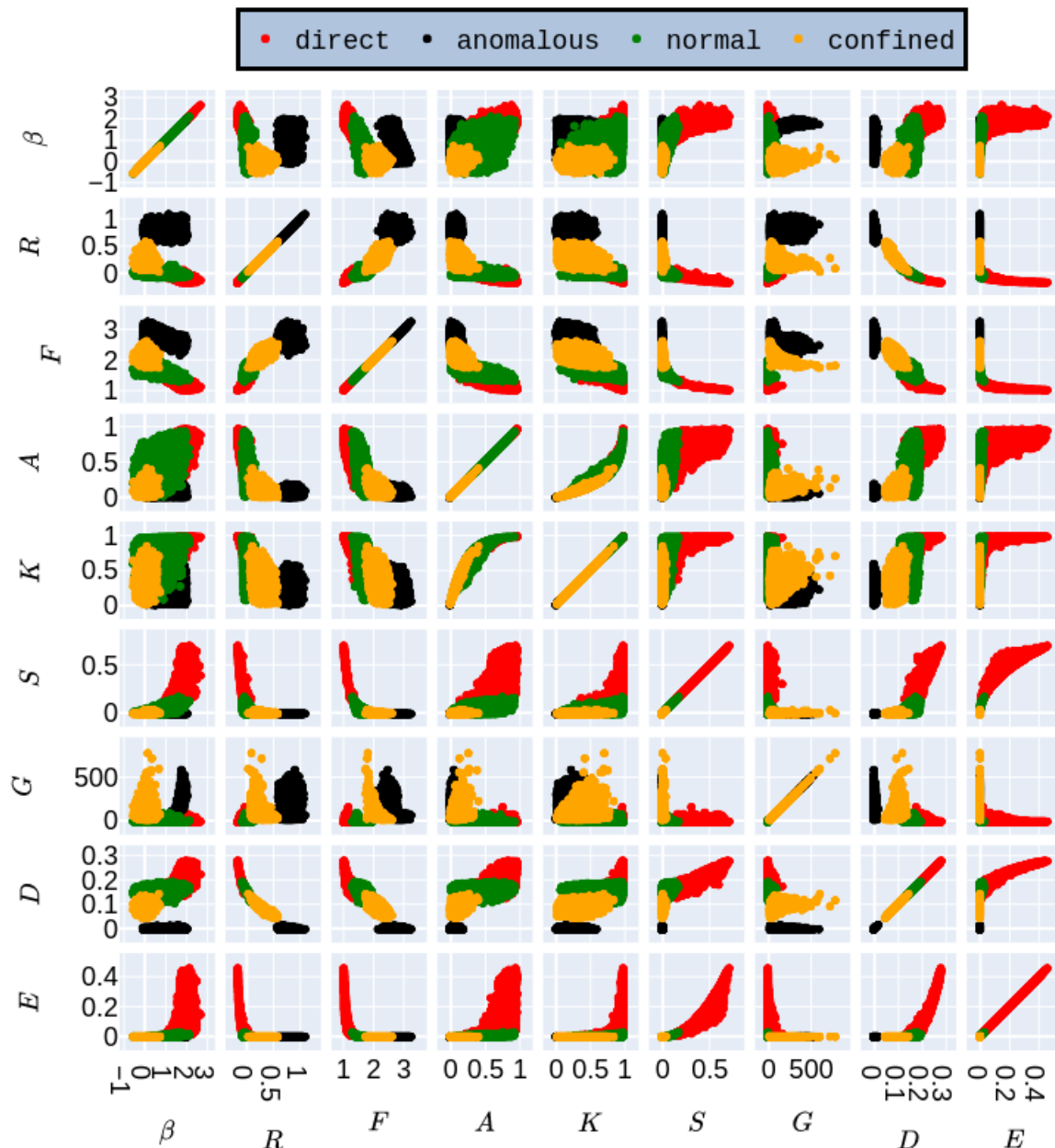


Figure 3.31 – This figure provides the scatter matrix for the 9 attributes we used to perform diffusion classification. Each color represents a diffusion class. The grouping of data points by color shows that these attributes are a good choice of parameters to classification. The labels in the x -axis - from left to right- represent the Anomalous Exponent, MSD Ratio, Fractal Dimension, Anisotropy, Kurtosis, Straightness, Gaussianity, Diffusion Coefficient and the Efficiency, respectively.

Fonte: The author.

the overall accuracy⁹ of the algorithm. This test consists of us providing to the algorithm - once it found the pattern - the remaining 30% of the dataset containing the attributes calculated. Once we do that, we are able to **predict** the diffusion class for the particles associated with those 30% remaining pieces of information. Given that the classification algorithm has never had access to the remaining 30% of the dataset, when we provide such data to the algorithm, we will be, essentially, classifying

⁹ By accuracy, we mean the number of times the algorithm correctly classified the diffusion of the particles in the dataset.

the diffusion of new particles. As a simple way to visualize the algorithm's accuracy for classifying the diffusion of particles it has never had access to, we use the Confusion Matrix shown in Figure 3.32.

Anomalous	10156.0	0.0	0.0	0.0
Confined	0.0	5990.0	1.0	0.0
Normal	0.0	0.0	8014.0	6.0
Direct	0.0	0.0	2.0	7807.0
	Anomalous	Confined	Normal	Direct

Figure 3.32 – This figure provides the Confusion Matrix for the diffusion classification performed in the 30% of the dataset. Simply put, we are comparing the known number of samples for each diffusion class in the x -axis to the predictions performed by the Random Forest algorithm in the y -axis. The main diagonal of this Confusion Matrix shows us the number of times the algorithm correctly classified the diffusion of the particles simulated. The other elements of the matrix show us the number of times the diffusion was classified as the wrong type.

Fonte: The author.

As we can see in Figure 3.32, the algorithm was able to correctly predict the diffusion class of the majority of particles, which shows that the Random Forest algorithm has actually found a pattern within the 70% of the dataset we provided and that this pattern does provide a good description of the data. Once we used the Random Forest algorithm to find the pattern within the data, we are able to predict the diffusion class of each polymer ring for every system. We may now use the Confusion Matrix to calculate two important metrics to measure the quality of the predictions: the True and False Positive Rates¹⁰.

In the context we are discussing, the True Positive Rate (TPR) measures the proportion of polymer rings whose diffusion was correctly classified, whereas the False Positive Rate (FPR) measures the analogous proportion regarding the number of times the diffusion was incorrectly classified. According to Jaskowiak, Costa e Campello (2022), these concepts are mathematically defined as

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \quad (3.13)$$

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}. \quad (3.14)$$

Let us explain the meaning of the four variables True and False Positives and True and False Negatives presented in Equation (3.13) and (3.14) by imagining a particle we simulated using TrajPy

¹⁰ These two metrics are sometimes called "Sensitivity" and "Specificity".

that presents Normal Diffusion. Given that this particle presents this specific motion type, we will consider to be positive the classification the algorithm made when it classified the motion type of this particle as Normal Diffusion and negative for the three other types.

A True Positive is observed when the algorithm classifies the motion type of that particle as Normal Diffusion and this classification is correct. A False Positive happens when that same classification is incorrect. On the other hand, a True Negative happens when the algorithm classifies the motion type to be other than Normal Diffusion and this classification is true. A False Negative is observed when the same classification is false, i.e., the motion type for that particle was, in fact, Normal Diffusion. All four variables for each class - in this case, the motion types - can be seen in the Confusion Matrix generated by a classification model.

These four concepts, True and False Positives and True and False Negatives, will change according to a certain threshold for the classification model. For every value of this threshold, that lies within zero and one, we will calculate the True and False Positives and True and False Negatives and, lastly, compute the TPR and FPR.

In the classification procedure, we calculate the probability of each class being the one the polymer ring presents. The threshold we just mentioned is used to determine how high this probability must be in order to assign a given diffusion class to a polymer ring. That is the reason this threshold lies within zero and one. Since we are working with four motion types, the one with the highest probability above the threshold will be the one we assign to each polymer ring. For each value of the threshold, we have a pair of results for the TPR and FPR. The curve that connects each pair of results is called the Receiver Operating Characteristic (ROC) curve. The reason we change the value of the threshold is to evaluate how well the classification model performs in different scenarios.

According to Spackman (1989), the TPR and FPR should be as close to one and zero as possible, respectively, for our classification model to provide good results. Figure 3.33 provides the ROC curve for each motion type for the Random Forest algorithm we have used to classify the diffusion of each polymer ring, where the Area Under the Curve (AUC) is used as a simple metric to relate both the TPR and FPR for each motion type - the closer to one the area under the curve, the better the classification model classified the diffusion class of those 30% of the dataset.

Figure 3.33 shows that our classification model indeed returned good values for the TPR and FPR, since the rate of incorrectly classifications is much lower than the rate for correct ones. If we had used more than one model to classify the diffusion of the polymer rings, we would have one ROC curve for each motion type for every classification model. Figure 3.34 presents such case for the use of multiple Machine Learning classification models to predict cardiac surgical operative mortality.

As we can see in Figures 3.32 and 3.33, our classification model performed well on the tests for prediction. Now, before we actually provide the classification procedure results, let us recall the results we expect.

We have established in the previous section that there is an apparent dynamical transition from Normal Diffusion to Confined or Anomalous Diffusion, where the Confined Diffusion is more likely to be observed, given the results we have gathered. As the pressure increases, the polymer rings become trapped, since the pressure makes them get closer and closer to each other in a way that

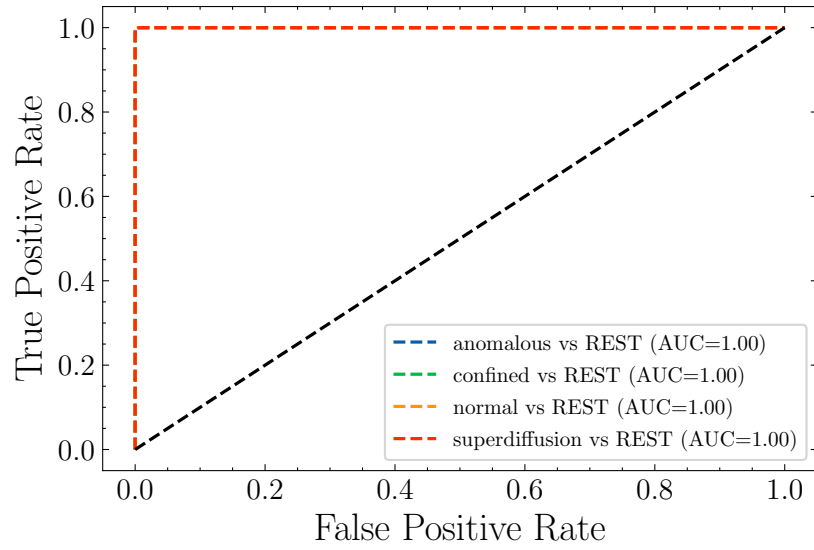


Figure 3.33 – This figure provides the ROC curve for the Random Forest classification algorithm we have used to classify the diffusion of each polymer ring for all systems, where we compare the TRP and FPR for a specific motion type with the remaining three. The black diagonal line shows where the TPR and FPR are equal.

Fonte: The author.

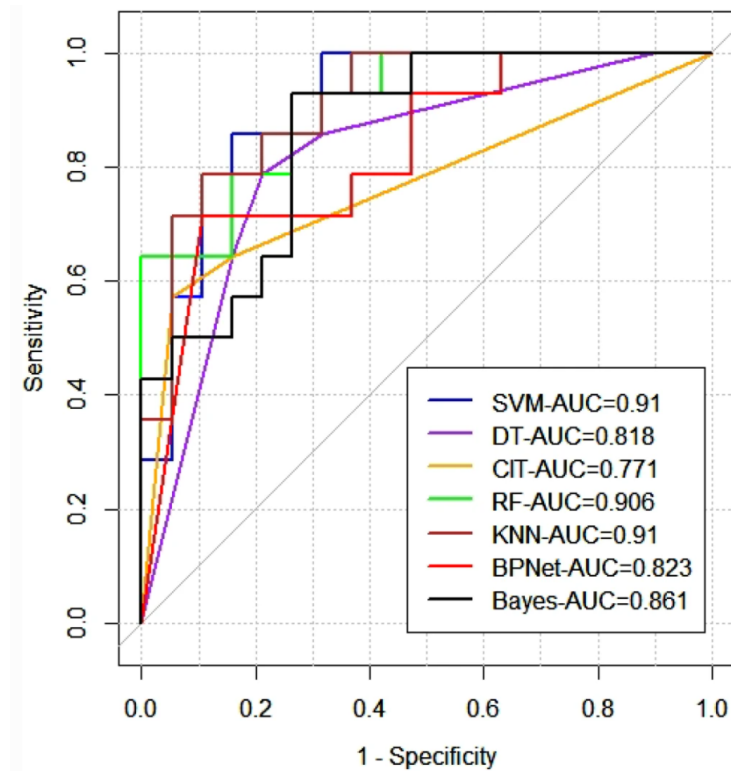


Figure 3.34 – This figure provides the ROC curves for multiple classification algorithms used to predict cardiac surgical operative mortality. The diagonal line shows where the TPR and FPR are equal.

Fonte: (DUAN *et al.*, 2022)

each polymer rings becomes an obstacle to others. When we discussed Figure 3.4, we stated that for the systems simulated using $k^* = 10.0$, the ones where the pressures are higher than $p = 0.12$ already present the confinement effect we have been observing. By making the same analysis for the graphs for the MSD by Ensemble average for the other values of k we have provided in Attachment B, we have come to the conclusion that this pressure threshold is basically the same, where it may vary a bit but not by a significant amount.

With this information in mind, we expect to observe a decrease in the number of polymer rings whose diffusion is classified as Normal Diffusion and an increase in the amount of Confined Diffusion as the pressure gets larger¹¹. We also wish to verify our claim that the springs' constants k^* will not have a significant effect in the number of polymer rings diffusion classification. Figure 3.35 shows the number of polymer rings classified as each diffusion class for each system.

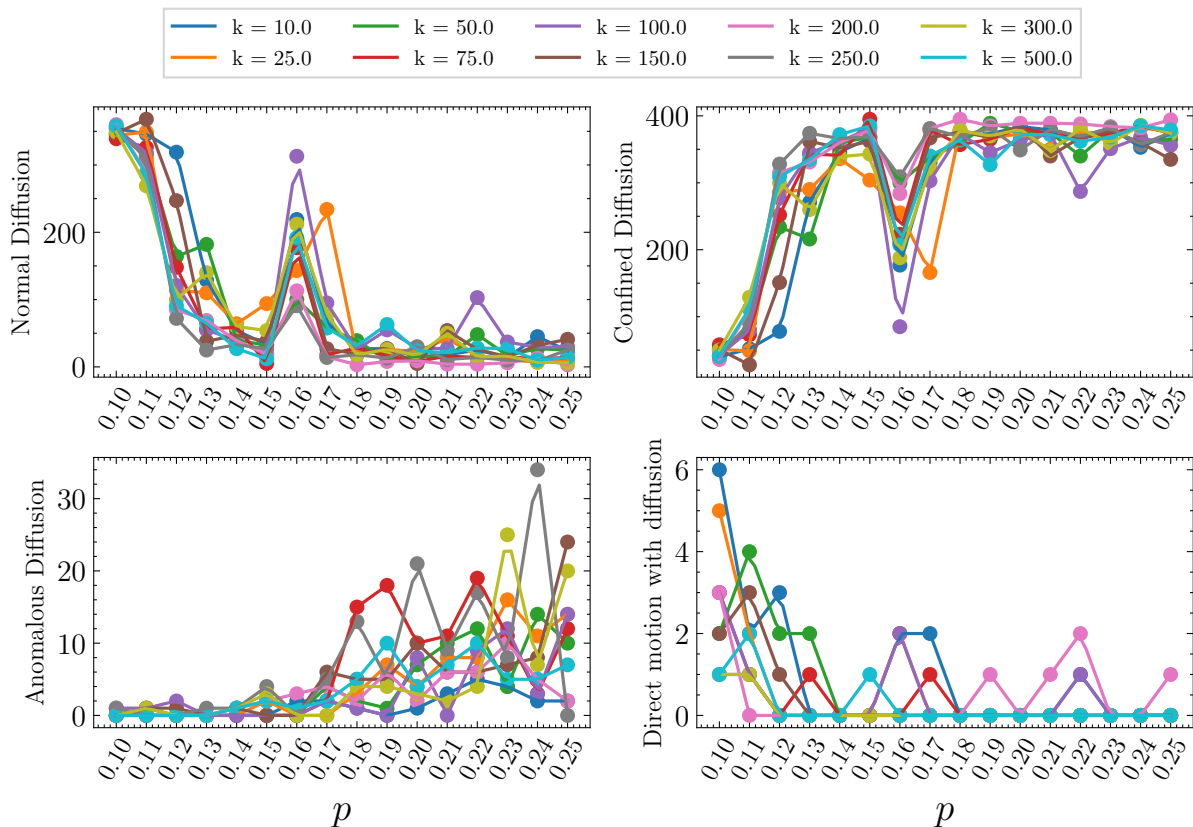


Figure 3.35 – This figure provides, for each system indicated by a combination of p^* in the x -axis and k^* in different colors, the number of polymer rings whose diffusion was classified as either Normal, Anomalous, Confined or Direct motion with diffusion in the y -axis. Each point corresponds to the number of polymer rings associated with each classification and the curves are linear interpolations performed over the data points.

Fonte: The author.

As we can see in Figure 3.35, the number of polymer rings whose diffusion was classified as Normal Diffusion decreases as the pressure p^* increases. In a similar way, we observe a great increase in the number of samples classified as Confined Diffusion - which is what the attributes computed and discussed in Subsection 3.2 pointed to.

¹¹ Perhaps we will observe an increase in Anomalous diffusion as well.

Interestingly, we see that the number of samples classified as each diffusion class does not seem to change much regarding the values of k^* , which is the conclusion we arrived at when discussing the results of the attributes in Subsection 3.2. Lastly, for Anomalous we observe that there is in deed an increase in the number of polymer rings with Anomalous Diffusion, whereas a decrease is perceived for Direct motion with diffusion. This decrease is expected, once the polymer rings get trapped as the pressure increases.

The sharp increase in the number of polymer rings whose diffusion was classified as Normal Diffusion - and, by extension, a decrease for Confined Diffusion - for $p^* = 0.16$ denotes that there is in fact something different about the systems simulated for this specific value of pressure. We showed in Subsection 3.1.3.1 that there is a phase transition happening in the systems for $p^* = 0.16$, which explains, in some sense, the abrupt changes we observe in Figure 3.35. We have now finished discussing the results obtained in the present research project. Our main goal in this dissertation, presenting and showing TrajPy's capabilities, is now complete. In the next chapter, we will provide a review of the entire discussion we made in this work along with the next steps we intend to take in the near future regarding the development of TrajPy and its applications.

4 CONCLUSION

In the present research project, we presented, discussed and improved a physics based feature engineering framework for trajectory analysis called TrajPy. As stated in Chapter 1, the quantitative description of trajectories is crucial to the understanding of the underlying mechanisms of any system, whether the system is based on ideas of Economics, Biology, Physics or any other field of knowledge.

Regarding physical systems, we showed that the concept of diffusion is used to model the motion of systems where the movement of its constituents depends on the concentration of matter in different locations. We described the concept of the MSD and the four basic motion types regarding diffusion: Normal, Confined, Anomalous and Direct Motion with Diffusion. In addition, we demonstrated that the time dependency of the MSD, specifically its time exponent β , may not be sufficient to classify the diffusion of the system as one of the four basic motion types. As a possible solution to this problem, we described the computational approach that TrajPy offers to trajectory analysis as a whole.

In Chapter 2 we described the three components of TrajPy. The first allows us to compute several quantities to describe any trajectory, where this trajectory may be generated in a computer simulation or taken from experiments. These quantities, as we mentioned, are part of the feature engineering procedure, where we transform a single piece of information - in this case, the trajectory being analysed - into several different quantities that can be used in Machine Learning algorithms.

Regarding the first component of TrajPy, we have made contributions by implementing the calculations of fourteen new quantities, in total. The Velocity Autocorrelation Function that measures the similarity between velocity vectors in different times, the Green-Kubo relation that allows us to compute the diffusion coefficient for any trajectory, a statistical description of the velocity, where we compute central tendencies, spreads, the skewness and kurtosis of the velocity distribution and a discrete version of the Fourier Transform that allows us to calculate the main frequencies of the trajectory. These contributions transform TrajPy into a more general trajectory analysis framework that can be used to describe the trajectory in both the time and frequency domain.

The second component of TrajPy allows us to simulate the four basic motion types. We may use these simulation engines as a way to generate the four basic motion types with a range of parameters such as the number of dimensions, displacements, timesteps and more. We can use these simulated trajectories and compute all the attributes described in Subsections 2.1.1 and 2.1.2 to produce a dataset that can be used to classify the diffusion of the system, where we make use of more attributes than just the time exponent β - and, by extension, solve the problem related to diffusion classification we previously described.

The third component of TrajPy consists of two graphical-user interfaces. The first allows us to compute the attributes to describe the trajectory in a way that we do not need to be familiar with Python programming. The second interface - another contribution we have made to TrajPy - was developed to be a possible solution to the bottleneck present in the process of drug discovery. This process usually involves the study of the effects of a specific drug in the physical behavior of rodents, where this study relies on the analysis of the animal's trajectory.

In the literature, the use of computational tracking techniques to extract the animal's trajectory from a video feed has been proposed and put in practice. However, the lack of automated video-based multiple animal tracking techniques - the lack of such techniques is the bottleneck we mentioned - has been a problem that has not been fully solved. As a first step to a solution to this very problem, we have developed the second graphical-user interface using different ideas from the field of Computer Vision.

In Chapter 3 we described the simple drop-like model for biological cells we chose as an application of TrajPy's methodology and performed trajectory analysis and diffusion classification for each system simulated with a constant pressure p^* and a spring constant k^* . As stated in Subsection 3.1.1, the process of cell migration is influenced by the plasticity of the cells involved. To explore the behavior of a simple drop-like model for biological cells for different p^* and k^* , we applied the TrajPy package.

In Section 3.2 we began the process of trajectory analysis by computing many attributes for the trajectory of each polymer ring for each system and then calculated the mean and standard deviation for that attribute. We then used the mean as way to describe the expected value of each attribute for each system and discussed the results we obtained. As a whole, the results of the attributes described a dynamical transition from Normal to Confined diffusion. This transition happened as the pressure increased and did not seem to change much for different values of k^* for the same pressure. For $p^* = 0.16$ we observed a different behavior that came to be explained when we analysed the structure and Thermodynamics of the systems with such pressure, where we observed a phase transition in those systems.

In Subsection 3.3.1 we provided a brief overview of the field of Machine Learning. In Subsection 3.3.2 we used the Random Forest Classifier algorithm to classify the diffusion of each polymer ring of every system into one of the four basic motion types. Regarding the results, we observed that for systems simulated with any value of k^* for the lowest pressures, the number of polymer rings whose diffusion was classified as Normal Diffusion was virtually 400 - the total number of polymer rings in the systems.

For larger values of pressure, say, $p^* = 0.12$ onward, we obtained a sharp decrease in the number of polymer rings for Normal Diffusion and a corresponding increase in the number of Confined Diffusion classifications. We also observed a mild increase in Anomalous Diffusion while the Direct Motion with Diffusion dropped to zero. Every one of these results were expected from the trajectory analysis we performed prior to the diffusion classification procedure.

As for the next steps of the development of TrajPy, we wish to implement new features to make the trajectory analysis an even more general aspect of the software. Regarding the second graphical-user interface, we aim to extend the object tracking algorithm to be used for multiple animals at once, so that TrajPy becomes a solution to the bottleneck in the process of effective chemobehavioral screening in drug discovery and neurotoxicology.

BIBLIOGRAPHY

- AEJMELAEUS-LINDSTRÖM, Petrus *et al.* Jammed architectural structures: towards large-scale reversible construction. **Granular Materials**, v. 18, p. 28, 2016.
- AGARWAL, Basant Lal. **Basic Statistics**. [s. l.]: New Age International, 2006.
- ALDER, Berni Julian; WAINWRIGHT, T. E. Decay of the velocity autocorrelation function. **Physical Review A**, American Physical Society, v. 1, p. 18–21, 1970. DOI: 10.1103/PhysRevA.1.18.
- ALLEN, Michael P.; TILDESLEY, Dominic J. **Computer Simulation of Liquids**. USA: Clarendon Press, 1989.
- ALLEN, Michael P.; TILDESLEY, Dominic J. **Computer Simulation of Liquids: Second Edition**. OUP Oxford, 2017. Disponível em: <https://books.google.com.br/books?id=WFExDwAAQBAJ>.
- ALMAGRO, Jorge *et al.* Tissue architecture in tumor initiation and progression. **Trends in Cancer**, v. 8, n. 6, p. 494–505, 2022. DOI: <https://doi.org/10.1016/j.trecan.2022.02.007>.
- ALTIERI, Ada. The jamming transition. *In*: ALTIERI, Ada. **Jamming and Glass Transitions: In Mean-Field Theories and Beyond**. Cham: Springer International Publishing, 2019. p. 45–64.
- ANDERSEN, Hans C. Molecular dynamics simulations at constant pressure and/or temperature. **The Journal of Chemical Physics**, v. 72, n. 4, p. 2384–2393, 1980. DOI: 10.1063/1.439486.
- ARFKEN, George. **Mathematical Methods for Physicists**. Third. San Diego: Academic Press Inc., 1985.
- ARNOLD, Axel *et al.* Espresso 3.1: Molecular dynamics software for coarse-grained models. *In*: GRIEBEL, Michael; SCHWEITZER, Marc Alexander (Ed.). **Meshfree Methods for Partial Differential Equations VI**. [s. l.]: Springer Berlin Heidelberg, 2013, (Lecture Notes in Computational Science and Engineering, v. 89). p. 1–23.
- BANASZAK, Michal *et al.* Self-organisation in spatial systems-from fractal chaos to regular patterns and vice versa. **PLoS one**, Public Library of Science, v. 10, n. 9, 2015.
- BENHAMOU, Simon. How to reliably estimate the tortuosity of an animal's path:: straightness, sinuosity, or fractal dimension? **Journal of Theoretical Biology**, v. 229, n. 2, p. 209–220, 2004. DOI: <https://doi.org/10.1016/j.jtbi.2004.03.016>.
- BERDIGALIYEV, Nurken; ALJOFAN, Mohamad. An overview of drug discovery and development. **Future Medicinal Chemistry**, v. 12, n. 10, p. 939–947, 2020. DOI: 10.4155/fmc-2019-0307.
- BERRY, Michael Victor; LEWIS, Z. V.; NYE, John Frederick. On the weierstrass-mandelbrot fractal function. **Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences**, The Royal Society, v. 370, n. 1743, p. 459–484, 1980.
- BHARGAVA, Anamika; PULLAGURI, Narasimha; BHARGAVA, Yogesh. Zebrafish as a xenotransplantation model for studying cancer biology and cancer drug discovery. *In*: _____. **Zebrafish Model for Biomedical Research**. [s. l.]: Springer Nature Singapore, 2022. p. 43–59.
- BIROLI, Giulio. Glass and jamming transitions. **Séminaire Poincaré**, v. 13, p. 37–67.
- BIROLI, Giulio. A new kind of phase transition? **Nature Physics**, v. 3, p. 222–223, 2007.

BOAS, Mary L. Mathematical methods in the physical sciences, 2nd ed. **American Journal of Physics**, v. 67, n. 2, p. 165–169, 1999. DOI: 10.1119/1.19218.

BOEKHORST, Veronika te; PREZIOSI, Luigi; FRIEDL, Peter. Plasticity of cell migration in vivo and in silico. **Annual Review of Cell and Developmental Biology**, v. 32, n. 1, p. 491–526, 2016. DOI: 10.1146/annurev-cellbio-111315-125201.

BONYÁR, Attila. Application of localization factor for the detection of tin oxidation with afm. *In: . [S. l.: s. n.]*, 2015.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV: Computer vision with the OpenCV library**. [s. l.]: O'Reilly Media, Inc., 2008.

BRANCADORO, Margherita *et al.* Toward a variable stiffness surgical manipulator based on fiber jamming transition. **Frontiers in Robotics and AI**, v. 6, 2019. DOI: 10.3389/frobt.2019.00012.

BRITO, Carolina; LERNER, Edan; WYART, Matthieu. Theory for swap acceleration near the glass and jamming transitions for continuously polydisperse particles. **Physical Review X**, American Physical Society, v. 8, 2018. DOI: 10.1103/PhysRevX.8.031050.

BROWN, Eric *et al.* Universal robotic gripper based on the jamming of granular material. **PNAS**, v. 107, p. 18809–18814, 2006.

BURADA, Poornachandra Sekhar *et al.* Diffusion in confined geometries. **ChemPhysChem**, v. 10, p. 45–54, 2009. DOI: <https://doi.org/10.1002/cphc.200800526>.

CAMPBELL, Eric; BAGCHI, Prosenjit. A computational model of amoeboid cell swimming. **Physics of Fluids**, v. 29, n. 10, 2017. DOI: 10.1063/1.4990543.

CASPI, Avi; GRANEK, Rony; ELBAUM, Michael. Diffusion and directed motion in cellular transport. **Physical Review E**, American Physical Society, v. 66, 2002. DOI: 10.1103/PhysRevE.66.011916.

CHARBONNEAU, Patrick *et al.* Glass and jamming transitions: From exact results to finite-dimensional descriptions. **Annual Review of Condensed Matter Physics**, v. 8, n. 1, p. 265–288, 2017. DOI: 10.1146/annurev-conmatphys-031016-025334.

CHENG, Wei *et al.* Granular hydrogels for 3d bioprinting applications. **VIEW**, v. 1, n. 3, 2020. DOI: <https://doi.org/10.1002/VIW.20200060>.

CHOWDHURY, Pranali Roy; BANERJEE, Malay; PETROVSKII, Sergei. Canards, relaxation oscillations, and pattern formation in a slow-fast ratio-dependent predator-prey system. **Applied Mathematical Modelling**, v. 109, p. 519–535, 2022. DOI: <https://doi.org/10.1016/j.apm.2022.04.022>.

CODLING, Edward; PLANK, Michael; BENHAMOU, Simon. Random walks in biology. **Journal of the Royal Society**, v. 5, p. 813–834, 2008. DOI: 10.1098/rsif.2008.0014.

COLIN, Jacques *et al.* Evidence for anisotropy of cosmic acceleration. **Astronomy and Astrophysics**, v. 631, 2019. DOI: 10.1051/0004-6361/201936373.

CUNHA, Pedro Vieira Pinto da *et al.* Chaotic lensing around boson stars and kerr black holes with scalar hair. **Physical Review D**, American Physical Society, v. 94, 2016. DOI: 10.1103/PhysRevD.94.104023.

- DAYANANDA, Mysore A. A direct derivation of fick's law from continuity equation for interdiffusion in multicomponent systems. **Scripta Materialia**, v. 210, 2022. DOI: <https://doi.org/10.1016/j.scriptamat.2021.114430>.
- DESPÓSITO, M. A.; VIÑALES, A. D. Subdiffusive behavior in a trapping potential: Mean square displacement and velocity autocorrelation function. **Physical Review E**, American Physical Society, v. 80, 2009. DOI: 10.1103/PhysRevE.80.021111.
- DIETTERICH, Thomas G. Ensemble methods in machine learning. *In: **Multiple Classifier Systems***. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 1–15.
- DUAN, Chongfeng *et al.* Comparison of radiomic models based on different machine learning methods for predicting intracerebral hemorrhage expansion. **Clinical Neuroradiology**, v. 32, p. 1–9, 2022. DOI: 10.1007/s00062-021-01040-2.
- DUY, Pham Thanh; HUONG, Hoang Giang Thi; HONG, Ic-Pyo. Anti-jamming ris communications using dqn-based algorithm. **IEEE Access**, p. 28422–28433, 2022. DOI: 10.1109/ACCESS.2022.3158751.
- EINSTEIN, Albert. **Investigations on the Theory of the Brownian Movement**. [s. l.]: Courier Corporation, 1956.
- ERNST, Dominique; KÖHLER, Jürgen; WEISS, Matthias. Probing the type of anomalous diffusion with single-particle tracking. **Physical Chemistry Chemical Physics**, Royal Society of Chemistry, v. 16, n. 17, p. 7686–7691, 2014.
- ERRINGTON, J. R.; DEBENEDETTI, P. D. Relationship between structural order and the anomalies of liquid water. **Nature**, v. 409, p. 318, 2001.
- FANG, Yuqiang *et al.* An active biomechanical model of cell adhesion actuated by intracellular tensioning-taxis. **Biophysical Journal**, v. 118, 2020. DOI: 10.1016/j.bpj.2020.04.016.
- FICK, Dr. Adolph. V. on liquid diffusion. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, Taylor Francis, v. 10, n. 63, p. 30–39, 1855. DOI: 10.1080/14786445508641925.
- FITZGERALD, Seth G.; DELANEY, Gary W.; HOWARD, David. A review of jamming actuation in soft robotics. **Actuators**, v. 9, n. 4, 2020. DOI: 10.3390/act9040104. Disponible em: <https://www.mdpi.com/2076-0825/9/4/104>.
- FIXMAN, Marshall. Radius of gyration of polymer chains. **The Journal of Chemical Physics**, v. 36, n. 2, p. 306–310, 1962. DOI: 10.1063/1.1732501.
- FRENKEL, Daan; SMIT, Berend. **Understanding Molecular Simulation: From Algorithms to Applications**. Second. San Diego: Academic Press, 2002. v. 1. (Computational Science Series, v. 1).
- FRIEDL, Peter; ALEXANDER, Stephanie. Cancer invasion and the microenvironment: Plasticity and reciprocity. **Cell**, v. 142, p. 992–1009, 2011.
- FU, Libi *et al.* Dynamics of bidirectional pedestrian flow in a corridor including individuals with disabilities. **Physica A: Statistical Mechanics and its Applications**, v. 580, 2021. DOI: <https://doi.org/10.1016/j.physa.2021.126140>.
- FUENTE, Ildelfonso M. De la; LOPEZ, José I. Cell motility and cancer. **Cancers**, v. 12, n. 8, 2020.

GENTHON, Arthur. The concept of velocity in the history of brownian motion. **The European Physical Journal H**, v. 45, p. 49–105, 2020. DOI: 10.1140/epjh/e2020-10009-8.

GHAHRAMANI, Zoubin. Unsupervised learning. *In*: _____. **Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tubingen, Germany, August 4 - 16, 2003, Revised Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 72–112.

GÖTZ, Holger *et al.* Soft particles reinforce robotic grippers: robotic grippers based on granular jamming of soft particles. **Granular Materials**, v. 24, 2022.

HAEGER, Anna *et al.* Cell jamming: Collective invasion of mesenchymal tumor cells imposed by tissue confinement. **Biochimica et Biophysica Acta**, v. 1840, n. 8, p. 2386–2395, 2014. DOI: <https://doi.org/10.1016/j.bbagen.2014.03.020>.

HAN, Chen *et al.* Spatial anti-jamming scheme for internet of satellites based on the deep reinforcement learning and stackelberg game. **IEEE Transactions on Vehicular Technology**, v. 69, n. 5, p. 5331–5342, 2020. DOI: 10.1109/TVT.2020.2982672.

HANNEZO, Edouard; HEISENBERG, Carl-Philipp. Rigidity transitions in development and disease. **Trends in Cell Biology**, v. 32, n. 5, p. 433–444, 2022. DOI: <https://doi.org/10.1016/j.tcb.2021.12.006>.

HANSEN, J.P.; MCDONALD, I.R. **Theory of simpleliquids**. [s. l.]: Elsevier Academic Press, London Burlington, MA, 2006.

HARRIS, Charles R. *et al.* Array programming with numpy. **Nature**, Springer Science and Business Media LLC, v. 585, p. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. **The elements of statistical learning: data mining, inference and prediction**. 2. ed. [s. l.]: Springer, 2009.

HECK, Tommy *et al.* The role of actin protrusion dynamics in cell migration through a degradable viscoelastic extracellular matrix: Insights from a computational model. **PLOS Computational Biology**, Public Library of Science, v. 16, n. 1, p. 1–34, 2020. DOI: 10.1371/journal.pcbi.1007250.

HELBING, Dirk. Traffic and related self-driven many-particle systems. **Review of Modern Physics**, American Physical Society, v. 73, p. 1067–1141, 2001. DOI: 10.1103/RevModPhys.73.1067.

HELMUTH, Jo A *et al.* A novel supervised trajectory segmentation algorithm identifies distinct types of human adenovirus motion in host cells. **Journal of structural biology**, Elsevier Inc, v. 159, n. 3, p. 347–358, 2007.

HELVERT, Sjoerd van; STORM, Cornelis; FRIEDL, Peter. Mechanoreciprocity in cell migration. **Nature Cell Biology**, v. 20, p. 8–20, 2018.

HENRY, Jason; WLODKOWIC, Donald. High-throughput animal tracking in chemobehavioral phenotyping: Current limitations and future perspectives. **Behavioural Processes**, v. 180, 2020. DOI: <https://doi.org/10.1016/j.beproc.2020.104226>.

HIRAKAWA, Hiromasa *et al.* Relationship between self-diffusion and interdiffusion in gaseous systems. **Bulletin of the Chemical Society of Japan**, v. 46, n. 9, p. 2659–2662, 1973. DOI: 10.1246/bcsj.46.2659.

HUBICKA, Katarzyna; JANCZURA, Joanna. Time-dependent classification of protein diffusion types: A statistical detection of mean-squared-displacement exponent transitions. **Physical Review E**, American Physical Society, v. 101, 2020. DOI: 10.1103/PhysRevE.101.022107.

HUET, S'ebastien *et al.* Analysis of transient behavior in complex trajectories: application to secretory vesicle dynamics. **Biophysical journal**, v. 91, n. 9, p. 3542–3559, 2006.

IKEDA, Harukuni *et al.* Jamming with tunable roughness. **Physical Review Letters**, American Physical Society, v. 124, 2020. DOI: 10.1103/PhysRevLett.124.208001.

ILINA, O. *et al.* Cell–cell adhesion and 3d matrix confinement determine jamming transitions in breast cancer invasion. **Nature cell biology**, v. 22, n. 9, p. 1103–1115, 2020.

JACKSON, Meyer B. **Molecular and Cellular Biophysics**. [s. l.]: Cambridge University Press, 2006.

JADHAV, Saurabh *et al.* Variable stiffness devices using fiber jamming for application in soft robotics and wearable haptics. **Soft Robotics**, v. 9, n. 1, p. 173–186, 2022. DOI: 10.1089/soro.2019.0203.

JAEGER, Heinrich M. Celebrating soft matter's 10th anniversary: Toward jamming by design. **Soft Matter**, The Royal Society of Chemistry, v. 11, p. 12–27, 2015. DOI: 10.1039/C4SM01923G.

JASKOWIAK, Pablo A.; COSTA, Ivan G.; CAMPELLO, Ricardo J. G. B. The area under the roc curve as a measure of clustering quality. **Data Mining and Knowledge Discovery**, v. 36, p. 1219–1245, 2022.

JIANG, Yanqun *et al.* Estimation of traffic emissions in a polycentric urban city based on a macroscopic approach. **Physica A: Statistical Mechanics and its Applications**, v. 602, 2022. DOI: <https://doi.org/10.1016/j.physa.2022.127391>.

JIN, Suoqin *et al.* Inference and analysis of cell-cell communication using cellchat. **Nature Communications**, v. 12, 2021. DOI: 10.1038/s41467-021-21246-9.

KANG, Wenying *et al.* A novel jamming phase diagram links tumor invasion to non-equilibrium phase separation. **iScience**, v. 21, 2021.

KATZ, Michael J.; GEORGE, Edwin B. Fractals and the analysis of growth paths. **Bulletin of Mathematical Biology**, v. 47, n. 2, p. 273–286, 1985. DOI: [https://doi.org/10.1016/S0092-8240\(85\)90053-9](https://doi.org/10.1016/S0092-8240(85)90053-9).

KIRKBY, M.J. **The fractal geometry of nature**. [S. l.: s. n.], 1983.

KLAFTER, Joseph; SOKOLOV, Igor M. Anomalous diffusion spreads its wings. **Physics World**, IOP Publishing, v. 18, n. 8, p. 29–32, aug 2005. DOI: 10.1088/2058-7058/18/8/33. Disponível em: <https://doi.org/10.1088/2058-7058/18/8/33>.

KLUMOV, Boris A.; KHRAPAK, Sergey A. Two-body entropy of two-dimensional fluids. **Results in Physics**, v. 17, p. 103020, 2020.

KRAMER, Kelby B.; WANG, Gerald J. Social distancing slows down steady dynamics in pedestrian flows. **Physics of Fluids**, v. 33, n. 10, 2021. DOI: 10.1063/5.0062331.

KRAPF, Diego. Chapter five - mechanisms underlying anomalous diffusion in the plasma membrane. In: KENWORTHY, Anne K. (Ed.). **Lipid Domains**. [s. l.]: Academic Press, 2015, (Current Topics in Membranes, v. 75). p. 167–207.

LACASA, Lucas; CEA, Miguel; ZANIN, Massimiliano. Jamming transition in air transportation networks. **Physica A: Statistical Mechanics and its Applications**, v. 388, n. 18, p. 3948–3954, 2009. DOI: <https://doi.org/10.1016/j.physa.2009.06.005>.

LEE, M. Howard. Fick's law, green-kubo formula, and heisenberg's equation of motion. **Physical Review Letters**, American Physical Society, v. 85, p. 2422–2425, 2000. DOI: 10.1103/PhysRevLett.85.2422.

LENNE, Pierre-François; TRIVEDI, Vikas. Sculpting tissues by phase transitions. **Nature Communication**, v. 13, p. 664, 2022.

LEVESQUE, D.; ASHURST, W. T. Long-time behavior of the velocity autocorrelation function for a fluid of soft repulsive particles. **Physical Review Letters**, American Physical Society, v. 33, p. 277–280, 1974. DOI: 10.1103/PhysRevLett.33.277.

LI, Qian. sctour: a deep learning architecture for robust inference and accurate prediction of cellular dynamics. **bioRxiv**, Cold Spring Harbor Laboratory, 2022. DOI: 10.1101/2022.04.17.488600.

LIMBACH, Hans-Jorg *et al.* Espresso - an extensible simulation package for research on soft matter systems. **Computer Physics Communication**, v. 174, p. 704–727, 2006.

LIU, Andrea J.; NAGEL, Sidney R. Jamming is not just cool any more. **Nature**, v. 396, p. 21–22, 1998.

LIU, Andrea J.; NAGEL, Sidney R. The jamming transition and the marginally jammed solid. **Annual Review of Condensed Matter Physics**, v. 1, n. 1, p. 347–369, 2010. DOI: 10.1146/annurev-conmatphys-070909-104045.

LIU, Tianxin *et al.* A positive pressure jamming based variable stiffness structure and its application on wearable robots. **IEEE Robotics and Automation Letters**, v. 6, n. 4, p. 8078–8085, 2021. DOI: 10.1109/LRA.2021.3097255.

LU, Zhuo; WANG, Wenye; WANG, Cliff. Modeling, evaluation and detection of jamming attacks in time-critical wireless applications. **IEEE Transactions on Mobile Computing**, v. 13, n. 8, p. 1746–1759, 2014. DOI: 10.1109/TMC.2013.146.

MAJMUDAR, T. S. *et al.* Jamming transition in granular systems. **Physical Review Letters**, American Physical Society, v. 98, 2007. DOI: 10.1103/PhysRevLett.98.058001.

MARIANI, Paolo *et al.* Labour market inclusion and economic well-being: a trajectory analysis for some european countries (1995–2019). **Quality & Quantity**, Springer, p. 1–22, 2022. DOI: <https://doi.org/10.1007/s11135-021-01301-9>.

MATEUS, Andre; SAVITSKI, Mikhail M; PIAZZA, Ilaria. The rise of proteome-wide biophysics. **Molecular Systems Biology**, v. 17, n. 7, 2021. DOI: <https://doi.org/10.15252/msb.202110442>.

MICHALET, Xavier. Mean square displacement analysis of single-particle trajectories with localization error: Brownian motion in an isotropic medium. **Physical Review E**, American Physical Society, v. 82, 2010. DOI: 10.1103/PhysRevE.82.041914.

MICHAUD-AGRAWAL, Naveen *et al.* Mdanalysis: A toolkit for the analysis of molecular dynamics simulations. **Journal of Computational Chemistry**, v. 32, n. 10, p. 2319–2327, 2011. DOI: <https://doi.org/10.1002/jcc.21787>.

MJOLSNESS, Eric; DECOSTE, Dennis. Machine learning for science: State of the art and future prospects. **Science**, v. 293, n. 5537, p. 2051–2055, 2001. DOI: 10.1126/science.293.5537.2051.

Moreira-Soares, Maurício. **Multi-Phase-Field Models for Biological Systems**. 2020. Tese (Doutorado) – University of Coimbra - Physics Department, School of Science and Technology, 2020.

Moreira-Soares, Maurício. **Open access training dataset for trajectory classification**. 2022. <https://zenodo.org/record/3627650.YtBRCFvMJhF>.

MOREIRA-SOARES, Maurício *et al.* Adhesion modulates cell morphology and migration within dense fibrous networks. **Journal of Physics: Condensed Matter**, IOP Publishing, v. 32, n. 31, p. 314001, 2020. DOI: 10.1088/1361-648x/ab7c17.

MUQRI, Mohammad Rafiq; WILSON, Eric John; SHAKIB, Javad. A taste of python – discrete and fast fourier transforms. *In: 2015 ASEE Annual Conference amp Exposition*. Seattle, Washington: ASEE Conferences, 2015. <https://peer.asee.org/23464>.

MURAMATSU, Masakuni; IRIE, Tunemasa; NAGATANI, Takashi. Jamming transition in pedestrian counter flow. **Physica A: Statistical Mechanics and its Applications**, v. 267, n. 3, p. 487–498, 1999. DOI: [https://doi.org/10.1016/S0378-4371\(99\)00018-7](https://doi.org/10.1016/S0378-4371(99)00018-7).

NAGATANI, Takashi. Dynamical jamming transition induced by a car accident in traffic-flow model of a two-lane roadway. **Physica A: Statistical Mechanics and its Applications**, v. 202, n. 3, p. 449–458, 1994. DOI: [https://doi.org/10.1016/0378-4371\(94\)90471-5](https://doi.org/10.1016/0378-4371(94)90471-5).

NAGATANI, Takashi. Thermodynamic theory for the jamming transition in traffic flow. **Physical Review E**, American Physical Society, v. 58, p. 4271–4276, 1998. DOI: 10.1103/PhysRevE.58.4271.

NICK, Todd G.; CAMPBELL, Kathleen M. Logistic regression. *In: ____*. **Topics in Biostatistics**. [s. l.]: Humana Press, 2007. p. 273–301.

OHIRA, Toru; SAWATARI, Ryusuke. Phase transition in a computer network traffic model. **Physical Review E**, American Physical Society, v. 58, p. 193–195, 1998. DOI: 10.1103/PhysRevE.58.193.

OLGAR, Handan; JANKE, Wolfhard. Gyration tensor based analysis of the shapes of polymer chains in an attractive spherical cage. **The Journal of chemical physics**, v. 138, 02 2013. DOI: 10.1063/1.4788616.

OLIVEIRA, Fernando A. *et al.* Anomalous diffusion: A basic mechanism for the evolution of inhomogeneous systems. **Frontiers in Physics**, v. 7, 2019. DOI: 10.3389/fphy.2019.00018. Disponível em: <https://www.frontiersin.org/article/10.3389/fphy.2019.00018>.

OLIVEIRA, Fernando A. *et al.* Anomalous diffusion: A basic mechanism for the evolution of inhomogeneous systems. **Frontiers in Physics**, 2019. DOI: 10.3389/fphy.2019.00018.

OSWALD, Linda *et al.* Jamming transitions in cancer. **Journal of Physics D: Applied Physics**, IOP Publishing, v. 50, n. 48, 2017. DOI: 10.1088/1361-6463/aa8e83.

PANCERASA, Mattia *et al.* Reconstruction of long-distance bird migration routes using advanced machine learning techniques on geolocator data. **Journal of The Royal Society Interface**, v. 16, n. 155, 2019. DOI: 10.1098/rsif.2019.0031.

PLOSZAJSKI, Anna R. *et al.* 4d printing of magnetically functionalized chainmail for exoskeletal biomedical applications. **MRS Advances**, v. 4, p. 1361–1366, 2019.

RACCIS, Riccardo *et al.* Confined diffusion in periodic porous nanostructures. **ACS Nano**, v. 5, n. 6, p. 4607–4616, 2011. DOI: 10.1021/nn200767x.

RAMANA, Venkata; SAI, A.; JABARI, Saif Eddin. Power laws and phase transitions in heterogenous car following with reaction times. **Physical Review E**, American Physical Society, v. 103, Mar 2021. DOI: 10.1103/PhysRevE.103.032202.

ROE, Daniel; CHEATHAM, Thomas. Ptraj and cpptraj: Software for processing and analysis of molecular dynamics trajectory data. **Journal of Chemical Theory and Computation**, v. 9, n. 7, p. 3084–3095, 2013. DOI: 10.1021/ct400341p.

RUDNICK, JOSEPH; GASPARI, GEORGE. The shapes of random walks. **Science (American Association for the Advancement of Science)**, The American Association for the Advancement of Science, v. 237, n. 4813, p. 384–389, 1987.

SADATI, Monirosadat *et al.* Glass-like dynamics in the cell and in cellular collectives. **WIREs Systems Biology and Medicine**, v. 6, n. 2, p. 137–149, 2014. DOI: <https://doi.org/10.1002/wsbm.1258>.

SAN-SEGUNDO, Rubén *et al.* Parkinson's disease tremor detection in the wild using wearable accelerometers. **Sensors**, v. 20, n. 20, 2020. DOI: <https://doi.org/10.3390/s20205817>.

SANDEV, Trifce; METZLER, Ralf; CHECHKIN, Aleksei. From continuous time random walks to the generalized diffusion equation. **Fractional Calculus and Applied Analysis**, v. 21, n. 1, p. 10–28, 2018. DOI: [doi:10.1515/fca-2018-0002](https://doi.org/10.1515/fca-2018-0002).

SANTOS, M. A. F. dos; JUNIOR, Luis Menon; CIUS, Danilo. **Superstatistical approach of the anomalous exponent for scaled Brownian motion**. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2206.07820>.

SAXTON, Michael J. Anomalous subdiffusion in fluorescence photobleaching recovery: A monte carlo study. **Biophysical Journal**, v. 81, n. 4, p. 2226–2240, 2001. DOI: [https://doi.org/10.1016/S0006-3495\(01\)75870-5](https://doi.org/10.1016/S0006-3495(01)75870-5). ISSN 0006-3495.

SAXTON, Michael J.; JACOBSON, Ken. Single-particle tracking: Applications to membrane dynamics. **Annual Review of Biophysics and Biomolecular Structure**, v. 26, n. 1, p. 373, 1997.

SCHUETT, Timo *et al.* Dialysis diffusion kinetics in polymer purification. **Macromolecules**, v. 54, n. 20, p. 9410–9417, 2021. DOI: 10.1021/acs.macromol.1c01241.

SEABOLD, Skipper; PERKTOLD, Josef. statsmodels: Econometric and statistical modeling with python. *In: 9th Python in Science Conference*. [S. l.: s. n.], 2010.

SHI, Xiu *et al.* Sik2 promotes ovarian cancer cell motility and metastasis by phosphorylating mylk. **Molecular Oncology**, v. 16, n. 13, p. 2558–2574, 2022. DOI: <https://doi.org/10.1002/1878-0261.13208>.

SPACKMAN, Kent A. Signal detection theory: Valuable tools for evaluating inductive learning. *In: SEGRE, Alberto Maria (Ed.). Proceedings of the Sixth International Workshop on Machine Learning*. San Francisco (CA): Morgan Kaufmann, 1989. p. 160–163. ISBN 978-1-55860-036-2.

STELTZ, E. *et al.* Jamming as an enabling technology for soft robotics. *In: BAR-COHEN, Yoseph (Ed.). Electroactive Polymer Actuators and Devices (EAPAD) 2010*. [s. l.]: SPIE, 2010. v. 7642, p. 640 – 648.

STROGATZ, Steven H. **Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering: With Applications to Physics, Biology, Chemistry, and Engineering**. 2. ed. [s. l.]: Westview Press, 2014. 513 p.

SU, Xiaogang; YAN, Xin; TSAI, Chih-Ling. Linear regression. **WIREs Computational Statistics**, v. 4, n. 3, p. 275–294, 2012. DOI: <https://doi.org/10.1002/wics.1198>.

SUTTON, Richard S.; BARTO, Andrew G. **Reinforcement Learning: An Introduction**. Second. [s. l.]: The MIT Press, 2018.

THOMPSON, Peter *et al.* Experimental analyses of step extent and contact buffer in pedestrian dynamics. **Physica A: Statistical Mechanics and its Applications**, v. 593, 2022. DOI: <https://doi.org/10.1016/j.physa.2022.126927>.

VSOLC, Karel. Shape of a random-flight chain. **Journal of Chemical Physics**, v. 55, p. 335–344, 1971.

VYMĚTAL, Jiří; VONDRÁŠEK, Jiří. Gyration- and inertia-tensor-based collective coordinates for metadynamics. application on the conformational behavior of polyalanine peptides and trp-cage folding. **The Journal of Physical Chemistry A**, v. 115, n. 41, p. 11455–11465, 2011. DOI: [10.1021/jp2065612](https://doi.org/10.1021/jp2065612).

WAGNER, Thorsten *et al.* Classification and segmentation of nanoparticle diffusion trajectories in cellular micro environments. **PLOS ONE**, Public Library of Science, v. 12, n. 1, p. 1–20, 01 2017. Disponível em: <https://doi.org/10.1371/journal.pone.0170165>.

WANG, Guanning *et al.* Wall-following searching or area coverage searching? simulation study of the panic evacuation considering the guidance of a single rescuer. **Physica A: Statistical Mechanics and its Applications**, v. 603, 2022. DOI: <https://doi.org/10.1016/j.physa.2022.127638>.

WANG, Ximing *et al.* Decentralized reinforcement learning based anti-jamming communication for self-organizing networks. *In: 2021 IEEE Wireless Communications and Networking Conference (WCNC)*. [S. l.: s. n.], 2021. p. 1–6.

WANG, Yifan *et al.* Structured fabrics with tunable mechanical properties. **Nature**, v. 596, p. 238–243, 2021.

WEEKS, John D.; CHANDLER, David; ANDERSEN, Hans C. Role of repulsive forces in determining the equilibrium structure of simple liquids. **The Journal of Chemical Physics**, v. 54, n. 12, p. 5237–5247, 1971. DOI: [10.1063/1.1674820](https://doi.org/10.1063/1.1674820).

WEI, Jing *et al.* Machine learning in materials science. **InfoMat**, v. 1, n. 3, p. 338–358, 2019. DOI: <https://doi.org/10.1002/inf2.12028>.

WILLIAMS, Stephen R. *et al.* Velocity autocorrelation functions of hard-sphere fluids: Long-time tails upon undercooling. **Physical Review Letters**, American Physical Society, v. 96, 2006. DOI: [10.1103/PhysRevLett.96.087801](https://doi.org/10.1103/PhysRevLett.96.087801).

WLODKOWIC, Donald. Future prospects of accelerating neuroactive drug discovery with high-throughput behavioral phenotyping. **Expert Opinion on Drug Discovery**, Taylor & Francis, v. 17, n. 4, p. 305–308, 2022. DOI: [10.1080/17460441.2022.2031971](https://doi.org/10.1080/17460441.2022.2031971).

WU, Xiao-Lun; LIBCHABER, Albert. Particle diffusion in a quasi-two-dimensional bacterial bath. **Physical Review Letters**, American Physical Society, v. 84, p. 3017–3020, 2000. DOI: [10.1103/PhysRevLett.84.3017](https://doi.org/10.1103/PhysRevLett.84.3017).

XINGYUAN, Wang; CHAO, Luo; JUAN, Meng. Nonlinear dynamic research on eeg signals in hai experiment. **Applied Mathematics and Computation**, v. 207, n. 1, p. 63–74, 2009. DOI: <https://doi.org/10.1016/j.amc.2007.10.064>.

ZABURDAEV, V.; DENISOV, S.; KLAFTER, J. Lévy walks. **Review of Modern Physics**, American Physical Society, v. 87, p. 483–530, 2015. DOI: 10.1103/RevModPhys.87.483.

ZHANG, Ya-Ru *et al.* Strategies to improve tumor penetration of nanomedicines through nanoparticle design. **Wiley Interdisciplinary Reviews: Nanomedicine and Nanobiotechnology**, v. 11, 2018. DOI: 10.1002/wnan.1519.

ZHOU, Jibiao *et al.* Stability analysis of pedestrian traffic flow in horizontal channels: A numerical simulation method. **Physica A: Statistical Mechanics and its Applications**, v. 587, 2022. DOI: <https://doi.org/10.1016/j.physa.2021.126528>.

ATTACHMENTS

ATTACHMENT A – ALGORITHMS

Code A.1 – Codes for the computation of each attribute.

```

1  import numpy as np
2  from scipy.stats import linregress
3  import trajpy.auxiliar_functions as aux
4  import warnings
5
6  class Trajectory(object):
7      """
8      This is the main class object in trajpy. It can be initialized
9      as a dummy object for calling its functions or you can initialize
10     it with a trajectory array or csv file.
11     """
12     def __init__(self, trajectory=np.zeros((1, 2)), box_length=None, **params
13         ):
14         """
15         Initialization function that can be left blank for using
16         staticmethods.
17         It can be initialized with an array with shape (N, dim)
18         where dim is the number of spatial dimensions plus the time component
19         .
20         The first column must be the time, followed by the x- and y-axis.
21         It also accepts tuples (t, x, y) or csv files.
22
23         The trajectory will be split between the temporal component self._t
24         and the spatial axis self._r.
25
26         :param trajectory: 2D trajectory as a function of time (t, x, y)
27         :param params: use params for passing parameters into np.genfromtxt()
28         """
29         if type(trajectory) == str:
30             trajectory = np.genfromtxt(trajectory, **params)
31
32         if type(trajectory) == np.ndarray:
33             self._t, self._r = trajectory[:, 0], trajectory[:, 1:]
34         elif type(trajectory) == tuple:
35             self._t, self._r = np.asarray(trajectory[0]), np.asarray(
36                 trajectory[1:])
37         else:
38             raise TypeError('trajectory receives an array or a filename as_

```

```

        input.')
```

35 **if** box_length != None:

36 self._r = aux.unfold(self._r[0], self._r, box_length)

37

38 self.ms_d_ta = None # *time-averaged mean squared displacement*

39 self.ms_d_ea = None # *ensemble-averaged mean squared displacement*

40 self.ms_d_ratio = None

41 self.anomalous_exponent = None

42 self.fractal_dimension = None

43 self.eigenvalues = None

44 self.gyration_radius = None

45 self.eigenvalues = None

46 self.eigenvectors = None

47 self.asymmetry = None

48 self.straightness = None

49 self.anisotropy = None

50 self.kurtosis = None

51 self.gaussianity = None

52 self.ms_d_ratio = None

53 self.efficiency = None

54 self.confinement_probability = None

55 self.diffusivity = None

56 self._r0 = None # *maximum distance between any two points of the*
 trajectory

57 self.velocity = None

58 self.velocity_description = None

59 self.frequency_spectrum = None

60

61 **def** compute_features(self):

62 """

63 *Compute every feature for the trajectory saved in self._r.*

64

65 *:return features: return the values of the features as a string.*

66 """

67 self.ms_d_ta = self.ms_d_time_averaged_(self._r, np.arange(len(self._r)
))

68 self.ms_d_ea = self.ms_d_ensemble_averaged_(self._r)

69 self.ms_d_ratio = self.ms_d_ratio_(self.ms_d_ta, n1=2, n2=10)

70 self.anomalous_exponent = self.anomalous_exponent_(self.ms_d_ea, self.
 _t)

71 self.fractal_dimension, self._r0 = self.fractal_dimension_(self._r)

72

```

73     self.gyration_radius = self.gyration_radius_(self._r).get('gyration_
        tensor')
74     self.eigenvalues = self.gyration_radius_(self._r).get('eigenvalues')
75     self.eigenvectors = self.gyration_radius_(self._r).get('eigenvectors'
        )
76
77     self.kurtosis = self.kurtosis_(self._r, self.eigenvectors[:,0])
78     self.anisotropy = self.anisotropy_(self.eigenvalues)
79     self.velocity = self.velocity_(self._r, self._t)
80     self.vacf = self.stationary_velocity_correlation_(self.velocity, self.
        _t, np.arange(int(len(self.velocity)/2)))
81     self.straightness = self.straightness_(self._r)
82     self.gaussianity = self.gaussianity_(self._r)
83     self.efficiency = self.efficiency_(self._r)
84     self.diffusivity = self.green_kubo_(self.velocity, self._t,
85         self.stationary_velocity_correlation_(self.
            velocity, self._t, np.arange(int(len(self.
                velocity)/2))))
86     self.velocity_description = self.velocity_description_(self.velocity)
87     self.frequency_spectrum = self.frequency_spectrum_(self._r, self._t)
88
89     #self.confinement_probability = self.confinement_probability_(2, self.
        diffusivity, self._t[-1])
90
91
92     features = (str(np.round(self.anomalous_exponent, 4)) + ', ' +
93         str(np.round(self.msd_ratio, 4)) + ', ' +
94         str(np.round(self.fractal_dimension, 4)) + ', ' +
95         str(np.round(self.anisotropy, 4)) + ', ' +
96         str(np.round(self.kurtosis, 4)) + ', ' +
97         str(np.round(self.straightness, 4)) + ', ' +
98         str(np.round(self.gaussianity, 4)) + ', ' +
99         str(np.round(self.efficiency, 4)) + ', ' +
100        str(np.round(self.diffusivity, 4)))
101
102     return features
103
104
105     @staticmethod
106     def msd_time_averaged_(spatial_components, tau):
107         """
108         calculates the time-averaged mean squared displacement

```

```

109
110 .. math::
111      $\langle \mathbf{r}_{\tau}^2 \rangle = \frac{1}{T-\tau} \sum_{t=1}^{N-\tau} \|\mathbf{x}_{t+\tau} - \mathbf{x}_t\|^2$ 
112
113     where math: ' $\tau$ ' is the time interval (time lag) between the two
114         positions and math: ' $T$ ' is total trajectory time length.
115
116     :param spatial_components: array containing trajectory spatial
117         coordinates
118     :param tau: time lag, it can be a single value or an array
119     :return msd: time-averaged MSD
120     """
121
122     if type(tau) == int:
123         tau = np.asarray([tau])
124
125     msd = np.zeros(len(tau))
126     time_lag = 0
127     for value in tau:
128
129         dx = []
130
131         for n in range(0, len(spatial_components) - value):
132             dx.append(spatial_components[n + value] - spatial_components[
133                 n])
134
135         dx = np.asarray(dx)
136
137         msd[time_lag] = np.sum(np.power(dx, 2)) / (spatial_components.
138             size - value + 1)
139         time_lag += 1
140
141     return msd
142
143 @staticmethod
144 def msd_ensemble_averaged_(spatial_components):
145     """
146     calculates the ensemble-averaged mean squared displacement
147
148     .. math::
149          $\langle \mathbf{r}^2 \rangle (t) = \frac{1}{N-1} \sum_{n=1}^N$ 

```

```

        
$$N \|\mathbf{x}_n - \mathbf{x}_0\|^2$$

145
146     where  $N$  is the number of trajectories ,  $\mathbf{r}_n(t)$ 
        is the position of the trajectory  $n$  at time  $t$ .
147
148     :param spatial_components: array containing trajectory spatial
        coordinates
149     :return msd: ensemble-averaged msd
150     """
151
152     msd = np.zeros(len(spatial_components))
153     for n in range(0, len(spatial_components)):
154         msd[n] = np.sum(np.power(spatial_components[n] -
        spatial_components[0], 2))
155     msd = msd / (len(spatial_components) - 1)
156
157     return msd
158
159     @staticmethod
160     def msd_ratio_(msd_ta, n1, n2):
161         """
162         Ratio of the ensemble averaged mean squared displacements.
163
164         .. math::
165             \langle r^2 \rangle_{\tau_1, \tau_2} = \frac{\langle r^2 \rangle_{\tau_1}}
166             \langle r^2 \rangle_{\tau_2}} - \frac{\tau_1}{\tau_2}
167
168         with
169
170         .. math::
171             \tau_1 < \tau_2
172
173         :return msd_ratio:
174         """
175
176         msd_ratio = msd_ta[n1]/msd_ta[n2] - n1/n2
177         return msd_ratio
178
179     @staticmethod
180     def anomalous_exponent_(msd, time_lag):
181         """

```

```

182     Calculates the diffusion anomalous exponent
183
184     .. math::
185         \beta = \frac{\partial \log \left( \langle x^2 \rangle \right)}{\partial \log(t)}
186
187     :param msd: mean square displacement
188     :param time_lag: time interval
189     :return: diffusion nomalous exponent
190     """
191
192     msd_log = np.log(msd[1:])
193     time_log = np.log(time_lag[1:])
194
195     x, y = time_log, msd_log
196
197     slope, intercept, r_value, p_value, std_err = linregress(x, y)
198
199     anomalous_exponent = np.round(slope, decimals=2)
200
201     return anomalous_exponent
202
203 @staticmethod
204 def fractal_dimension_(trajectory):
205     """
206     Estimates the fractal dimension of the trajectory
207
208     .. math::
209         \frac{\log(N)}{\log(dNL^{-1})}
210
211     :return fractal_dimension: returns the fractal dimension
212     """
213     dr = np.zeros(np.power(len(trajectory), 2))
214
215     # calculating the distance between each pair of points in the
216     # trajectory
217     n_distance = 0
218     for i_pos in range(0, len(trajectory) - 1):
219         for j_pos in range(i_pos + 1, len(trajectory) - 1):
220             dr[n_distance] = np.sum(np.power(trajectory[i_pos] -
221                                             trajectory[j_pos], 2))
222             n_distance += 1

```

```

221
222     d_max = np.sqrt(np.max(dr)) # maximum distance between any two
           points of the trajectory
223     n_points = trajectory.size
224     length = 0
225     diff = np.zeros(trajectory.shape)
226
227     for i_pos in range(0, len(trajectory) - 1):
228         diff[i_pos] = np.round(trajectory[i_pos + 1], decimals=2) \
229             - np.round(trajectory[i_pos], decimals=2)
230         length += np.sqrt(np.sum(np.power(diff[i_pos], 2)))
231
232     fractal_dimension = np.round(np.log(n_points) / (np.log(n_points)
233         + np.log(d_max * np.power(length, -1))),
           decimals=2)
234
235     return fractal_dimension, d_max
236
237 @staticmethod
238 def gyration_radius_(trajectory):
239     """
240     Calculates the gyration radius tensor of the trajectory
241
242     .. math::
243         R_{mn} = \frac{1}{2N^2} \sum_{i=1}^N \sum_{j=1}^N \left( r_{m}^{\{i\}} - r_{m}^{\{j\}} \right) \left( r_{n}^{\{i\}} - r_{n}^{\{j\}} \right)
244
245     where :math:'N' is the number of segments of the trajectory, :math:
246     '\mathbf{r}_i' is the :math:'i'-th position vector along the
247     trajectory,
248     :math:'m' and :math:'n' assume the values of the corresponding
249     coordinates along the directions :math:'x, y, z'.
250
251     :return gyration_radius: gyration_radius dictionary containing the
252         tensor, eigenvalues in descending order
253         and the corresponding eigenvectors by column
254     """
255
256     dim = trajectory.shape[1] # number of dimensions
257     r_gyr = np.zeros((dim, dim)) # gyration radius tensor
258     r_mean = np.mean(trajectory, axis=0)

```

```

255
256     for m in range(0, dim):
257         for n in range(0, dim):
258             r_gyr[m, n] = np.sum(np.matmul(trajectory[:, m] - r_mean[m],
259                                     trajectory[:, n] - r_mean[n]))
260
261     g_radius = np.sqrt(np.abs(r_gyr/trajectory.size)) #gyration radius
262             tensor
263
264     eigenvalues, eigenvectors = np.linalg.eig(g_radius) #computes the
265             eigenvalues and eigenvectors
266     id = eigenvalues.argsort()[::-1]
267     eigenvalues = eigenvalues[id] #eigenvalues in descending order
268     eigenvectors = eigenvectors[:, id] #eigenvectors corresponding to the
269             descending order
270     gyration_radius = {'gyration_tensor':g_radius,
271                       'eigenvalues':eigenvalues,
272                       'eigenvectors':eigenvectors}
273     return gyration_radius #dictionary
274
275 @staticmethod
276 def asymmetry_(eigenvalues):
277     """
278     Takes the eigenvalues of the gyration radius tensor
279     to estimate the asymmetry between axis.
280
281     .. math::
282         a = - \log\left\{ \frac{( \lambda_1 - \lambda_2 )^2}{ \lambda_1 + \lambda_2 } \right\}
283
284     :param eigenvalues: eigenvalues of the gyration radius tensor
285     :return: asymmetry coefficient
286     """
287
288     if len(eigenvalues) == 2:
289         eigenvalues[::-1].sort() # the eigen values must be in the
290             descending order
291
292         asymmetry = - np.log(1. - np.power(eigenvalues[0] - eigenvalues
293             [1], 2) /
294             (2. * np.power(eigenvalues[0] + eigenvalues
295             [1], 2)))

```



```

290     else :
291         raise IndexError("This_function_is_meant_for_2D_trajectories_only
292             .")
293
294     return asymmetry
295
296     @staticmethod
297     def anisotropy_(eigenvalues):
298         """
299         Calculates the trajectory anisotropy using the eigenvalues of the
300         gyration radius tensor.
301
302         .. math::
303             a^2 = 1 - 3 \frac{\lambda_1 \lambda_2 + \lambda_2 \lambda_3 +
304                 \lambda_3 \lambda_1}{(\lambda_1 + \lambda_2 + \lambda_3)^2}
305
306         """
307         eigenvalues[::-1].sort() # the eigenvalues must be in the descending
308             order
309
310         if len(eigenvalues) == 2:
311             eigenvalues = np.concatenate((eigenvalues, np.array([0.0])), axis
312                 =0)
313         anisotropy = 1. - 3. * ((eigenvalues[0] * eigenvalues[1]
314             + eigenvalues[1] * eigenvalues[2]
315             + eigenvalues[2] * eigenvalues[0])
316             / np.power(np.sum(eigenvalues[:]), 2))
317
318         return anisotropy
319
320     @staticmethod
321     def straightness_(trajectory):
322         """
323         Estimates how much straight is the trajectory
324
325         .. math::
326             S = \frac{\|\mathbf{x}_{N-1} - \mathbf{x}_0\|}{\sum_{i=1}^{N-1} \|\mathbf{x}_i - \mathbf{x}_{i-1}\|}
327
328         :return straightness: measure of linearity
329         """

```

```

327     summation = 0.
328
329     for i_pos in range(1, len(trajectory)):
330         summation += np.sqrt(np.dot(trajectory[i_pos] - trajectory[i_pos
331             - 1],
332                                     trajectory[i_pos] - trajectory[i_pos
333                                         - 1]))
334
335     straightness = np.sqrt(np.dot(trajectory[-1] - trajectory[0],
336                                     trajectory[-1] - trajectory[0]))/
337         summation
338
339     return straightness
340
341 @staticmethod
342 def kurtosis_(trajectory, eigenvector):
343     """
344     We obtain the kurtosis by projecting each position of the trajectory
345     along the main principal eigenvector of the radius of gyration
346     tensor
347     :math: 'r_i^p = \mathbf{r} \cdot \hat{e}_1' and then calculating
348     the quartic moment
349
350     .. math::
351         K = \frac{1}{N} \sum_{i=1}^N \frac{\left( r_i^p - \langle r^p \rangle \right)^4}{\sigma_{r^p}^4}
352
353     where :math: '\langle r^p \rangle' is the mean position of the
354     projected trajectory and :math: '\sigma_{r^p}^2' is the variance.
355     The kurtosis measures the peakiness of the distribution of points in
356     the trajectory.
357
358     :return kurtosis: K
359     """
360     N = len(trajectory)
361     r_projection = np.zeros(N)
362     for n, position in enumerate(trajectory):
363         r_projection[n] = np.dot(position, eigenvector)
364
365     mean_ = r_projection.mean()
366     std_ = r_projection.std()
367     r_projection -= mean_
368     kurtosis = (1./N) * np.sum(np.power(r_projection, 4))/np.power(std_,

```

```

360         4)
361
362     return kurtosis
363
364     @staticmethod
365     def gaussianity_(trajectory):
366         """
367         measure of how close to a gaussian distribution is the trajectory.
368
369         .. math::
370             
$$g(n) = \frac{\langle r_n^4 \rangle}{2 \langle r_n^2 \rangle^2}$$

371
372         :return gaussianity: measure of similarity to a gaussian function
373         """
374         fourth_order = aux.moment_(trajectory, 4)
375         second_order = aux.moment_(trajectory, 2)
376
377         gaussianity = (2/3) * (fourth_order/second_order) - 1
378
379         return gaussianity
380
381     @staticmethod
382     def confinement_probability_(r0, D, t, N=100):
383         """ new
384         Estimate the probability of Brownian particle with
385         diffusivity :math:'D' being trapped in the interval :math:'[-r0, +r0
386         ]' after a period of time t.
387
388         .. math::
389             
$$P(r, D, t) = \int_{-r_0}^{r_0} p(r, D, t) \mathrm{d}r$$

390
391         :param r: position
392         :param D: diffusivity
393         :param t: time length
394         :return probability: probability of the particle being confined
395         """
396         p = np.zeros(N)
397         X = np.linspace(-r0, r0, N)
398         dx = X[1]-X[0]
399         for n, x in enumerate(X):

```

```

399     p[n] = aux.einstein_diffusion_probability(x, D, t)
400     probability = np.sum(p)*dx
401     return 1-probability
402
403     @staticmethod
404     def efficiency_(trajectory):
405         """
406             Calculates the efficiency of the movement, a measure that is
407             related to
408             the straightness.
409
410             .. math::
411                 E = \frac{|\mathbf{x}_{N-1} - \mathbf{x}_0|^2}{(N-1) \sum_{i=1}^{N-1} |\mathbf{x}_i - \mathbf{x}_{i-1}|^2}
412
413             :return efficiency: trajectory efficiency.
414         """
415         den = 0.
416
417         for n in range(1, len(trajectory)):
418             den += np.sum(np.power(trajectory[n] - trajectory[n - 1], 2))
419
420         efficiency = np.sum(np.power(trajectory[-1] - trajectory[0], 2)) / \
421             ((len(trajectory) - 1) * den)
422
423         return efficiency
424
425     @staticmethod
426     def velocity_(position, time):
427         """
428             Computes the velocity associated with the trajectory
429         """
430
431         velocity = np.diff(position, axis=0)/(time[1]-time[0])
432
433         return velocity
434
435     @staticmethod
436     def stationary_velocity_correlation_(velocity, t, taus):
437         """
438             Computes the stationary velocity autocorrelation function by time

```

```

    average
439     .. math:
440          $\langle \vec{v}(t+\tau) \cdot \vec{v}(t) \rangle$ 
441     :param velocity: velocity array
442     :param t: time array
443     :param taus: single or array of non-negative integer values
         representing the time lag
444     :return time_averaged_corr_velocity: velocity autocorrelation
         function output
445     """
446     time_averaged_corr_velocity = np.zeros(len(taus))
447     N = len(velocity)
448     for tau in taus:
449         time_averaged_corr_velocity[tau] = (np.sum(np.einsum('ij, ij -> i',
450             np.take(a=velocity, indices=np.arange(0, N-tau)+tau, axis=0),
451             np.take(a=velocity, indices=np.arange(0, N-tau), axis=0))) +
452             time_averaged_corr_velocity[tau-1]) * (t[1]-t[0]) / (N-tau)
453     return time_averaged_corr_velocity
454
455     @staticmethod
456     def green_kubo_(velocity, t, vacf):
457         """
458             Computes the generalised Green-Kubo's diffusion constant
459             :return diffusivity: diffusion constant obtained by the Green-
         Kubo relation
460         """
461
462         diffusivity = 0.
463         N = len(velocity)
464         dt = t[1] - t[0]
465         diffusivity = sum(vacf) * (dt / velocity.shape[1])
466         return diffusivity
467
468     @staticmethod
469     def velocity_description_(velocity):
470         """
471             Computes the main features of the velocity distribution: mean,
         median, mode, variance,
472             standard deviation, range, skewness and kurtosis
473
474             :param velocity: velocity array
475             return velocity_description: returns a dictionary where the

```



```

511         'skewness': skewness}
512     return velocity_description
513
514     @staticmethod
515     def frequency_spectrum_(position, t, freq_threshold):
516         """
517         Computes the Frequency Spectrum using the Fast Fourier Transform
518         algorithm
519         param position: spatial coordinates
520         param t: time
521         return frequency_spectrum_: returns a dictionary containing the
522         dominant frequency and the amplitude associated with it,
523         the mean and main frequencies, the main amplitude and two arrays -
524         one contains the frequencies and the other the amplitudes.
525         """
526         dt = t[1] - t[0]
527         n = len(t)
528         yvalues = position
529         xvalues = range(len(yvalues))
530         yvalues_detrended = np.zeros(shape=yvalues.shape)
531         for col in range(yvalues.shape[1]):
532             z1 = np.polyfit(xvalues, yvalues[:, col], deg=1)
533             p1 = np.poly1d(z1)
534             yvalues_detrended[:, col] = yvalues[:, col] - p1(xvalues)
535         fourier = np.fft.fft(yvalues_detrended, axis=0, n=n)
536         limit = np.arange(1, np.floor(n/2), dtype='int')
537         power = 2*np.abs(fourier)/n
538         power = power[1:max(limit), :]
539         f = (1/(dt*n))*np.arange(n)
540         f = f[1:max(limit)]
541         dominant_frequency = f[np.argmax(power[:max(limit)], axis=0)]
542         dominant_amp = np.max(power[:max(limit)], axis=0)
543         main_frequencies = np.empty(shape=yvalues_detrended.shape[1], dtype=
544             object)
545         main_amplitudes = np.empty(shape=yvalues_detrended.shape[1], dtype=
546             object)
547         mean_frequency = np.zeros(yvalues_detrended.shape[1])
548         for col in range(yvalues_detrended.shape[1]):
549             indice = np.where(power[:, col] >= freq_threshold)
550             main_amplitudes[col] = power[indice, col]
551             main_frequencies[col] = f[indice]
552             mean_frequency[col] = np.mean(indice)

```

```

548     frequency_spectrum = { 'dominant_frequency': dominant_frequency,
549                           'dominant_amplitude': dominant_amp,
550                           'mean_frequency': mean_frequency,
551                           'main_frequencies': main_frequencies,
552                           'main_amplitudes': main_amplitudes,
553                           'x': f,
554                           'y': power}
555     return frequency_spectrum

```

Code A.2 – Codes for the simulation of trajectories for the four basic motion types.

```

1  import numpy as np
2  def weierstrass_mandelbrot(t, n_displacements, alpha):
3      """
4      Calculates the weierstrass mandelbrot function
5
6      .. math::
7          W(t) = \sum_{n=-\infty}^{\infty} \frac{\cos(\phi_n)}{(\gamma^n t^\alpha + \phi_n)} \gamma^{n\alpha/2} , .
8
9      :param t: time step
10     :param n_displacements: number of displacements
11     :param alpha: anomalous exponent
12     :return: anomalous step
13     """
14     gamma = np.sqrt(np.pi)
15     t_star = (2. * np.pi * t) / n_displacements
16
17     wsm = 0.
18
19     for iteration in range(-8, 49): # [-8, 48]
20         phi = 2. * np.random.rand() * np.pi
21         wsm += (np.cos(phi) - np.cos(np.power(gamma, iteration) * t_star +
22         phi)) / \
23             (np.power(gamma, iteration * (alpha / 2.)))
24
25     return wsm
26
27 def anomalous_diffusion(n_steps, n_samples, time_step, alpha):
28     """
29     Generates an ensemble of anomalous trajectories.

```



```

30     :param n_steps: total number of steps
31     :param n_samples: number of simulations
32     :param time_step: time step
33     :param alpha: anomalous exponent
34     :return x, y: time, array containing N_sample trajectories with Nsteps
35     """
36     x = np.zeros(n_steps) * time_step
37     y = np.zeros((n_steps, n_samples))
38
39     for i_sample in range(0, n_samples):
40
41         for i_step in range(0, n_steps):
42             t = i_step * time_step
43             y[i_step, i_sample] = weierstrass_mandelbrot(t, n_steps, alpha=
44                 alpha)
45             x[i_step] = t
46
47         if n_samples == 1:
48             y = y.transpose()[0]
49
50     return x, y
51
52 def normal_distribution(u, D, dt):
53     """
54     This is the steplength probability density function for normal diffusion.
55
56     :param u: absolute distance travelled by the particle durint the time
57     interval dt
58     :param D: diffusivity
59     :param dt: time interval
60     :return pdf: probability density function
61
62     """
63     diff = 4. * D * dt
64     pdf = ((2. * u) / np.sqrt(diff)) * np.exp(-np.power(u, 2) / diff)
65     return pdf
66
67 def normal_diffusion(n_steps, n_samples, dx, y0, D, dt):
68     """
69     Generates an ensemble of normal diffusion trajectories.

```

```

70
71     :param n_steps: total steps
72     :param n_samples: number of trajectories
73     :param dx: maximum step length
74     :param y0: starting position
75     :param D: diffusivity
76     :param dt: time step
77     :return x, y: time, array containing N_samples trajectories with N_steps
78     """
79
80     y = np.zeros((n_steps, n_samples))
81     x = np.linspace(0, n_steps, n_steps)
82     y[0, :] = y0
83
84     for i_sample in range(0, n_samples):
85         i_step = 1
86         while True:
87
88             if i_step >= n_steps:
89                 break
90
91             random_number = np.random.rand()
92             u = (0.5 - random_number) * dx # step length and direction
93             if random_number >= normal_distribution(np.abs(u), D, dt):
94                 y[i_step, i_sample] = y[i_step-1, i_sample] + u
95
96             i_step += 1
97     return x, y
98
99
100 def confined_diffusion(radius, n_steps, n_samples, dx, y0, D, dt):
101     """
102     Generates trajectories under confinement.
103
104     :param radius: confinement radius
105     :param n_steps: number of displacements
106     :param n_samples: number of trajectories
107     :param dx: displacement
108     :param y0: initial position
109     :param D: diffusion coefficient
110     :param dt: time step
111     :return x, y: time, array containing N_samples trajectories with N_steps

```

```

112     """
113     y = np.zeros((n_steps, n_samples))
114     x = np.linspace(0, n_steps, n_steps)
115     y[0, :] = y0
116     sub_step = 0.0
117     for i_sample in range(0, n_samples):
118
119         for i_step in range(0, n_steps):
120
121             sub_x, sub_y = normal_diffusion(n_steps=100, n_samples=1, dx=dx,
122                                             y0=sub_step, D=D, dt=dt)
123
124             if sub_y[-1] < radius:
125                 t = i_step * dt
126                 y[i_step, i_sample] = sub_y[-1]
127                 sub_step = sub_y[-1]
128                 x[i_step] = t
129
130     return x, y
131
132 def superdiffusion(velocity, n_steps, n_samples, y0, dt):
133     """
134     Generates direct diffusion trajectories.
135     Combine pairwise with normal diffusion components.
136
137     :param velocity: constant velocity
138     :param n_steps: number of time steps
139     :param n_samples: number of trajectories
140     :param y0: initial position
141     :param dt: time interval
142     :return x, y: time, array containing N_samples trajectories with N_steps
143     """
144     y = np.zeros((n_steps, n_samples))
145     x = np.linspace(0, n_steps, n_steps)
146     y[0, :] = y0
147
148     for i_sample in range(0, n_samples):
149
150         for i_step in range(1, n_steps):
151             y[i_step, i_sample] = y[i_step-1, i_sample] + velocity * dt
152             x[i_step] = i_step * dt

```

```

153
154     return x, y
155
156
157 def save_to_file(y, param, path):
158     """
159     Saves the trajectories to a file.
160
161     :param y: trajectory array
162     :param param: a parameter that characterizes the kind of trajectory
163     :param path: path to the folder where the file will be saved
164     """
165     dims = ['x', 'y']
166     np.savetxt(path + '/traj_' + str(param) + '.csv', y, delimiter=',',
167               header='t, ' + (", ".join(dims[:y.shape[1]] * int((y.shape[1]) / 2))), comments=
                '')

```

Code A.3 – Code for single object tracking.

```

1 import cv2
2 import numpy as np
3
4
5 def capture(kind, camera, file_name, number1, number2):
6     if kind == 'live':
7         cap = cv2.VideoCapture(camera)
8     elif kind == 'rec':
9         cap = cv2.VideoCapture(str(camera))
10    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
11    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
12    fps = cap.get(cv2.CAP_PROP_FPS)
13
14    salvar = cv2.VideoWriter(str(file_name) + '.mp4', cv2.VideoWriter_fourcc(*'
        mp4v'), fps, (width, height), isColor=True)
15    back_sub = cv2.createBackgroundSubtractorKNN()
16    kernel = np.ones((30,30), np.uint8)
17    arquivo = open(str(file_name) + '.txt', 'w')
18    numero_frames = 0
19    while (True):
20        numero_frames += 1
21        ret, frame = cap.read()
22        if not ret:

```

```

23         break
24
25
26         fg_mask = back_sub.apply(frame)
27
28         fg_mask = cv2.morphologyEx(fg_mask, cv2.MORPH_CLOSE, kernel)
29
30         fg_mask = cv2.medianBlur(fg_mask, 5)
31
32         _, fg_mask = cv2.threshold(fg_mask, 127, 255, cv2.THRESH_BINARY)
33
34         fg_mask_bb = fg_mask
35         contours, hierarchy = cv2.findContours(fg_mask_bb, cv2.RETR_TREE, cv2.
36             CHAIN_APPROX_SIMPLE)[-2:]
37         areas = [cv2.contourArea(c) for c in contours]
38
39         if len(areas) < 1:
40
41             continue
42
43         else :
44
45             max_index = np.argmax(areas)
46
47             cnt = contours[max_index]
48             x,y,w,h = cv2.boundingRect(cnt)
49             cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255), 3)
50
51             if w > h:
52                 px_per_cm = w/float(number1)
53             elif w < h:
54                 px_per_cm = h/float(number1)
55
56             x2 = x + int(w/2)
57             y2 = y + int(h/2)
58             arquivo.write(str(np.round(numero_frames/fps, 2))+', '+str(np.round(x2/
59                 px_per_cm, 1))+', '+str(np.round(y2/px_per_cm, 1))+'\n')
60
61             cv2.circle(frame, (x2,y2), 4, (0,255,0), -1)
62
63             text = "x:␣" + str(np.round(x2/px_per_cm, 1)) + ", y:␣" + str(np.round(

```

```

        y2/px_per_cm,1))
63     cv2.putText(frame, text, (x2, y2), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0
        ,255), 2)
64
65     fps_text = 'Tempo_=_ ' + str(np.round(numero_frames/fps,2))
66     cv2.putText(frame, fps_text, (120,80), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
        (0,0,255), 2)
67
68     stop_text = 'Press_"q"_to_stop_tracking'
69     cv2.putText(frame, stop_text, (120,120), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
        (0,0,255), 2)
70
71     salvar.write(frame)
72     cv2.line(frame, (0,0), (511,0), (255,0,0), 5)
73     cv2.line(frame, (0,0), (0,511), (255,0,0), 5)
74     cv2.imshow('frame', frame)
75     if cv2.waitKey(1) & 0xFF == ord('q'):
76         break
77
78     cap.release()
79     cv2.destroyAllWindows()
80     arquivo.close()
81     salvar.release()
82
83 if __name__ == '__main__':
84     capture()

```

Code A.4 – ESPResSo code of the simulation.

```

1 #
   #####
2 #####          MD simulation of a 2D cell model
   #####
3 #####          Script by J. R. Bordin - Apr 2022
   #####
4 #####          Powered by ESPRESSO
   #####
5 #
   #####

```

```

6 set systemTime [clock seconds]
7 puts "The initial time is: [clock format $systemTime -
  format %H:%M:%S]"
8 puts "The initial date is: [clock format $systemTime -
  format %D]"
9
10
11 set pi [expr 4.*atan (1.0) ];#set value of pi
12 t_random seed [ pid ] ;#starts the random number generator
13 set sigma 1.0 ;#distance unit, in micrometers
14
15 #Set the simulation box
16 set box_lx 210.0 ;# in units of sigma
17 set box_ly 210.0 ;# in units of sigma
18 set box_lz 210.0
19 setmd box_l $box_lx $box_ly $box_lz;#defines the initial
  simulation box
20 setmd periodic 1 1 0 ;#PBC in x and y directions
21 set Atot [expr $box_lx*$box_ly]
22
23 #Set the cell radius and number of cells
24 set radius 3.0 ;#cell radius in units of sigma
25 set n_cell 400 ;#total number of cells
26 set Acell [expr $pi*($radius+0.5)*($radius+0.5)] ;#are of
  one cell
27 set Acell [expr $n_cell*$Acell] ;#area occupied by the
  cells
28 set kstr 500.0 ;#coil strength for the bond between the
  central ghost bead and the ring. It controls the cell
  deformation
29
30 cellsystem domain_decomposition
31
32
33 #set the temperature and pressure range
34 set temp 0.50
35 set p_ini 0.16; #initial pressure
36 set p_fin 0.30; #final pressure
37 set dp 0.01 ;#temperature step

```

```
38 set Np [expr (( $\$p\_fin - \$p\_ini$ )/ $\$dp$ )+1] ;#number of points in
    the isotherm
39 set gamma_0 1.0 ;#same dumping parameter from Langevin/DPD
    thermostat
40 set gamma_v 0.001 ;#coupling parameter for the volume
    control
41 set piston_mass 0.0001 ;#mass of the piston that will
    control the pressure
42 setmd skin 0.4;#for thermostat
43
44
45 #set the integration data
46 set time_step 0.01
47 setmd time_step  $\$time\_step$ ; #MD time step
48 #warm up steps
49 set integ_steps_warm 1000000
50 #equilibration steps
51 set integ_steps_equil 1000000
52 #production cycles
53 set integ_steps 25000
54 set total_cycles 400
55 set run_time [expr  $\$integ\_steps * \$total\_cycles * \$time\_step$ ]
56
57
58
59 #each species needs a number to "define" his type. here I
    set a number to each species.
60 set cell 0 ;#the polymer ring
61 set ghost 1 ;#the central bead
62 set insert 2 ;#for insertion try
63
64
65
66 #####defining the bonded interactions
67
68
69 #bond interaction for two neighbour cell beads
70 set sig 1.0 ;#size of each bead
71 set ring 21
```



```

72 set lambda 1.0
73 inter $ring harmonic 100.0 $lambda [expr 3.*$lambda]
74
75 #bond with the central bead
76 set rigid 22
77 inter $rigid harmonic $kstr $radius [expr 3.*$radius]
78
79 ##### defining the non-
      bonded interactions
80
81 set WCA_cut [expr pow(2,1./6.)] ;#repulsive WCA interaction
82 set eps 1.0 ;#interaction parameter
83
84 inter $cell $cell lennard-jones 1.0 1.0 $WCA_cut auto
85
86 #placing the ring-like cell
87 set nions_per_ring [expr int(((2.*$pi*$radius)/$lambda))]
88 set teta [expr 2.0*$pi/($nions_per_ring-1)]
89 set innerangle 41 ;#inner angular interaction to keep the
      ring structure
90 set k_angle 100
91 inter $innerangle angle $k_angle $teta
92 set ns 0
93 for {set cell_counter 0} { $cell_counter < $n_cell } {incr
      cell_counter 1} {
94     #insert the i-th ring
95     if {$cell_counter == 0} {
96         set posy0 [expr $box_ly*[t_random]]
97         set posx0 [expr $box_lx*[t_random]]
98         set posz0 [expr $box_lz*0.5]
99         part $ns pos $posx0 $posy0 $posz0 q 0 type $ghost mass
            1.0 fix 0 0 1;# first include the central bead for
            each ring
100        part $ns mol $cell_counter
101        }
102        if {$cell_counter > 0} {
103            set overlap 1
104            set try 0
105            while { $overlap == 1} {

```

```

106     incr try 1
107     if {$try > 100000} {
108     puts "more than 100000 attempts were made to include
        the cell $cell_counter"
109     puts "try to put fewer cells...bye bye"
110     exit
111     }
112     set posy0 [expr $box_ly*[t_random]]
113     set posx0 [expr $box_lx*[t_random]]
114     set posz0 [expr $box_lz*0.5]
115     part $ns pos $posx0 $posy0 $posz0 q 0 type $ghost
        mass 1.0 fix 0 0 1;# first include the central
        bead for each ring
116     part $ns mol $cell_counter
117     set overlap_test [analyze mindist $ghost $ghost]
118 #     puts "$overlap_test"
119     if {$overlap_test > [expr 2.*$radius+0.5]} {
120     set overlap 0
121     }
122
123 }
124 }
125 set gid $ns
126 set ns [expr $ns+1]
127 for {set j 1} {$j < [expr $nions_per_ring]} {incr j
    1} {
128 #insert the ring
129 set posz [expr $box_lz*0.5]
130 set posy [expr $posy0 +($radius)*cos($j*$teta)]
131 set posx [expr $posx0 +($radius)*sin($j*$teta)]
132 part $ns pos $posx $posy $posz q 0 type $cell mass 1.0
        fix 0 0 1
133 part $ns mol $cell_counter
134 part $ns bond $rigid $gid ;#bond the bead with the inner
        monomer
135 if { $j > 1 } {
136     part $ns bond $ring [expr $ns-1]
137     part $ns exclude [expr $ns-1]
138     part $gid bond $innerangle [expr $ns-1] $ns;#bond

```

```

                two adjacent beads with the inner monomer by a
                angular constraint
139     }
140     set ns [expr $ns+1]
141     }
142 }
143
144 for {set j 0 } { $j < [setmd n_part] } {incr j [expr
    $nions_per_ring]} {
145     part [expr $j+1] bond $ring [expr $nions_per_ring+$j-1]
146     part $j bond $innerangle [expr $j+1] [expr $j+1]
147     part [expr $j+1] exclude [expr $nions_per_ring+$j-1]
148 }
149
150 set n_cell_part [setmd n_part]
151 set n_part [setmd n_part]
152
153 set systemTime [clock seconds]
154 puts "End_of_insertion_step:[clock_format $systemTime -
    format %H:%M:%S]"
155
156 analyze set chains 0 $n_cell $nions_per_ring
157
158 set Nmolecules [setmd n_part]
159 set id "k-$kstr"
160 set rhofile [open "rho-$id.dat" "w"] ;#open the rho x p
    file
161 set rg_file [open "rg-$id.dat" "w"];#open the rg x p file
162
163 ##### Initial warm steps at the lowest
    pressure #####
164 thermostat set npt_isotropic $temp $gamma_0 $gamma_v ;# set
    the NPT simulation
165 integrate set npt_isotropic $p_ini $piston_mass 1 1 0
166 inter forcecap 200.0
167 integrate $integ_steps_warm
168 inter forcecap 0
169 ##### End of Warm steps
    #####

```

```

170
171 #simulations along the isotherm
172 for { set Pcount 0 } { $Pcount < $Np } { incr Pcount 1 } {
173     #define a random number for each pressure
174     t_random seed [ pid ]
175     #define the pressure for this simulation
176     set press [format %1.4g [expr $p_ini+$Pcount*$dp]]
177     #sets the thermostat
178     thermostat set npt_isotropic $temp $gamma_0 $gamma_v ;#
        set the NPT simulation
179     integrate set npt_isotropic $press $piston_mass 1 1 0
180     #open the files
181     set id "k-$kstr-p-$press"
182     set xyzfile [open "snap-$id.xyz" "w" ] ;#open the xyz
        file
183     set trajfile [open "traj-$id.xyz" "w" ] ;#open the CM
        trajectory file
184     set velfile [open "vel-$id.dat" "w" ] ;#open the CM
        velocity file
185     puts $velfile [expr ($n_cell*$total_cycles)]
186     set obsfile [open "obs-$id.dat" "w" ] ;#open the
        observable file
187
188     ##### Equilibration steps
        #####
189     integrate $integ_steps_equil
190     set equil_time [setmd time]
191     #print the initial positions to the trajectory file
192     for {set nion 0} { $nion < [setmd n_part] } { incr nion
        } {
193     set x1 [expr $sigma*[lindex [part $nion print
        folded_position] 0]]
194     set y1 [expr $sigma*[lindex [part $nion print
        folded_position] 1]]
195     set z1 [expr $sigma*[lindex [part $nion print
        folded_position] 2]]
196     if { [lindex [part $nion ] 6] == $ghost } {
197         puts $trajfile "␣[expr␣([setmd␣time]␣-␣$equil_time)]
        ␣␣␣$x1␣␣␣$y1␣␣␣$z1"

```

```

198     flush $trajfile
199 }
200 }
201 ##### End of Equilibration steps
202 #####
203 puts $obsfile "\#Langevin Dynamics simulation of 2D
204     cells in a solvent"
205 puts $obsfile "\#Powered by Espresso! - utcl script by J
206     . R. Bordin - April 22"
207 puts $obsfile "
208     \#-----
209     "
210 puts $obsfile "\# $n_cell cells"
211 puts $obsfile "\# Total number of particles is [setmd
212     n_part]"
213 puts $obsfile "\# Temperature = $temp and damping
214     parameter = $gamma_0 and pressure = $press"
215
216 set vel [observable new particle_velocities type [list
217     $ghost]]
218 set vacf [correlation new obs1 $vel corr_operation
219     scalar_product tau_max [expr 2000.*$time_step] dt [
220     expr $time_step]]
221 correlation $vacf autoupdate start
222
223 #####main simulation steps - production
224     of results
225     #####
226 set rho 0.0
227 set rgsum 0.0
228 for {set i 1} { $i <= $total_cycles} { incr i} {
229
230 integrate $integ_steps
231
232 analyze append
233
234 set rg [analyze rg 0 $n_cell $nions_per_ring]

```

```

225 set rg2 [expr [lindex $rg 0]]
226 set rgsum [expr $rgsum+$rg2]
227 set lxnow [expr [lindex [setmd box_1] 0]]
228 set lynow [expr [lindex [setmd box_1] 1]]
229 set rho [expr $rho + ($n_cell/($lxnow*$lynow))]
230 puts $obsfile "[expr ([setmd time]-[setmd time])][
    expr $n_cell/($lxnow*$lynow)]$rg2"
231 flush $obsfile
232
233 puts $xyzfile "    $n_cell_part"
234 puts $xyzfile "    Atoms"
235 for {set nion 0} { $nion < [setmd n_part] } { incr nion
    } {
236     set x1 [expr $sigma*[lindex [part $nion print
        folded_position] 0]]
237     set y1 [expr $sigma*[lindex [part $nion print
        folded_position] 1]]
238     set z1 [expr $sigma*[lindex [part $nion print
        folded_position] 2]]
239     if { [lindex [part $nion ] 6] == $cell } {
240     puts $xyzfile "O    $x1    $y1    $z1"
241     }
242     if { [lindex [part $nion ] 6] == $ghost } {
243     puts $xyzfile "H    $x1    $y1    $z1"
244     set vx1 [expr [lindex [part $nion print v] 0]]
245     set vy1 [expr [lindex [part $nion print v] 1]]
246     set v2 [expr $vx1*$vx1+$vy1*$vy1]
247     puts $velfile "[expr (pow($v2,1./2.))]"
248     flush $velfile
249     puts $trajfile "[expr ([setmd time]-[setmd time])][
        $x1    $y1    $z1"
250     flush $trajfile
251     }
252
253     flush $xyzfile
254 }
255 }
256 ##### end of main steps
257

```

```
258 correlation $vacf write_to_file "vacf-cor-$id.dat"
259
260 set systemTime [clock seconds]
261 puts "Production_end_time_is:[clock_format_$systemTime
      _format_%H:%M:%S]"
262 puts "Production_end_date_is:[clock_format_$systemTime
      _format_%D]"
263
264
265 #close $vacf_file
266 close $xyzfile
267 close $velfile
268 close $obsfile
269 puts $rhofile "$press_[expr_$rho/$total_cycles]"
270 flush $rhofile
271 puts $rg_file "$press_[expr_$rgsum/$total_cycles]"
272 flush $rg_file
273 }
274 #####end of program#####
275 exit
```


ATTACHMENT B – SUPPLEMENTARY MATERIAL

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 25.0$

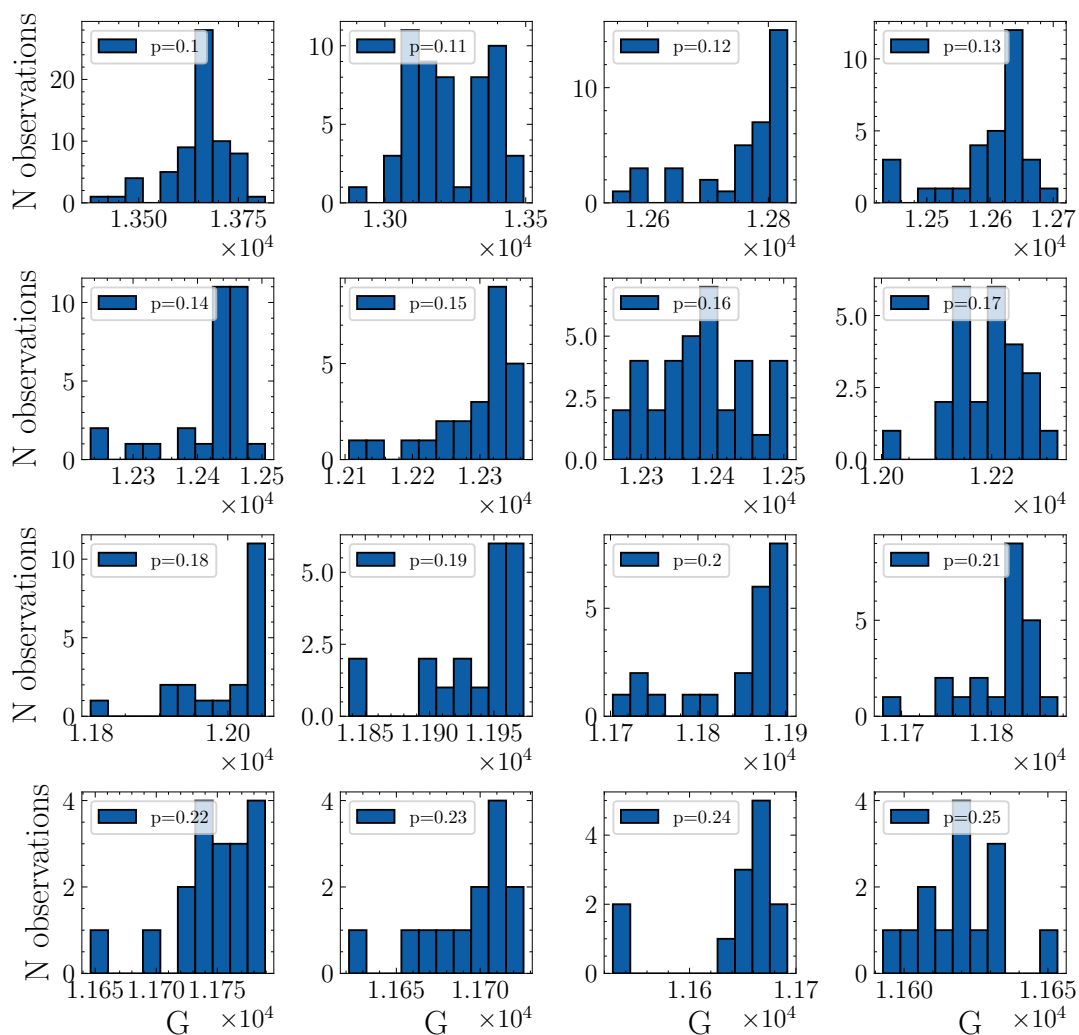


Figure B.1 – This figure the histograms for gaussianity values larger than one for $k = 25.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 50.0$

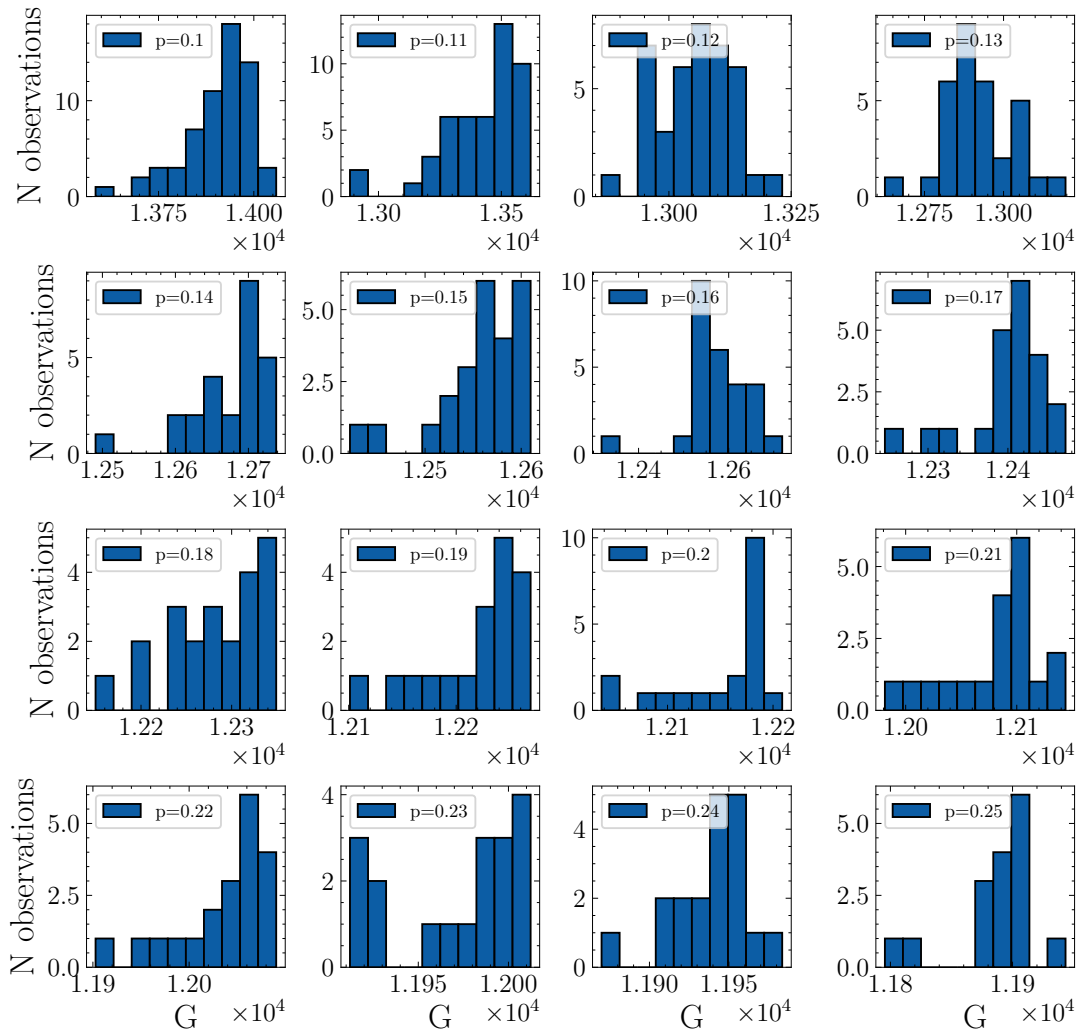


Figure B.2 – This figure the histograms for gaussianity values larger than one for $k = 50.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 75.0$

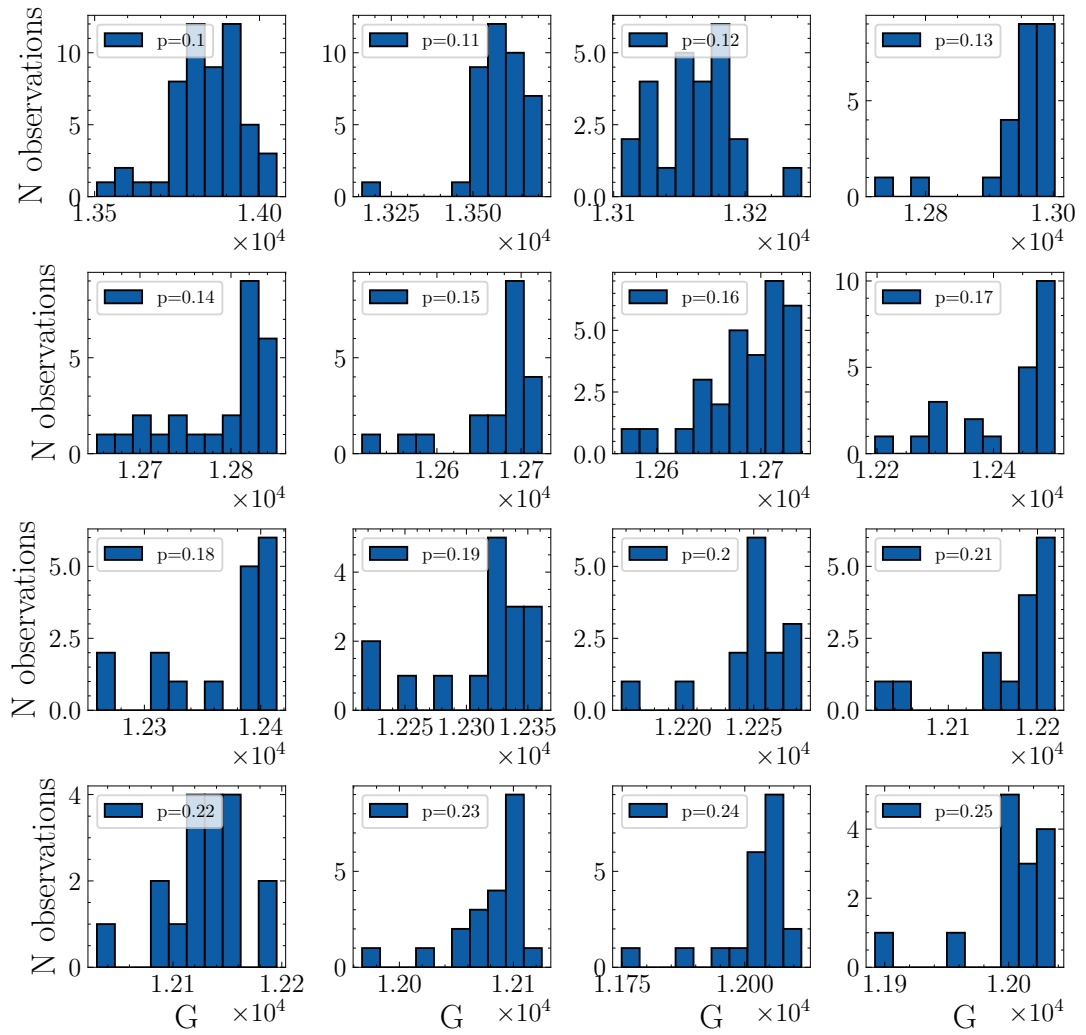


Figure B.3 – This figure the histograms for gaussianity values larger than one for $k = 75.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 100.0$

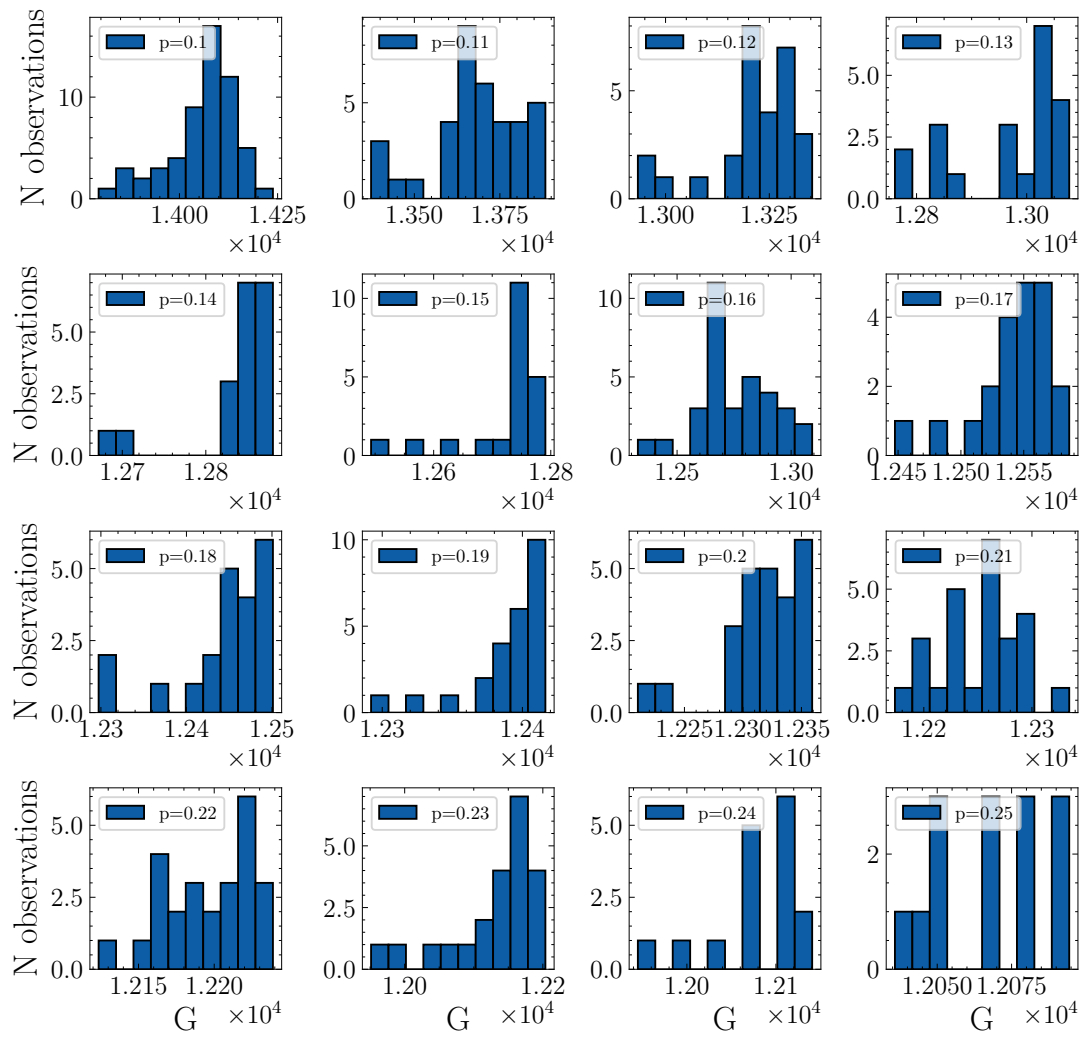


Figure B.4 – This figure the histograms for gaussianity values larger than one for $k = 100.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 150.0$

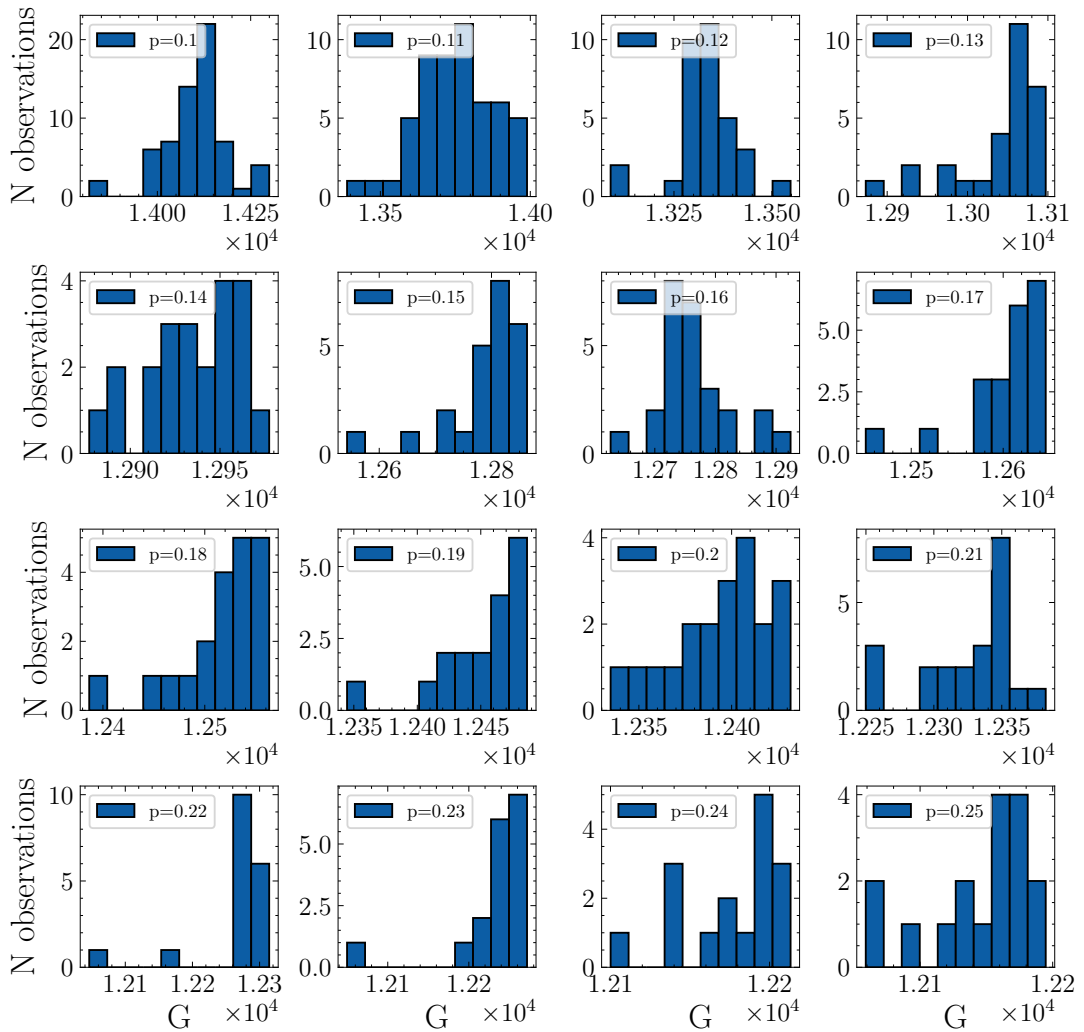


Figure B.5 – This figure the histograms for gaussianity values larger than one for $k = 150.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 200.0$

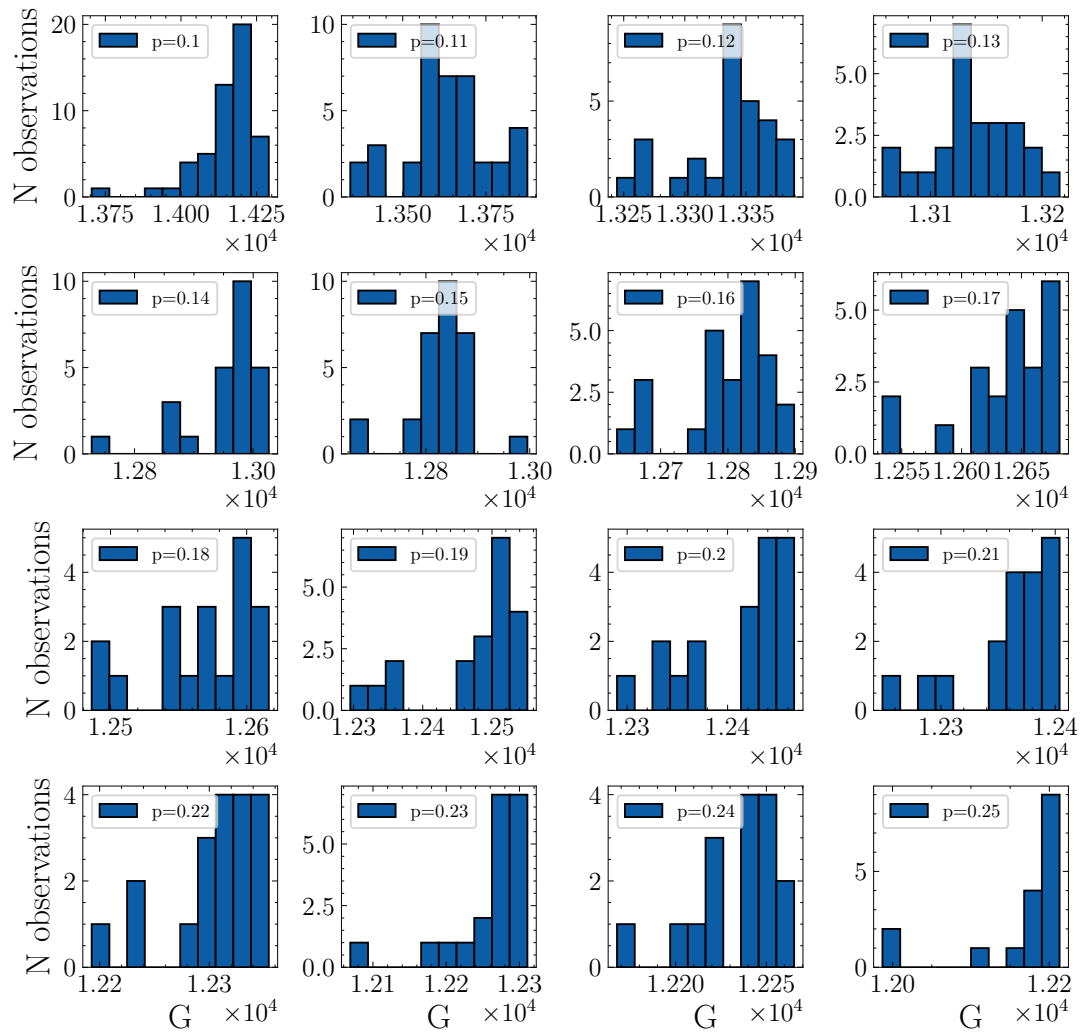


Figure B.6 – This figure the histograms for gaussianity values larger than one for $k = 200.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 250.0$

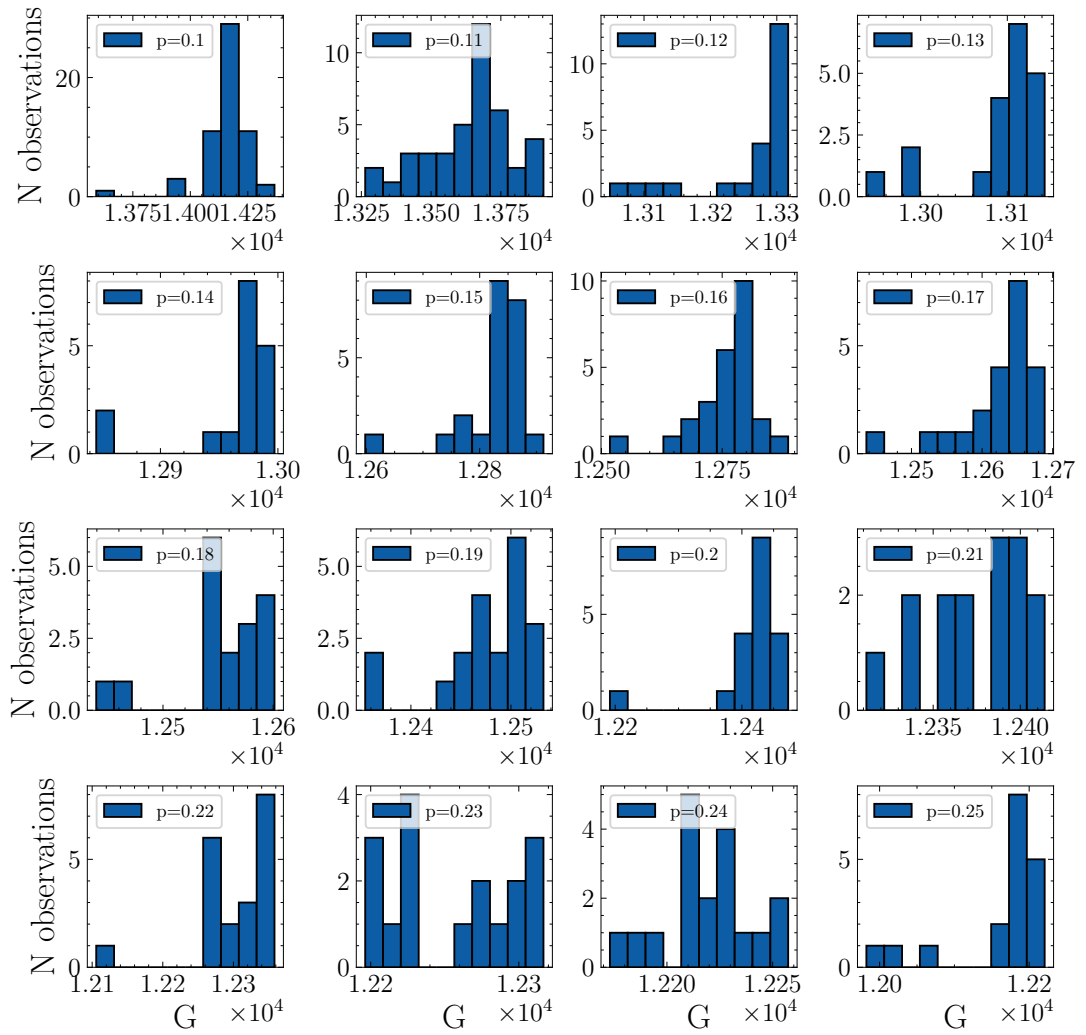


Figure B.7 – This figure the histograms for gaussianity values larger than one for $k = 250.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 300.0$

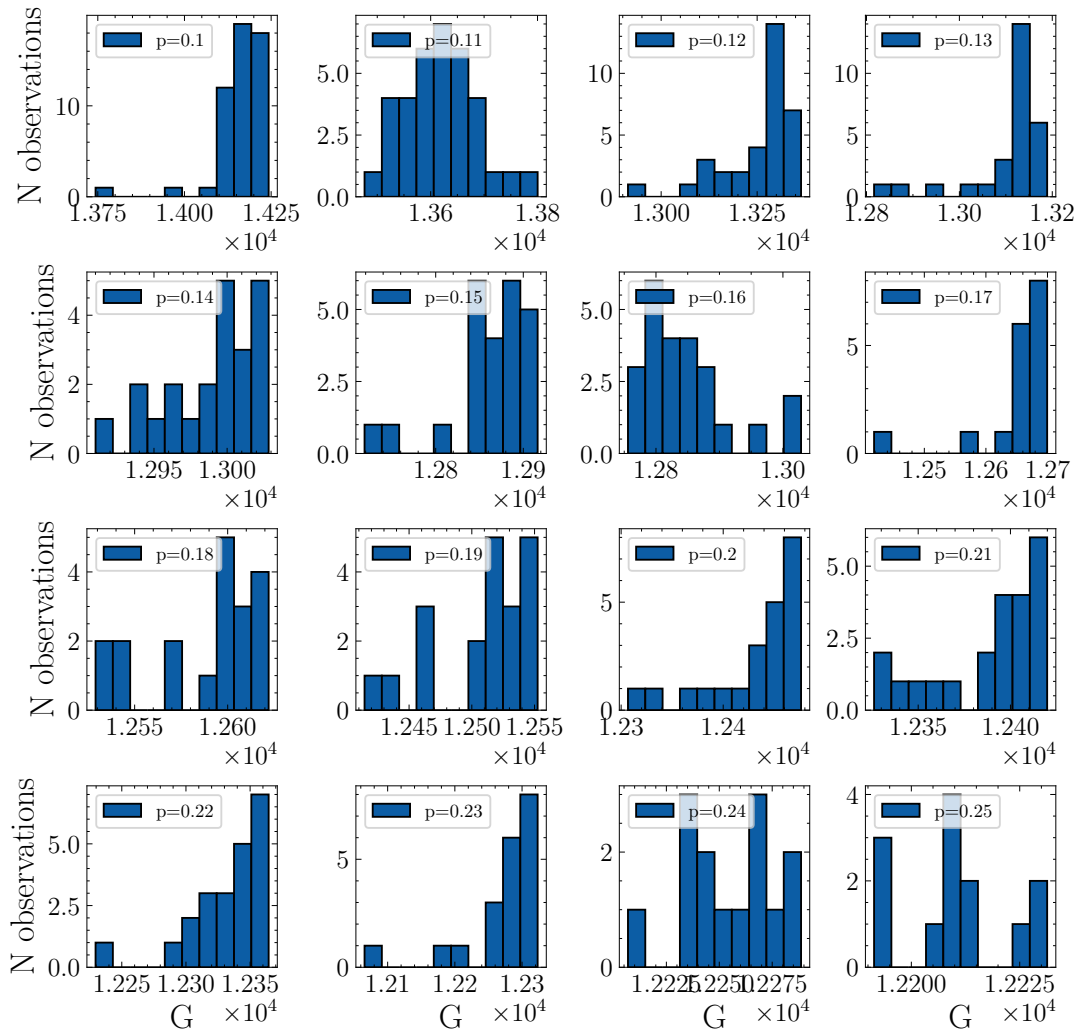


Figure B.8 – This figure the histograms for gaussianity values larger than one for $k = 300.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of gaussianity larger than one for all pressures p for $k = 500.0$

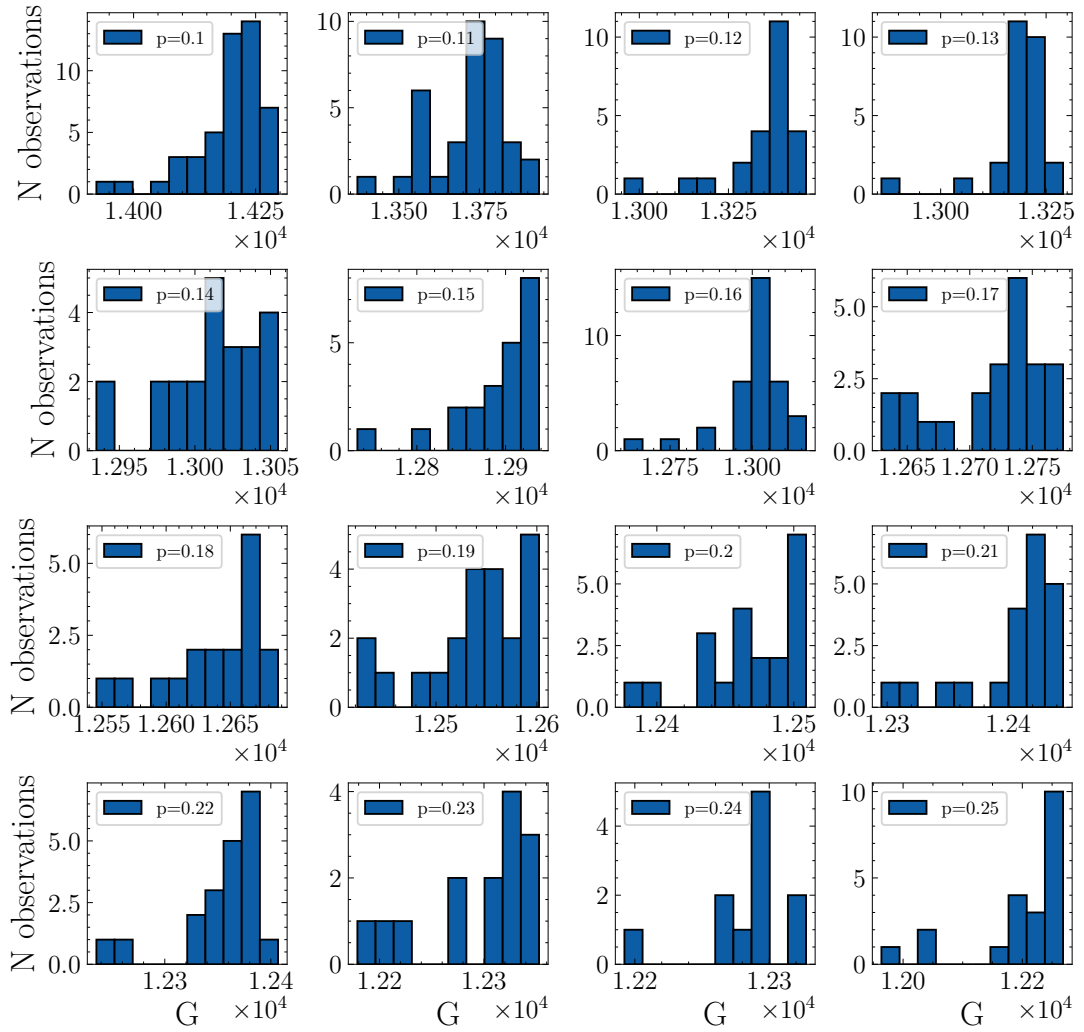


Figure B.9 – This figure the histograms for gaussianity values larger than one for $k = 500.0$ for all pressures p .

Fonte: o autor (2022).

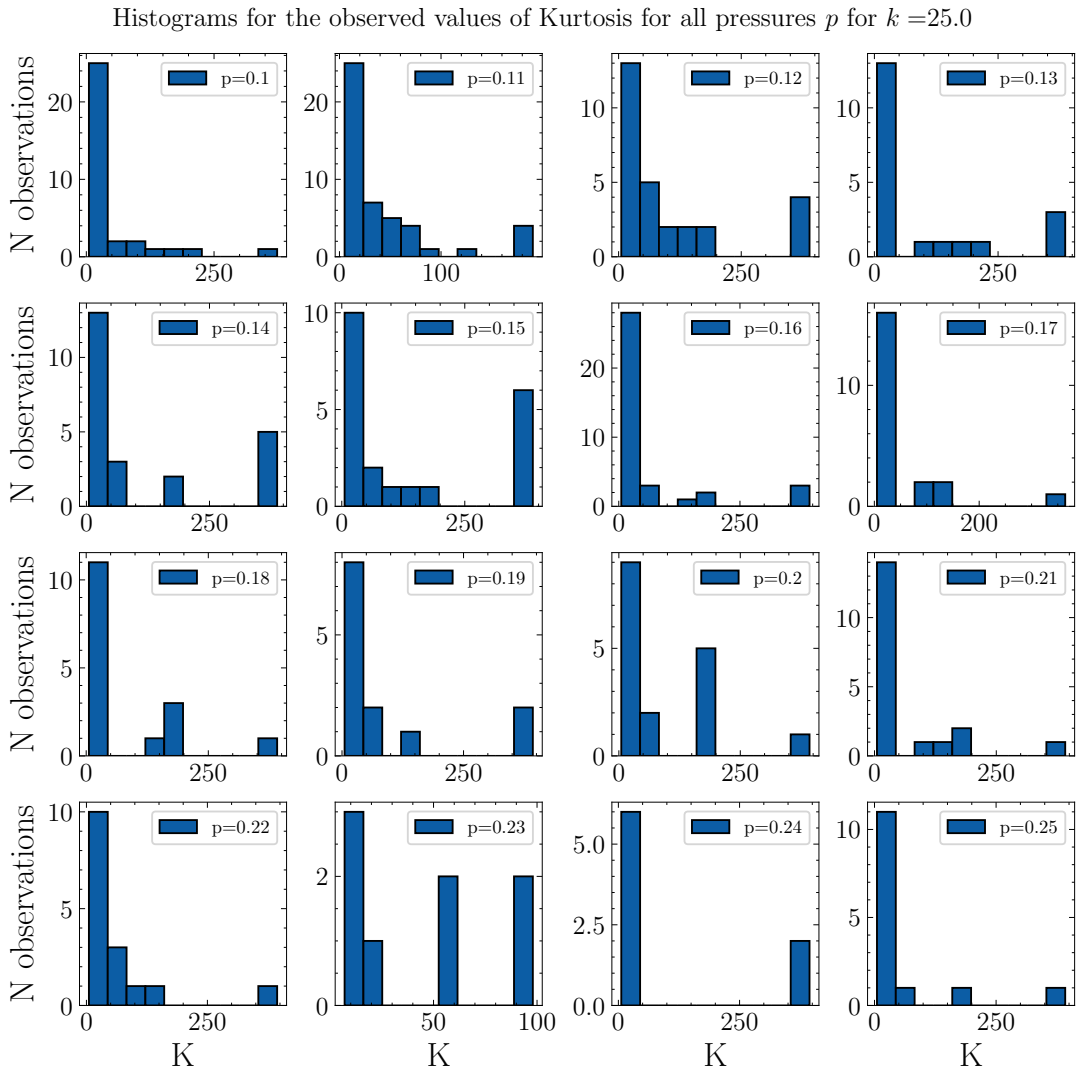


Figure B.10 – This figure the histograms for kurtosis values larger than 5 for $k = 25.0$ for all pressures p .

Fonte: o autor (2022).

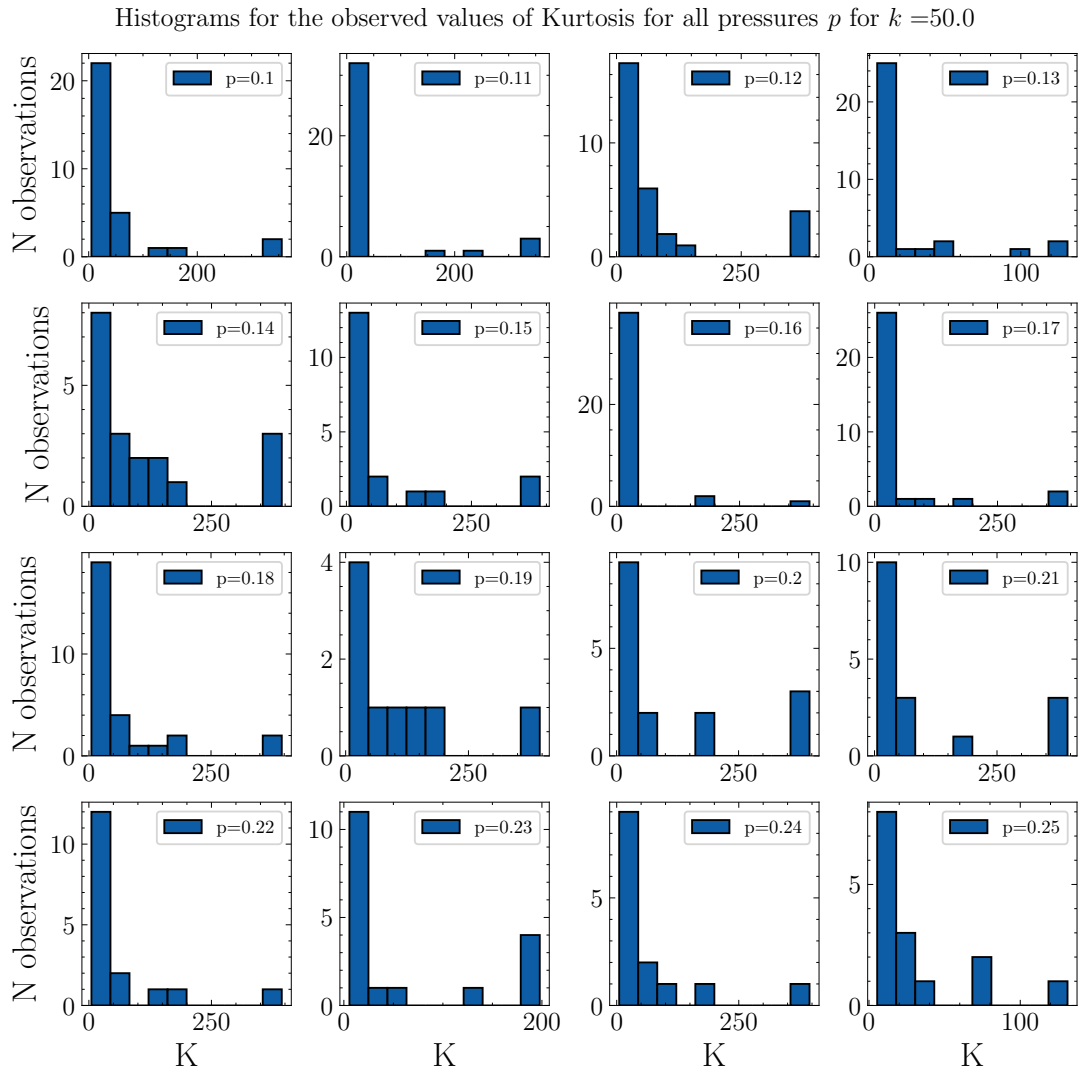


Figure B.11 – This figure the histograms for kurtosis values larger than 5 for $k = 50.0$ for all pressures p .

Fonte: o autor (2022).

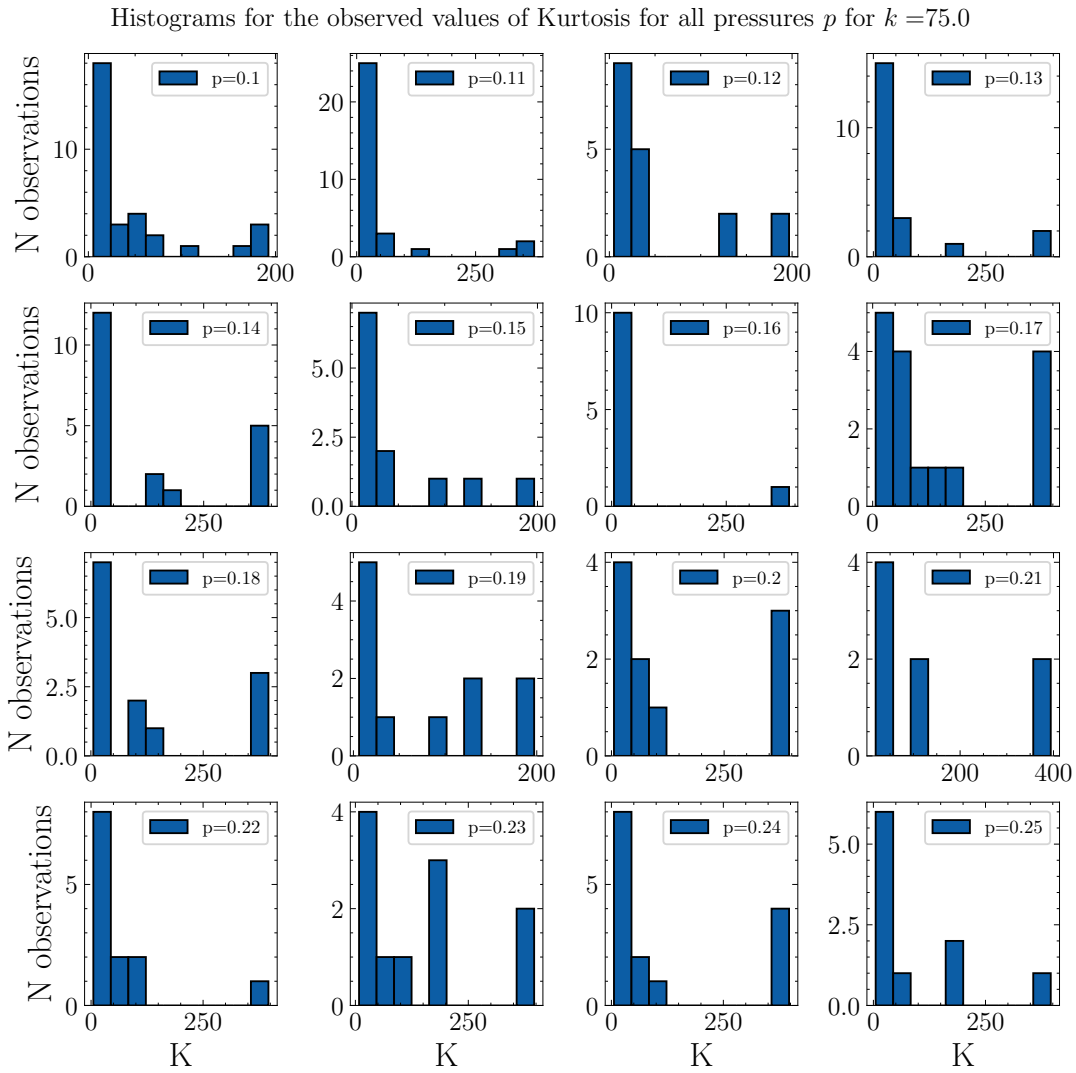


Figure B.12 – This figure the histograms for kurtosis values larger than 5 for $k = 75.0$ for all pressures p .

Fonte: o autor (2022).

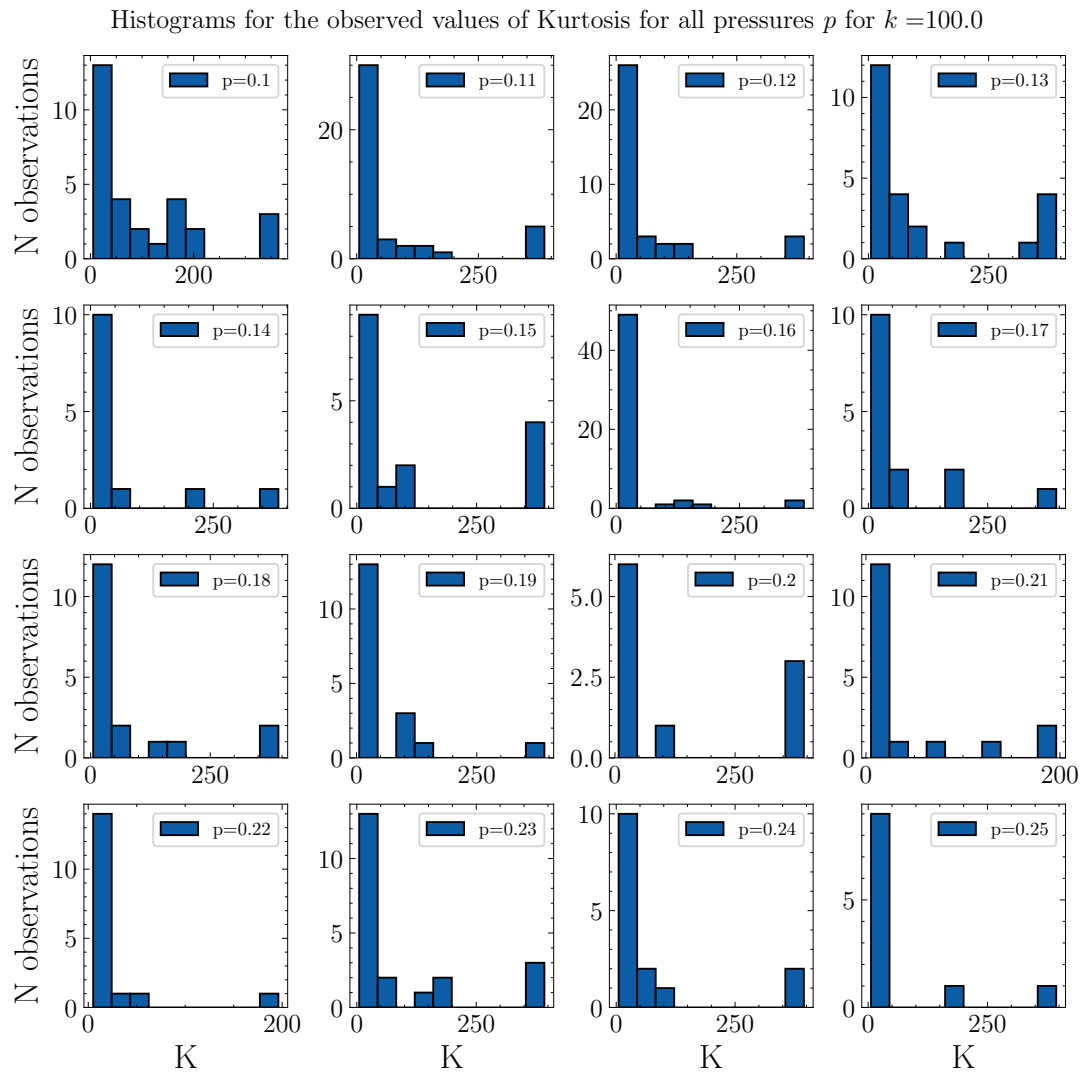


Figure B.13 – This figure the histograms for kurtosis values larger than 5 for $k = 100.0$ for all pressures p .

Fonte: o autor (2022).

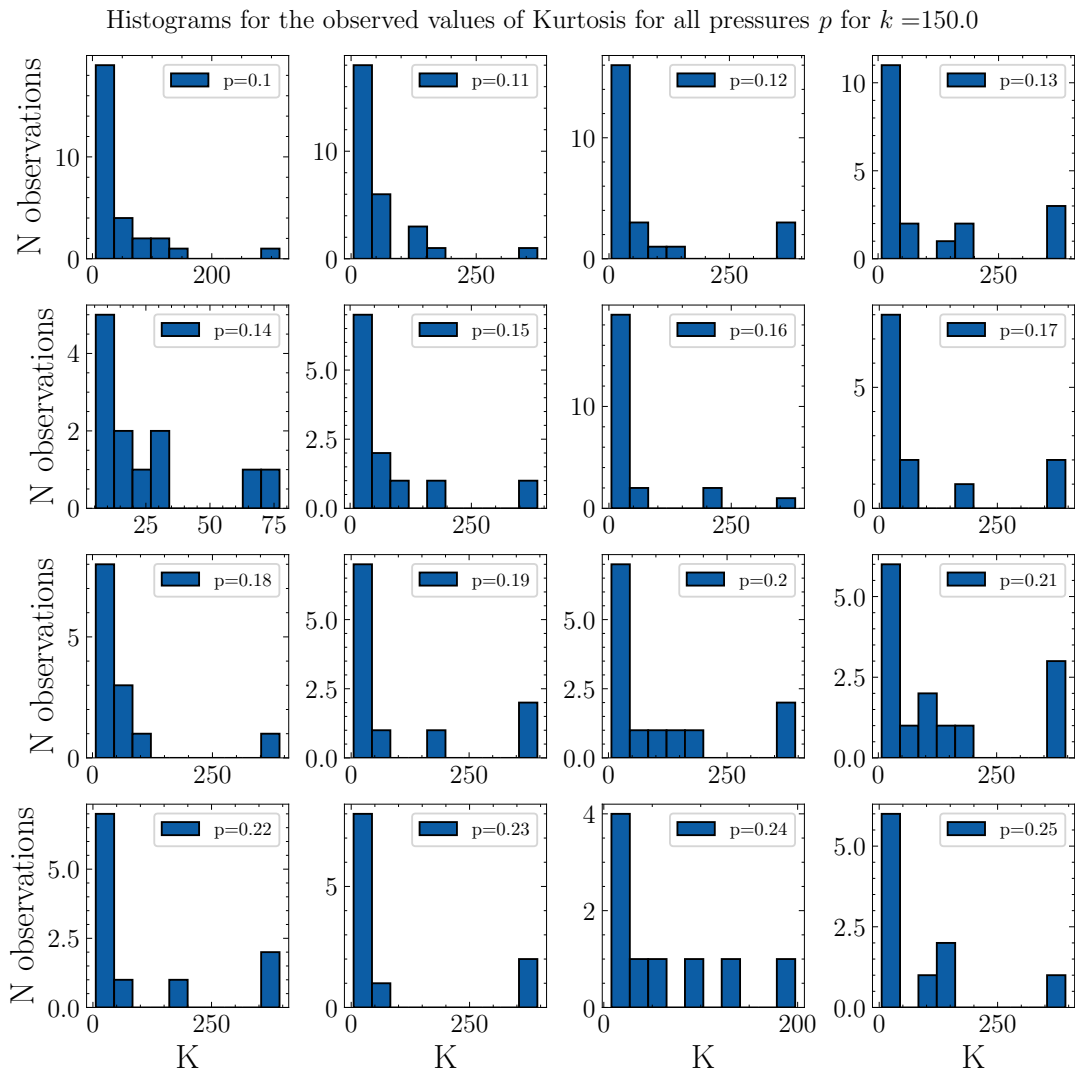


Figure B.14 – This figure the histograms for kurtosis values larger than 5 for $k = 150.0$ for all pressures p .

Fonte: o autor (2022).

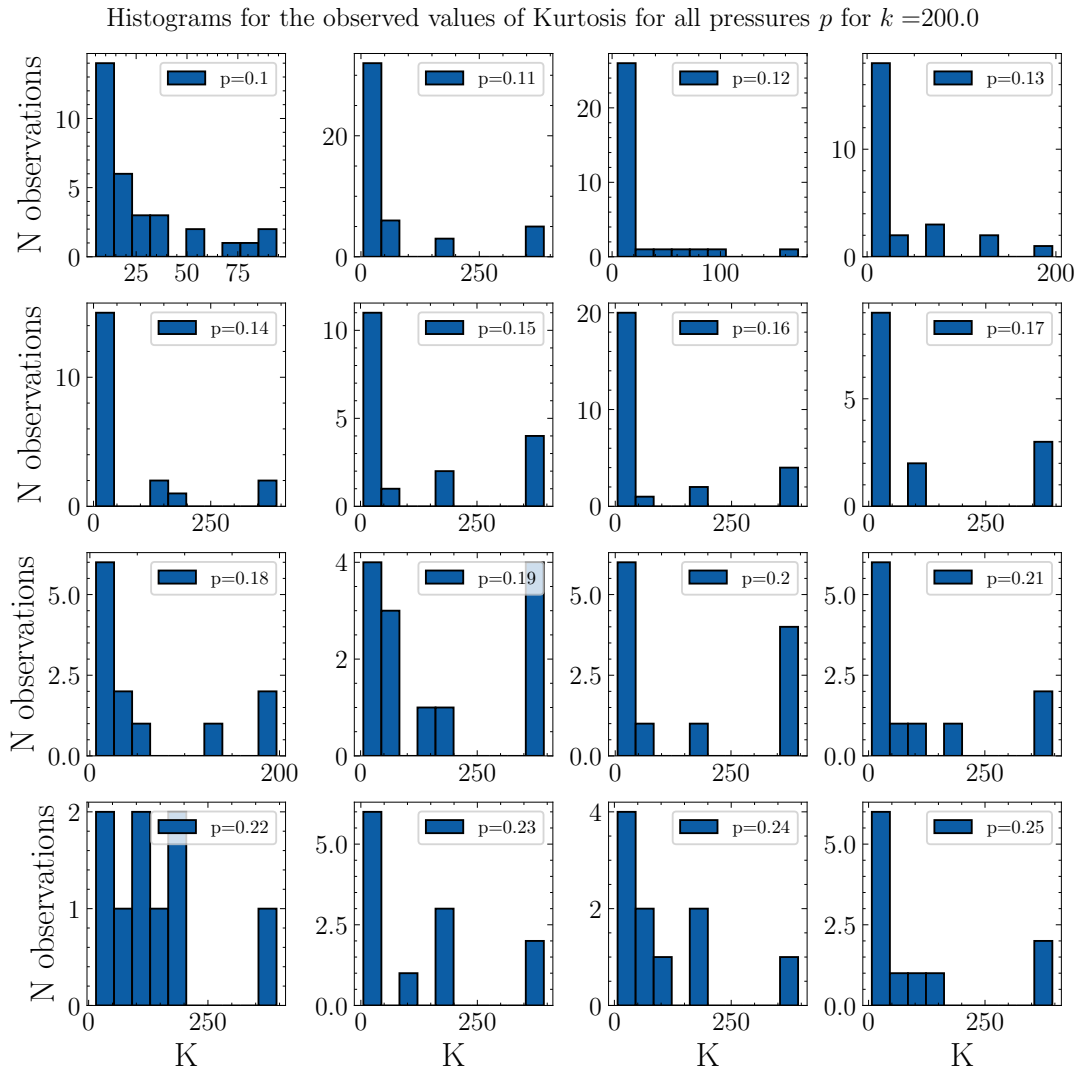


Figure B.15 – This figure the histograms for kurtosis values larger than 5 for $k = 200.0$ for all pressures p .

Fonte: o autor (2022).

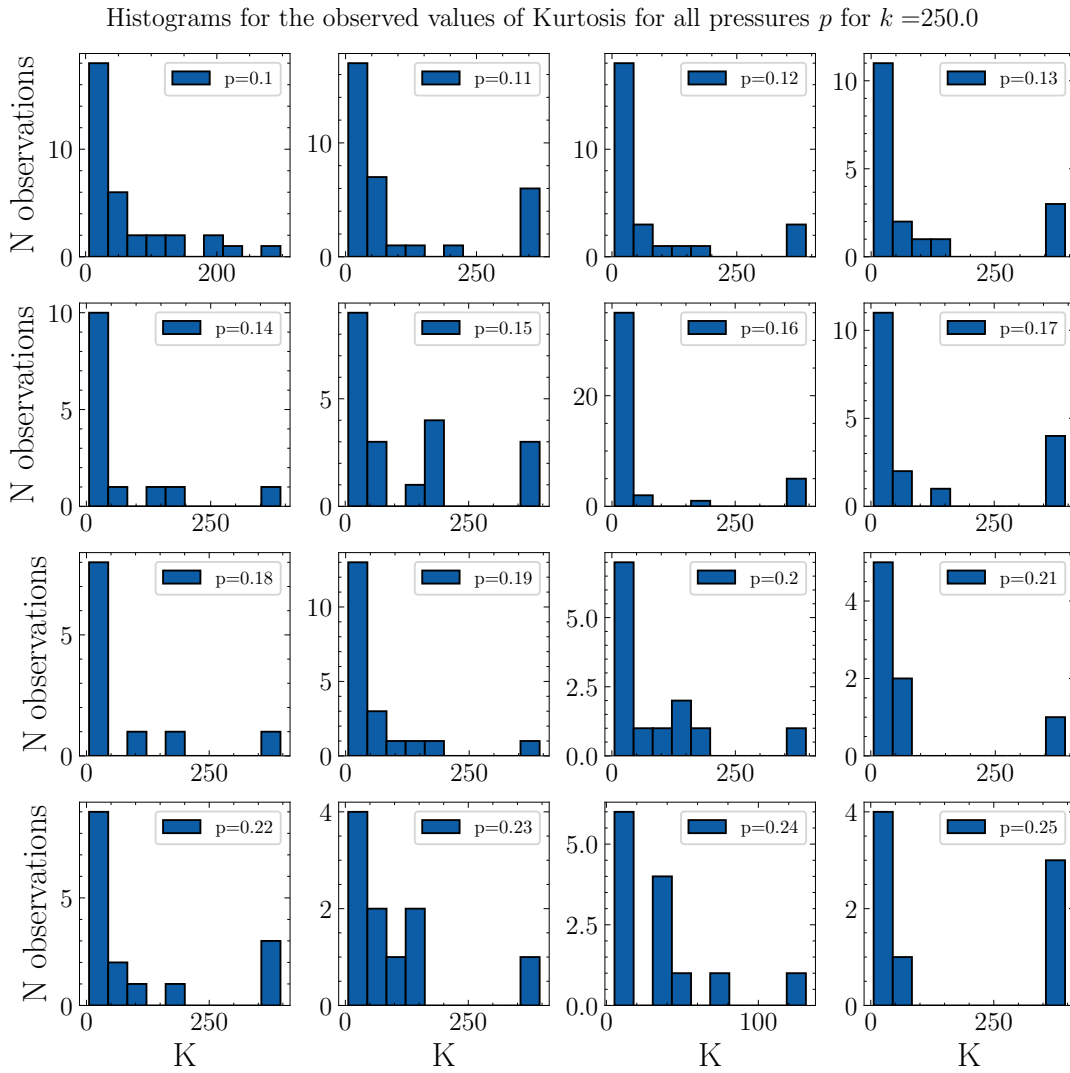


Figure B.16 – This figure the histograms for kurtosis values larger than 5 for $k = 250.0$ for all pressures p .

Fonte: o autor (2022).

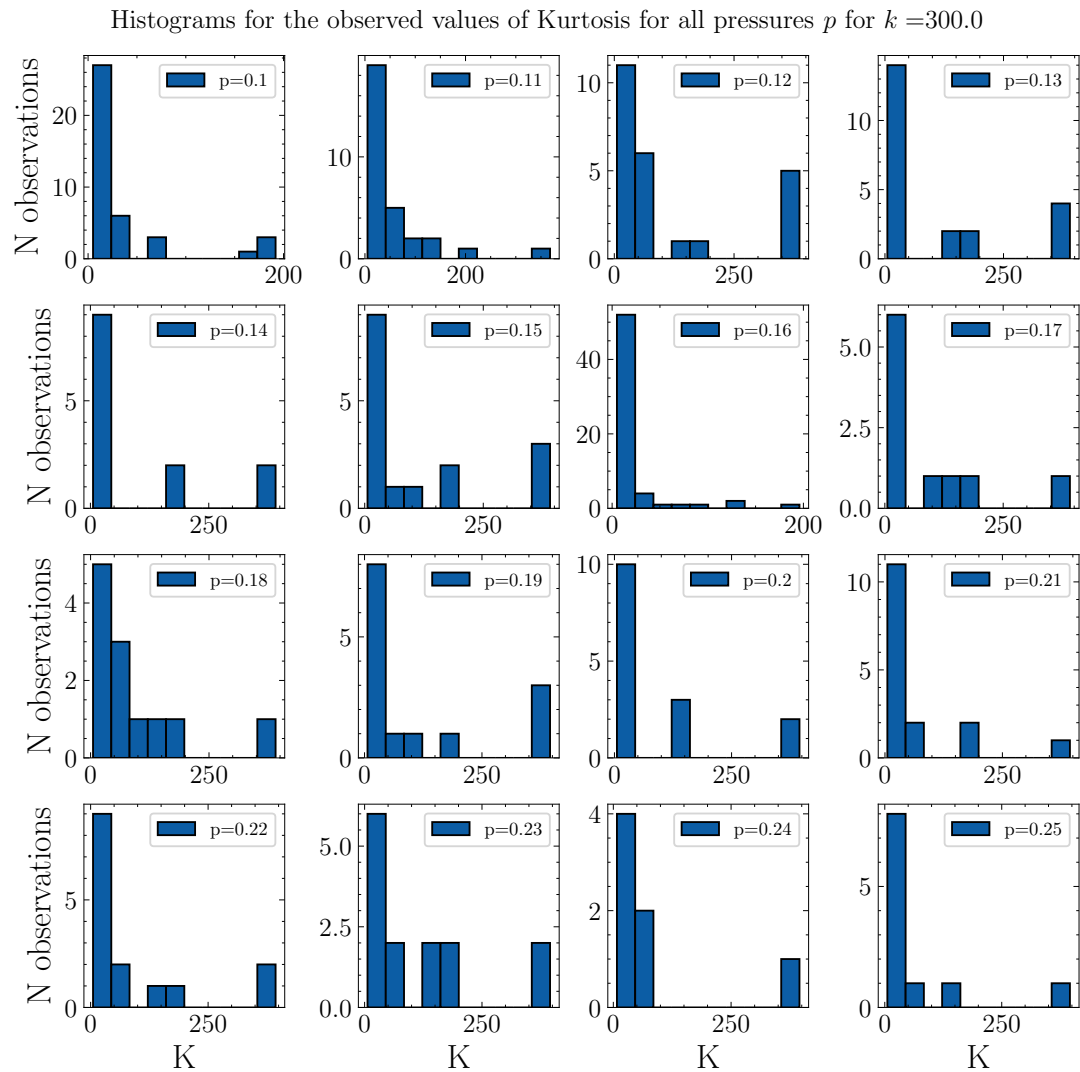


Figure B.17 – This figure the histograms for kurtosis values larger than 5 for $k = 300.0$ for all pressures p .

Fonte: o autor (2022).

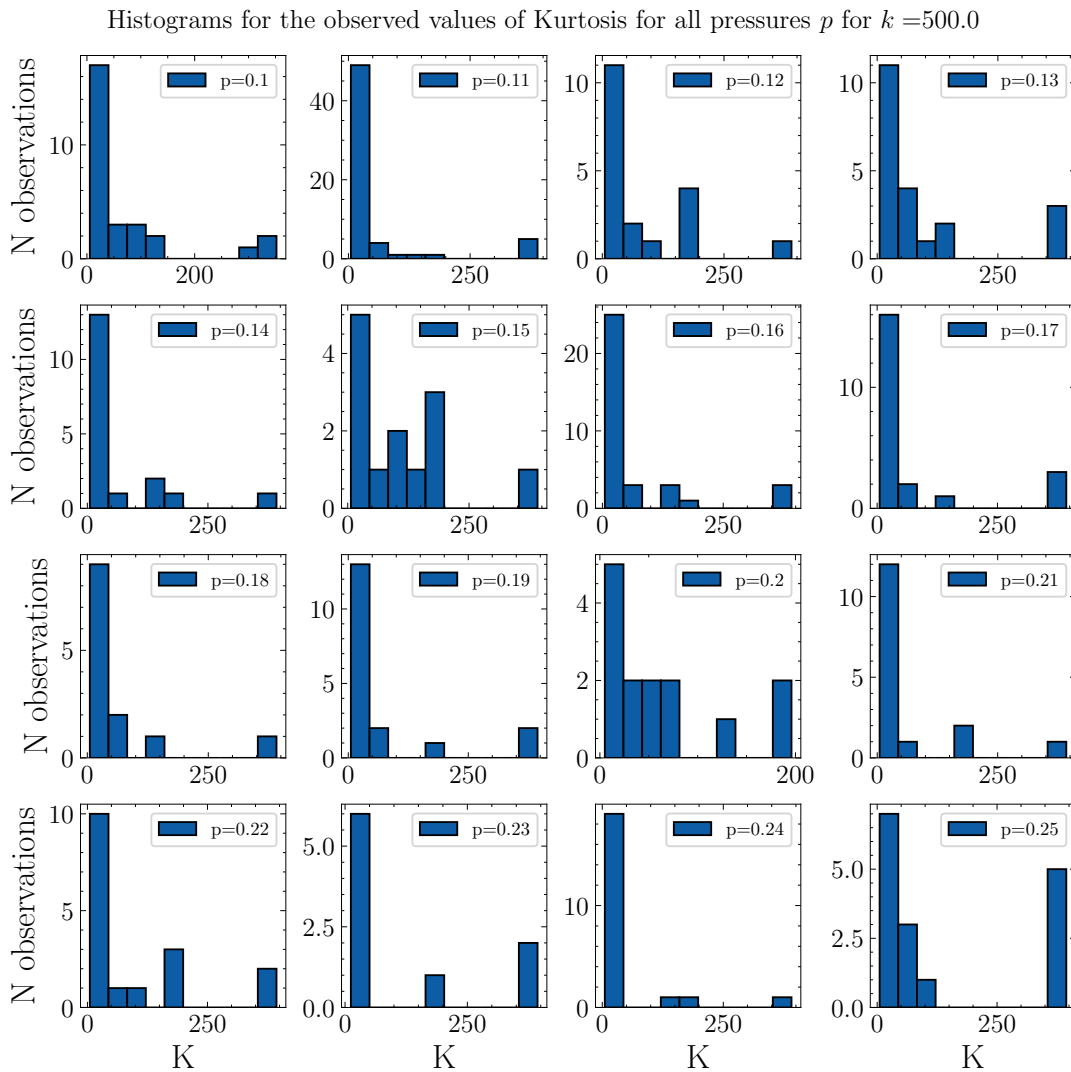


Figure B.18 – This figure the histograms for kurtosis values larger than 5 for $k = 500.0$ for all pressures p .

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 25.0$

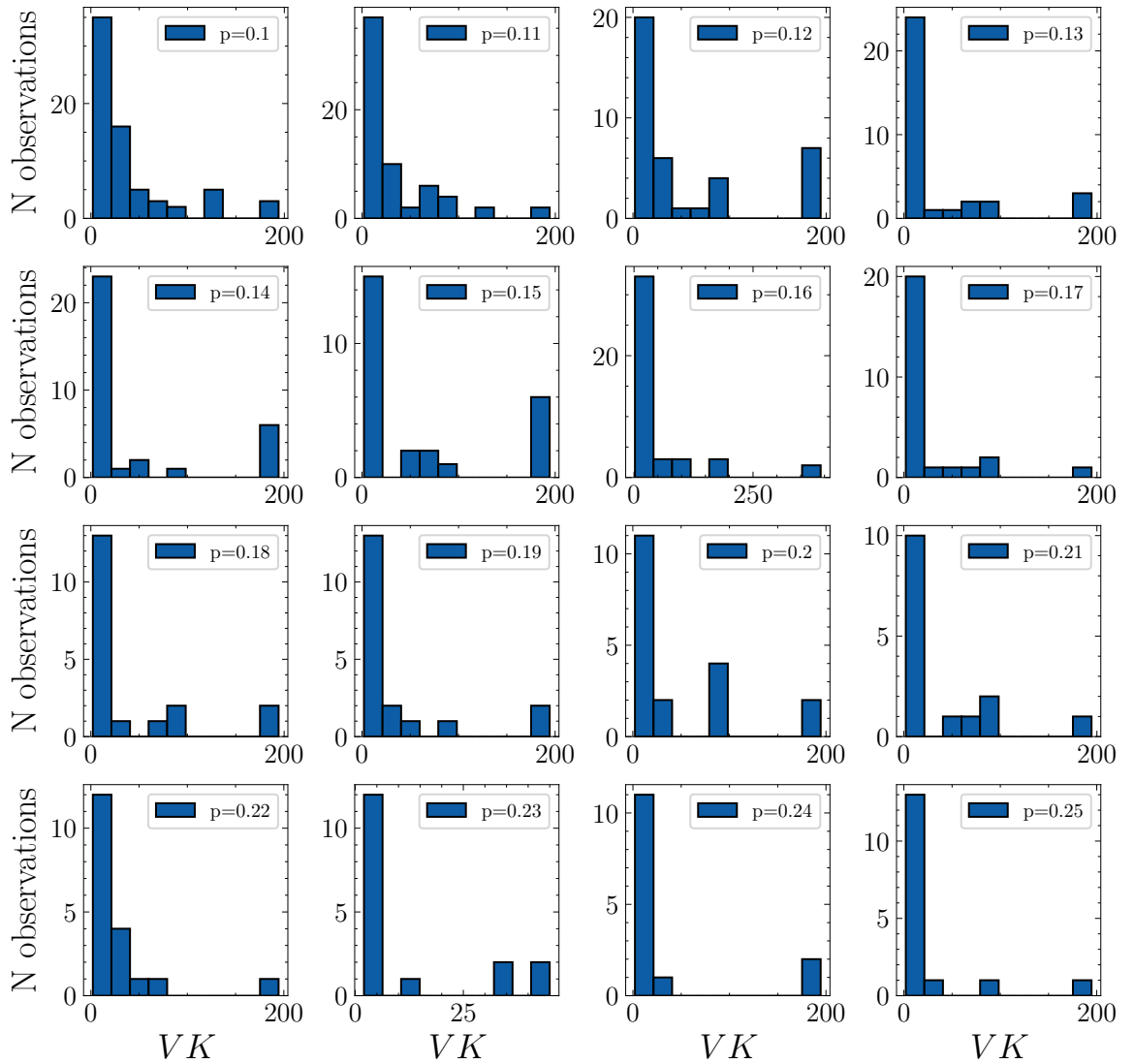


Figure B.19 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 25.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 50.0$

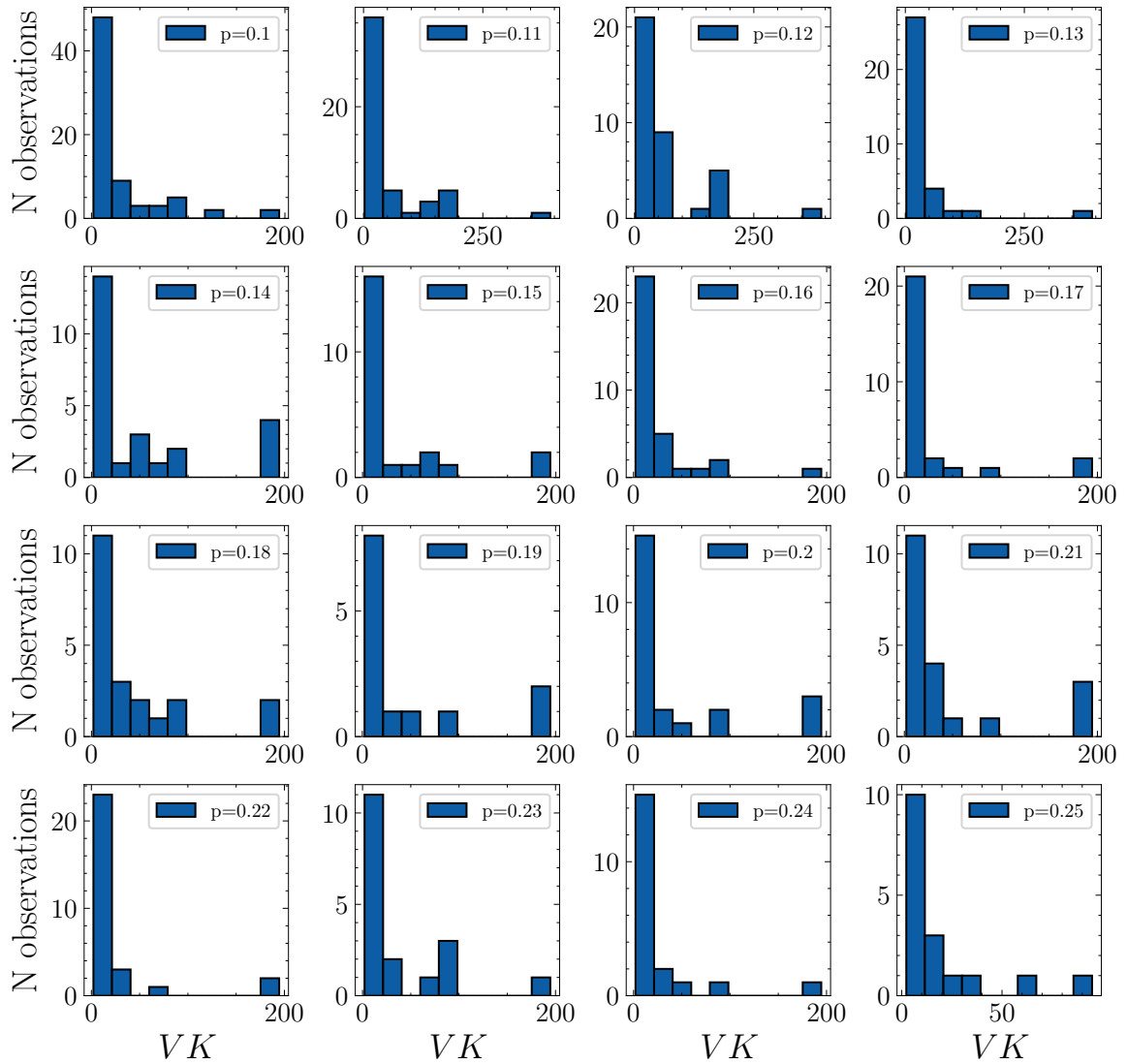


Figure B.20 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 50.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 75.0$

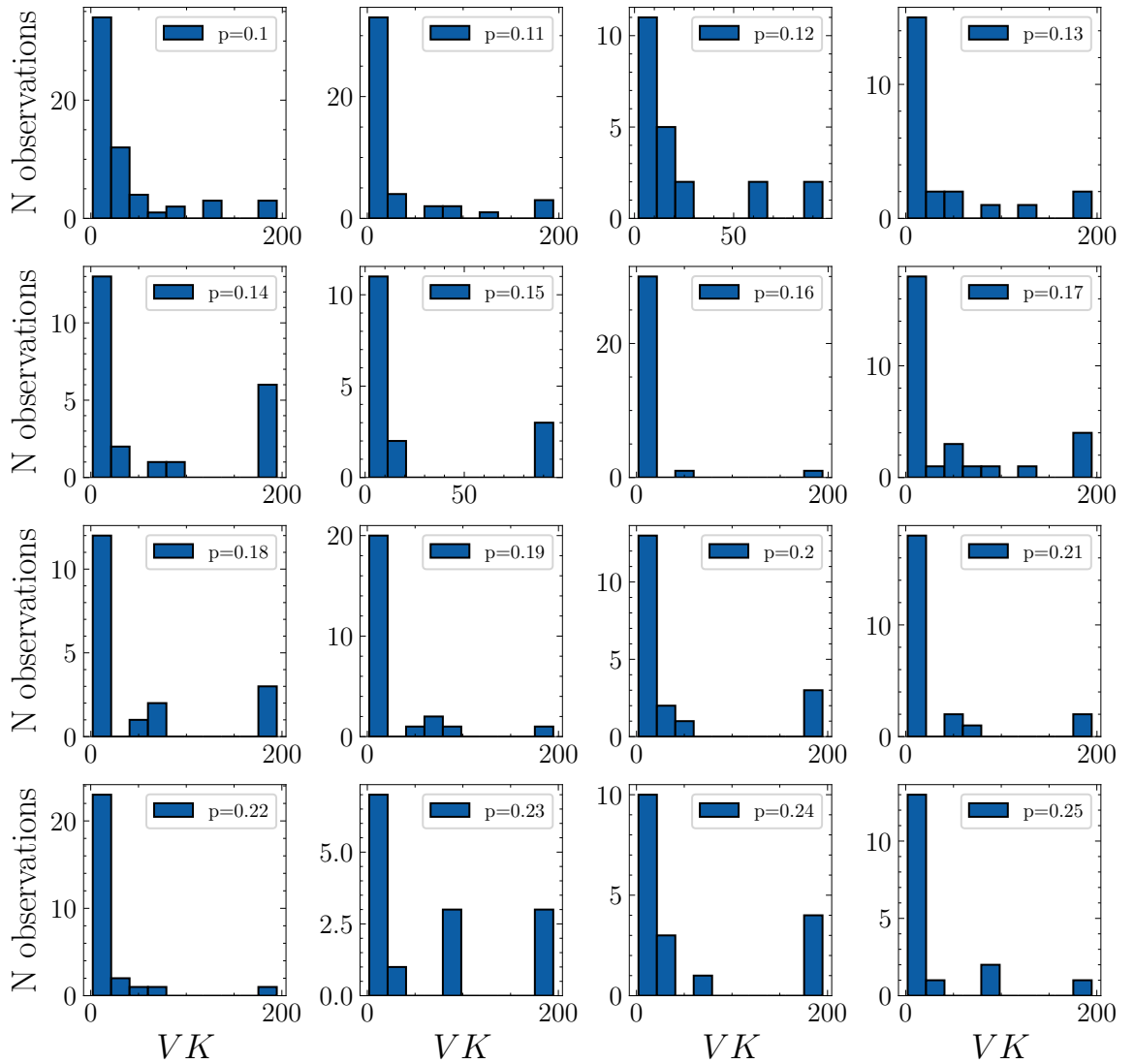


Figure B.21 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 75.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 100.0$

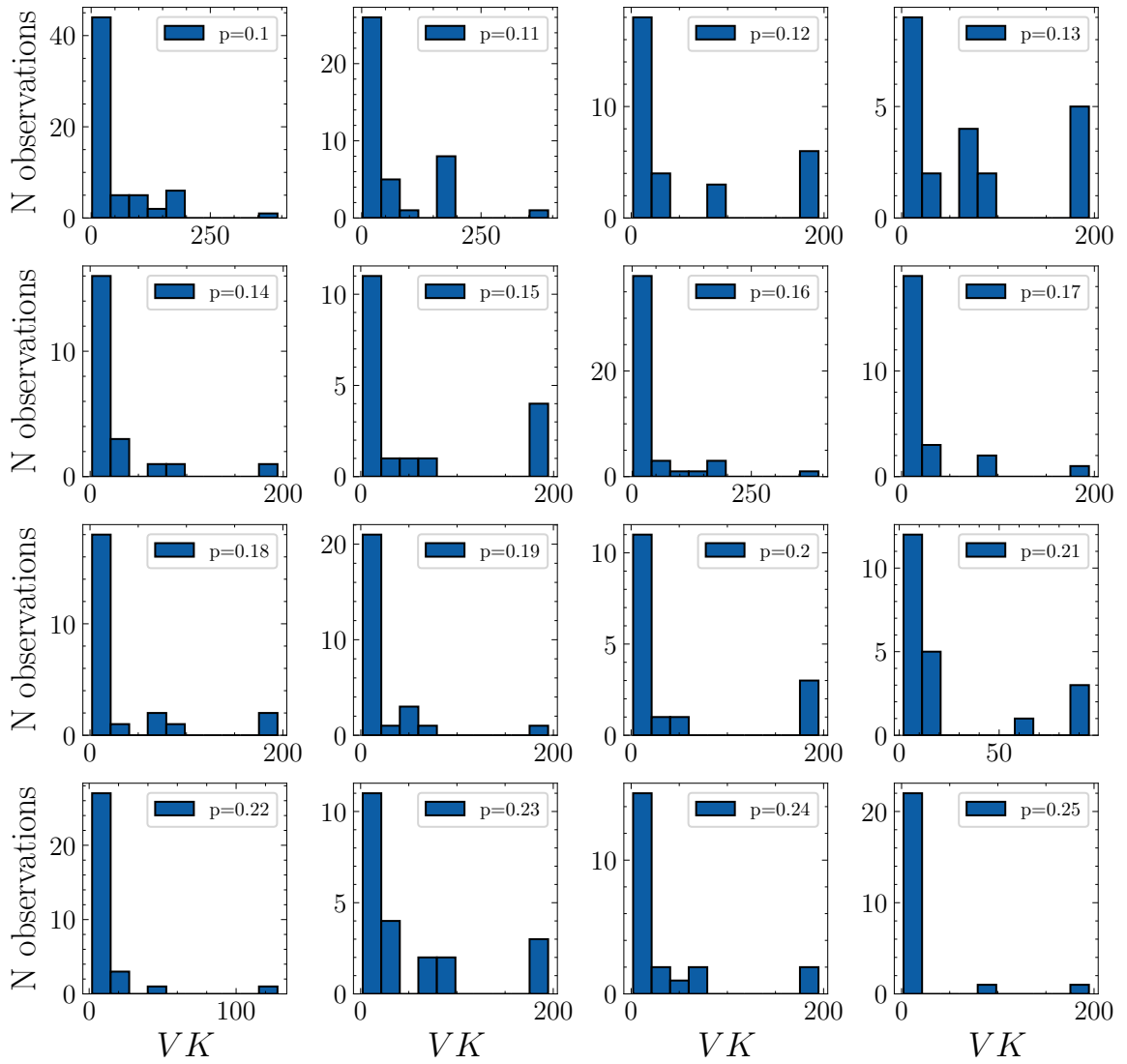


Figure B.22 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 100.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 150.0$

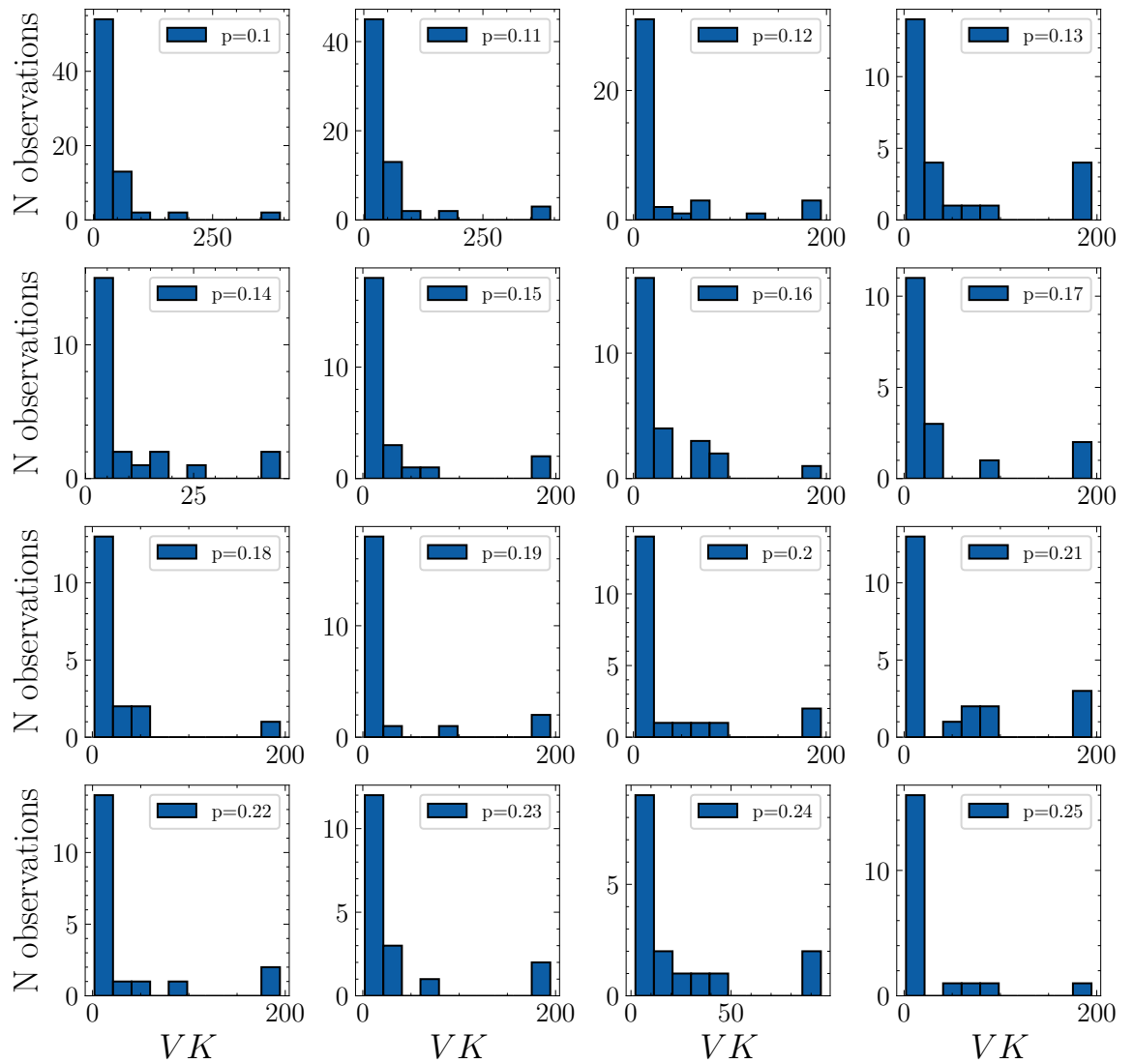


Figure B.23 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 150.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 200.0$

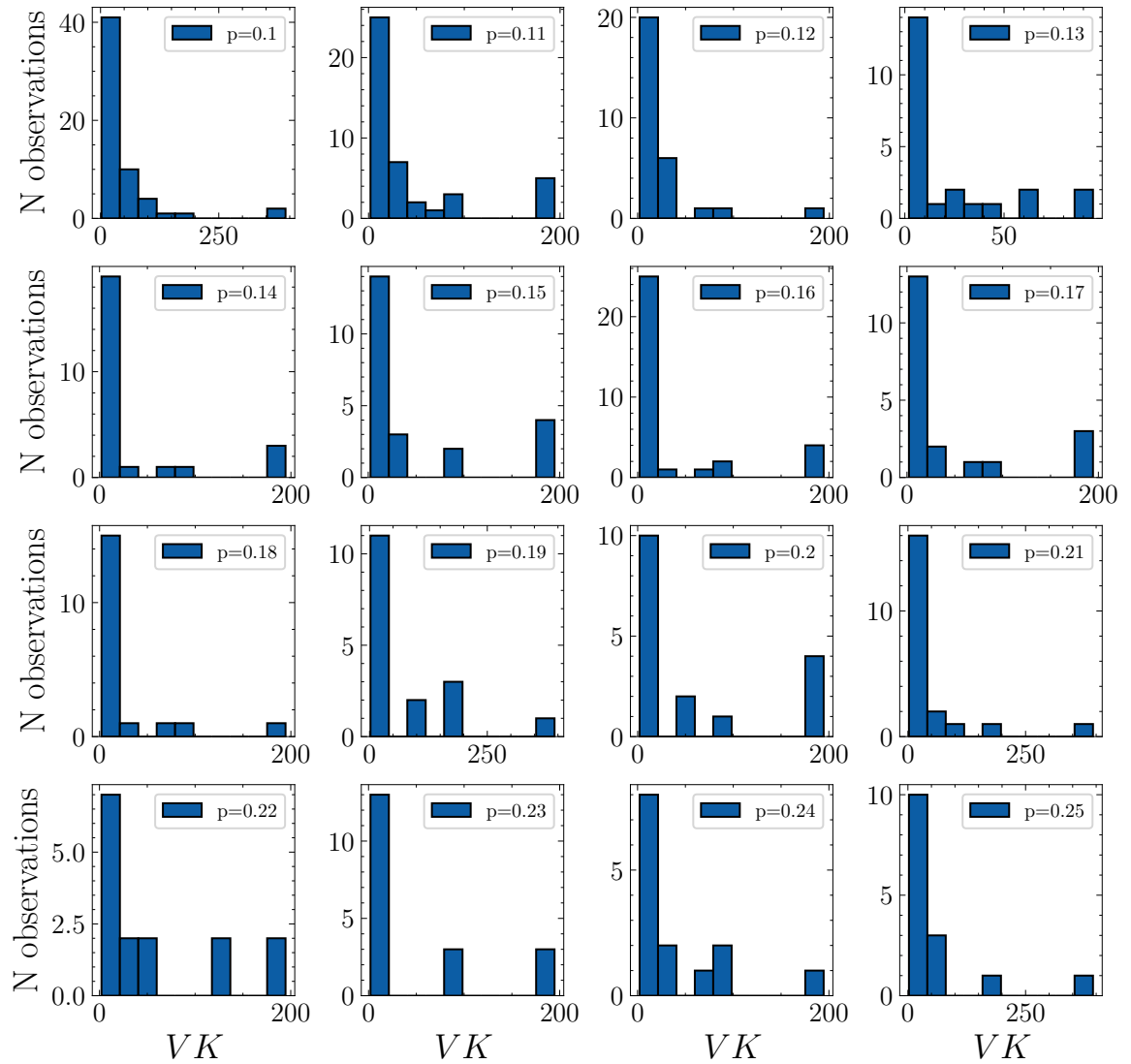


Figure B.24 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 200.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 250.0$

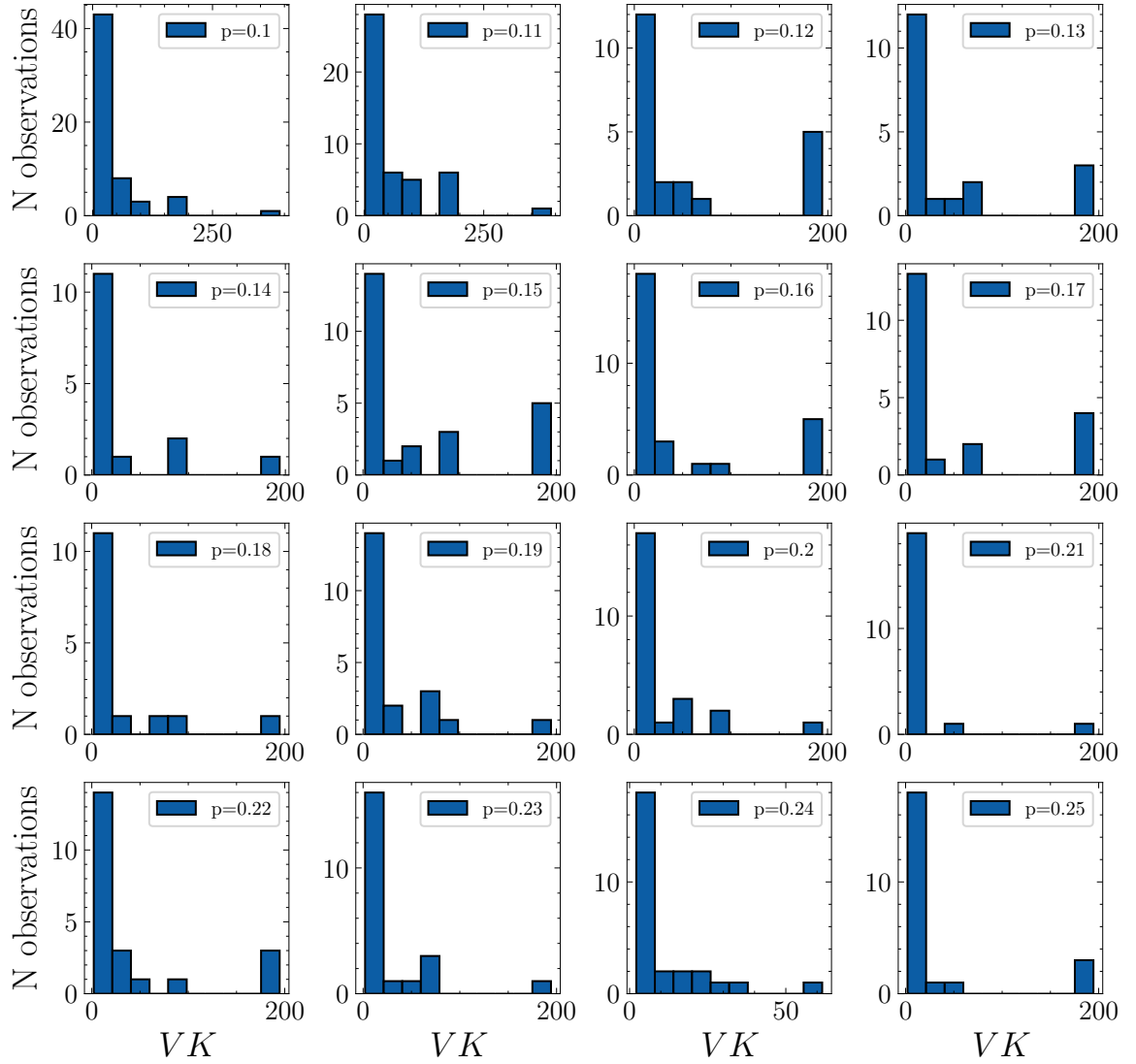


Figure B.25 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 250.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 300.0$

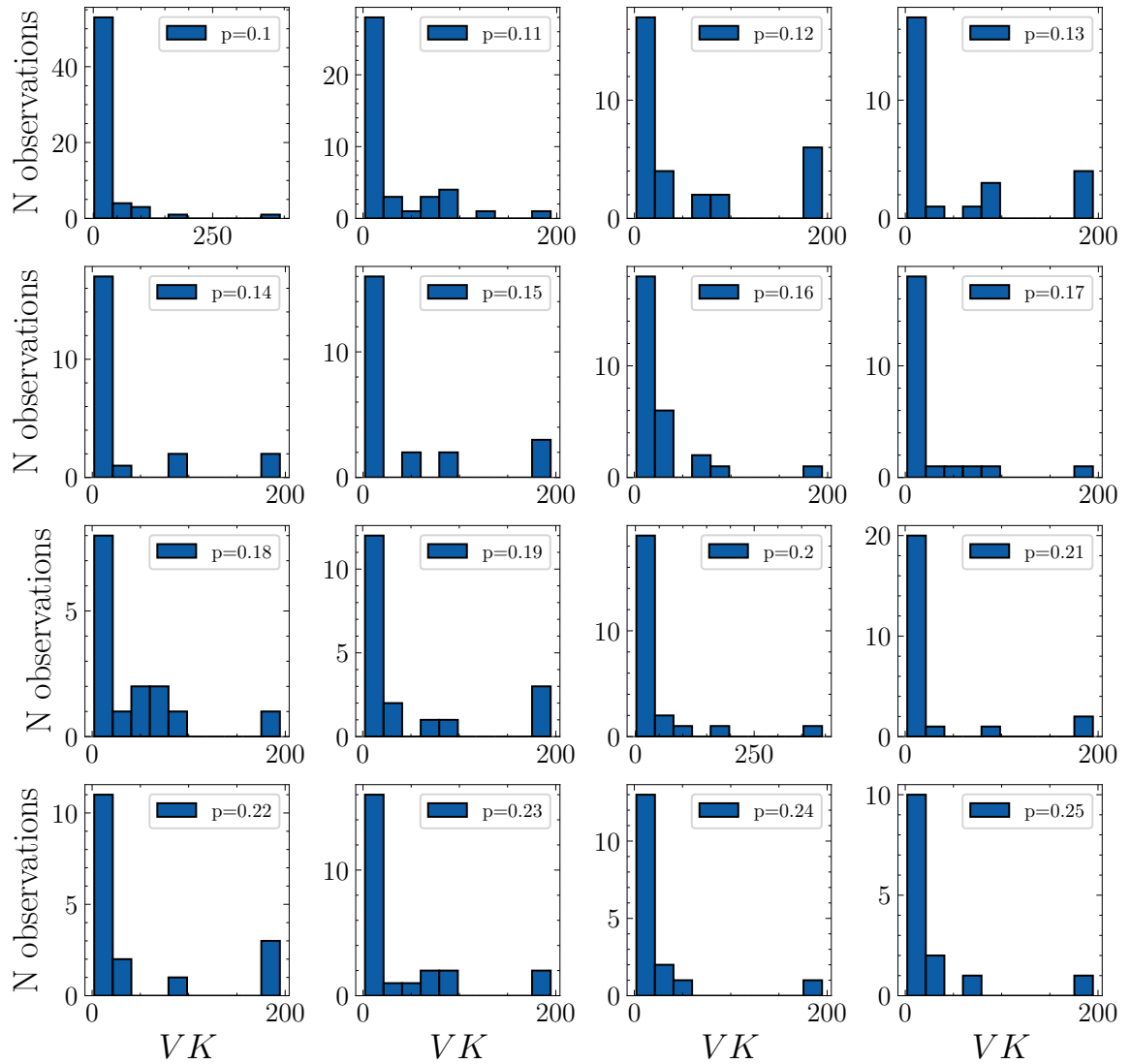


Figure B.26 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 300.0$.

Fonte: o autor (2022).

Histograms for the observed values of Velocity Kurtosis larger than one for all pressures p for $k = 500.0$

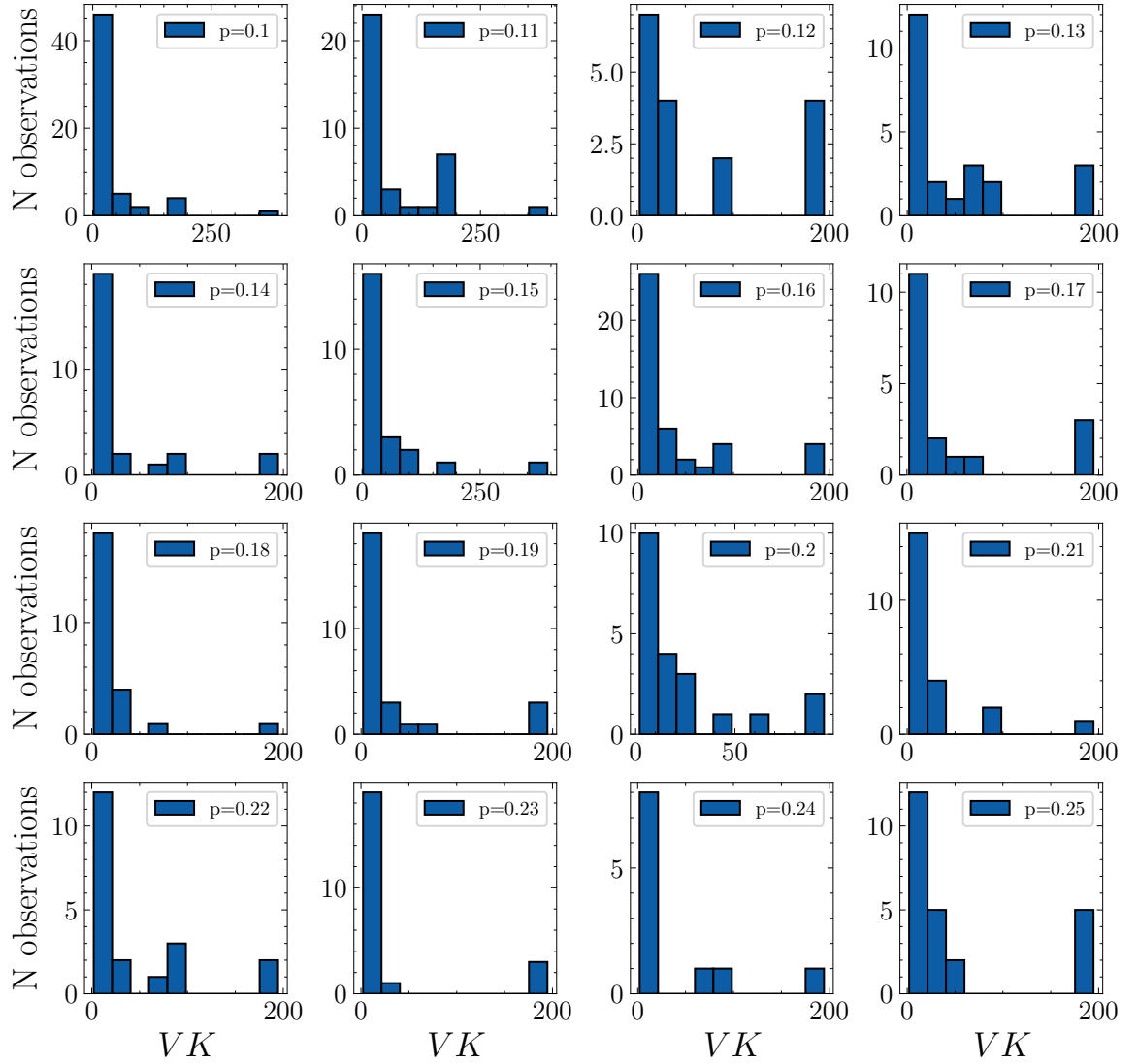


Figure B.27 – This figure provides the histograms for the observed values of velocity kurtosis larger than 2 for every value of p for $k = 500.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 25.0$

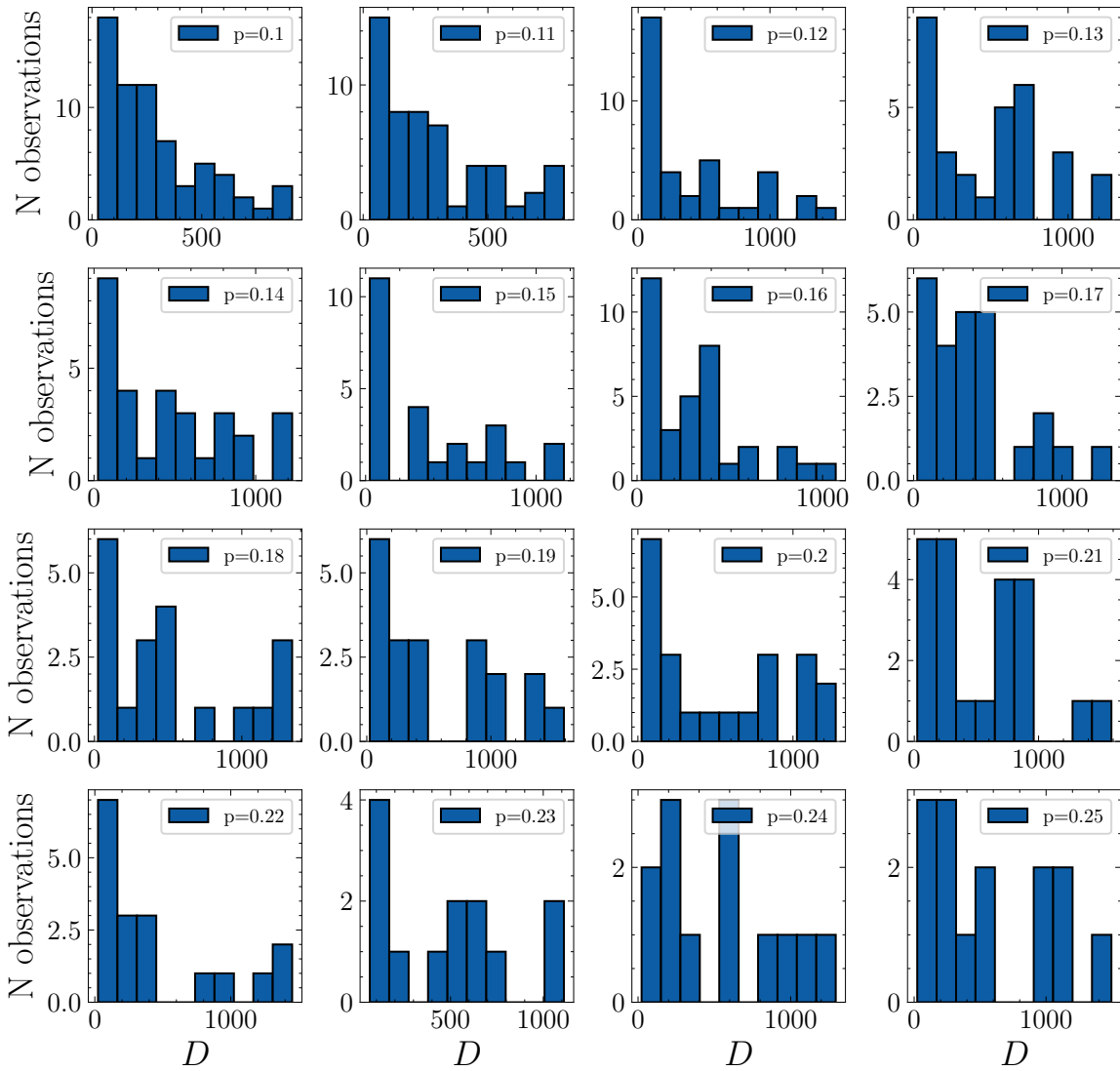


Figure B.28 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 25.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 50.0$

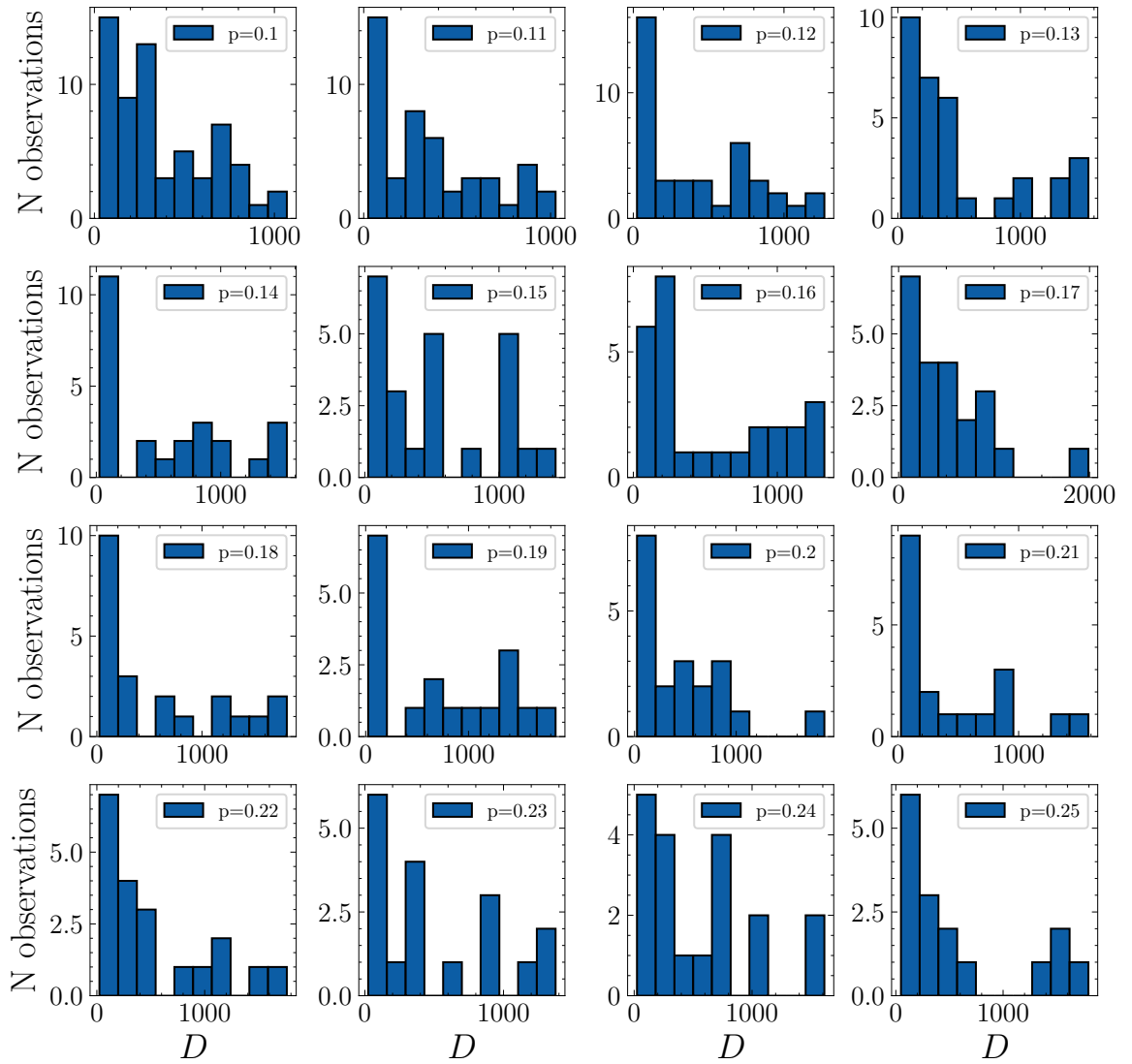


Figure B.29 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 50.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 75.0$

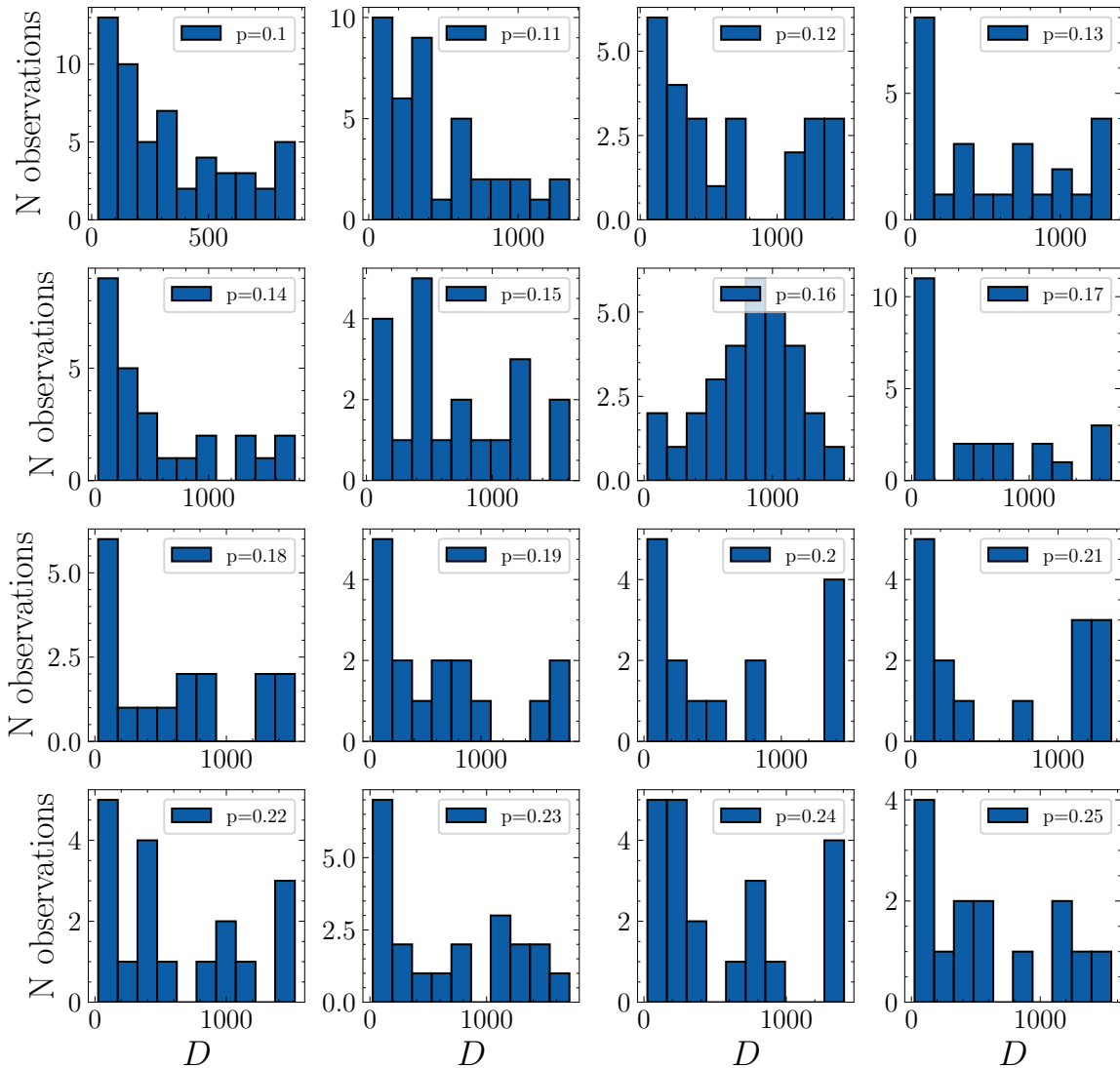


Figure B.30 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 75.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 100.0$

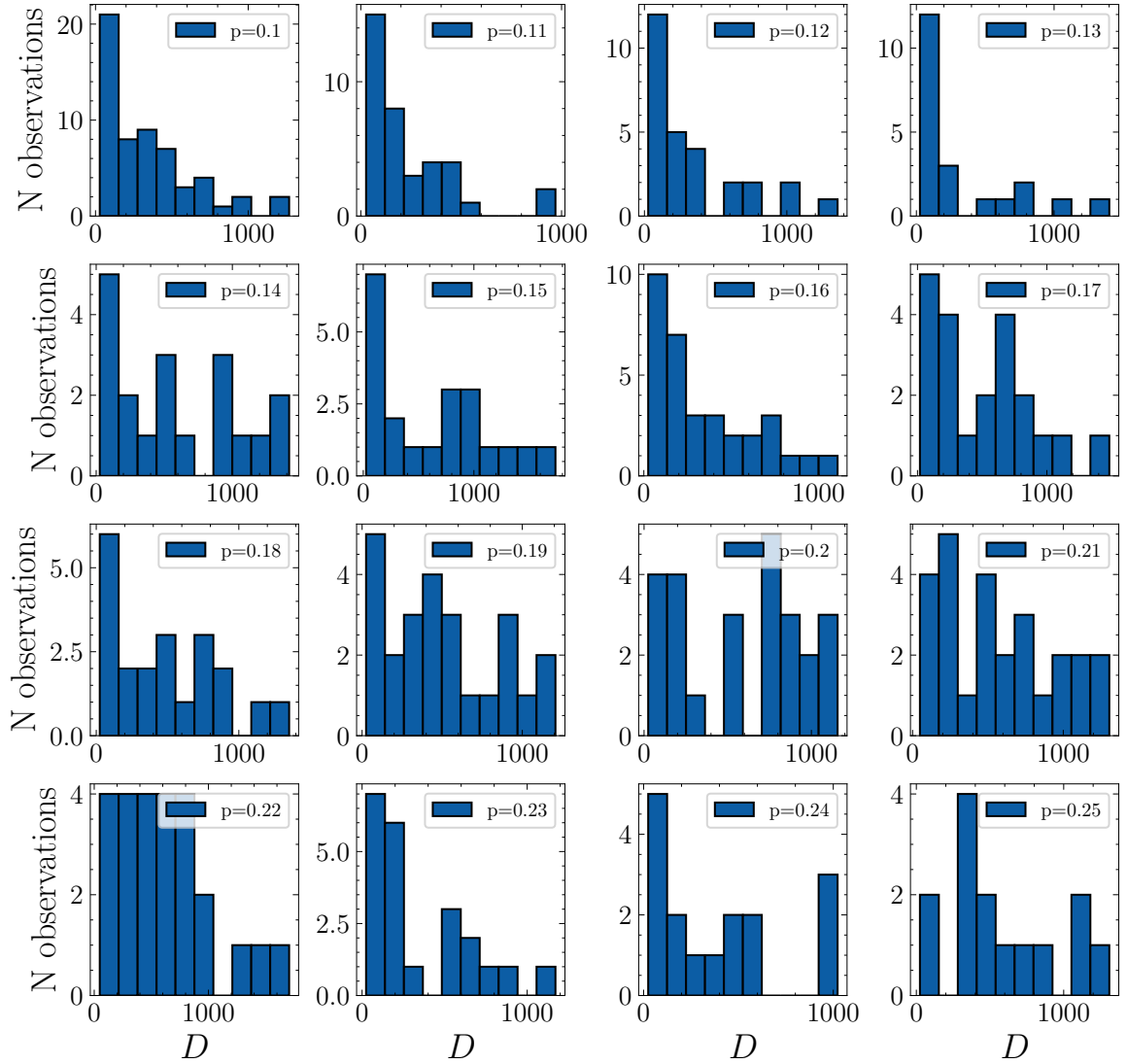


Figure B.31 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 100.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 150.0$

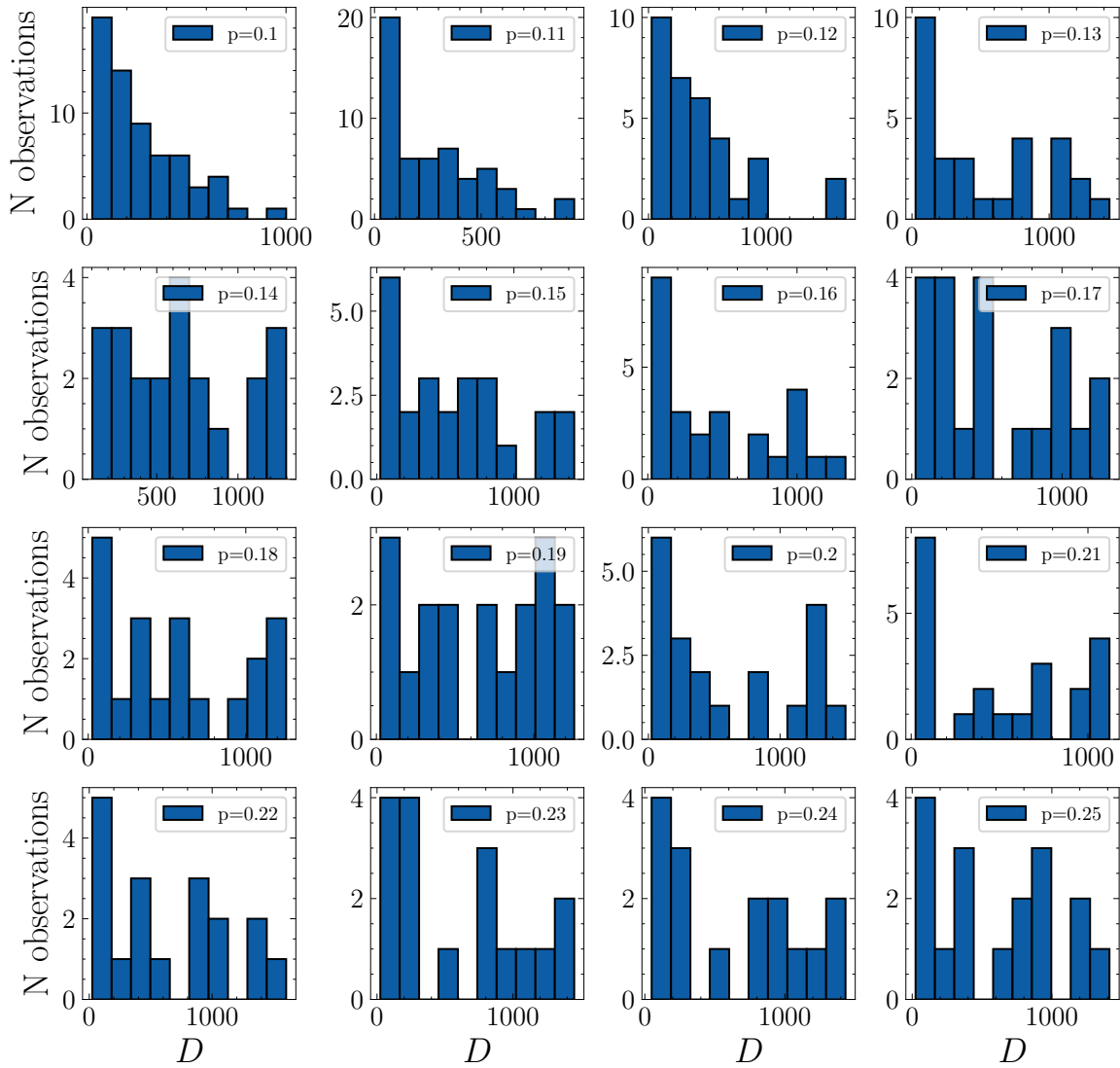


Figure B.32 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 150.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 200.0$

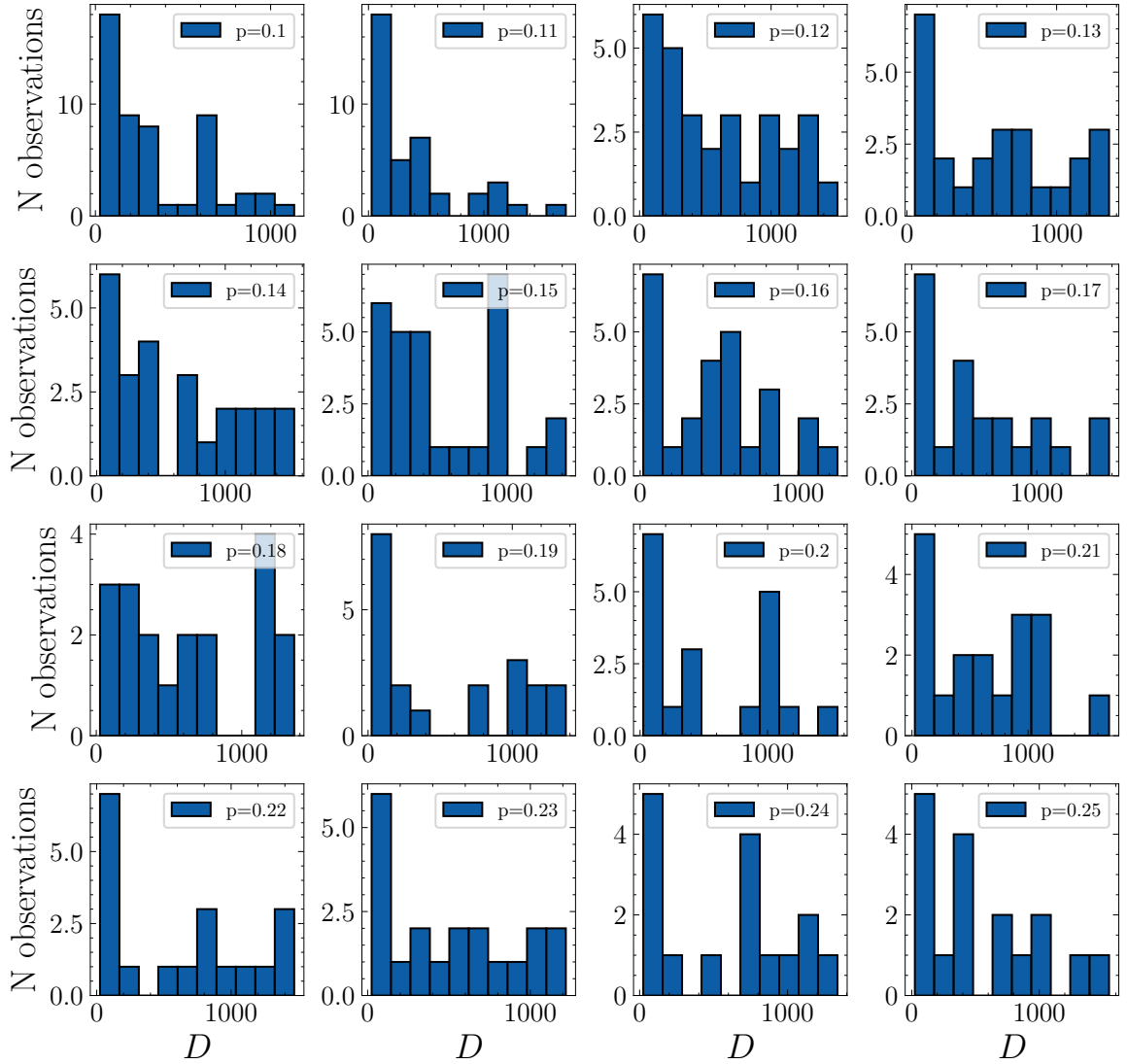


Figure B.33 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 200.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 250.0$

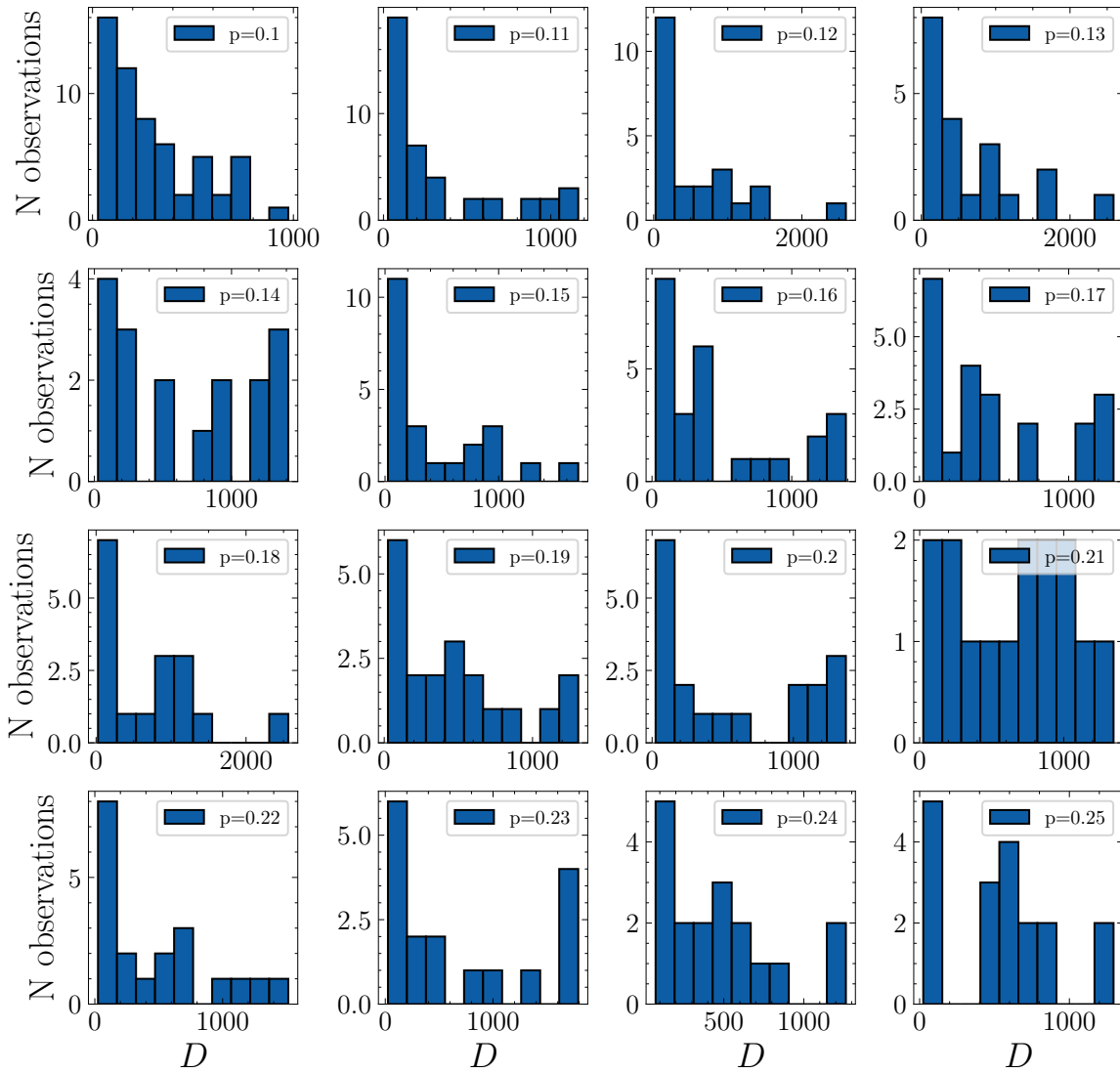


Figure B.34 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 250.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 300.0$

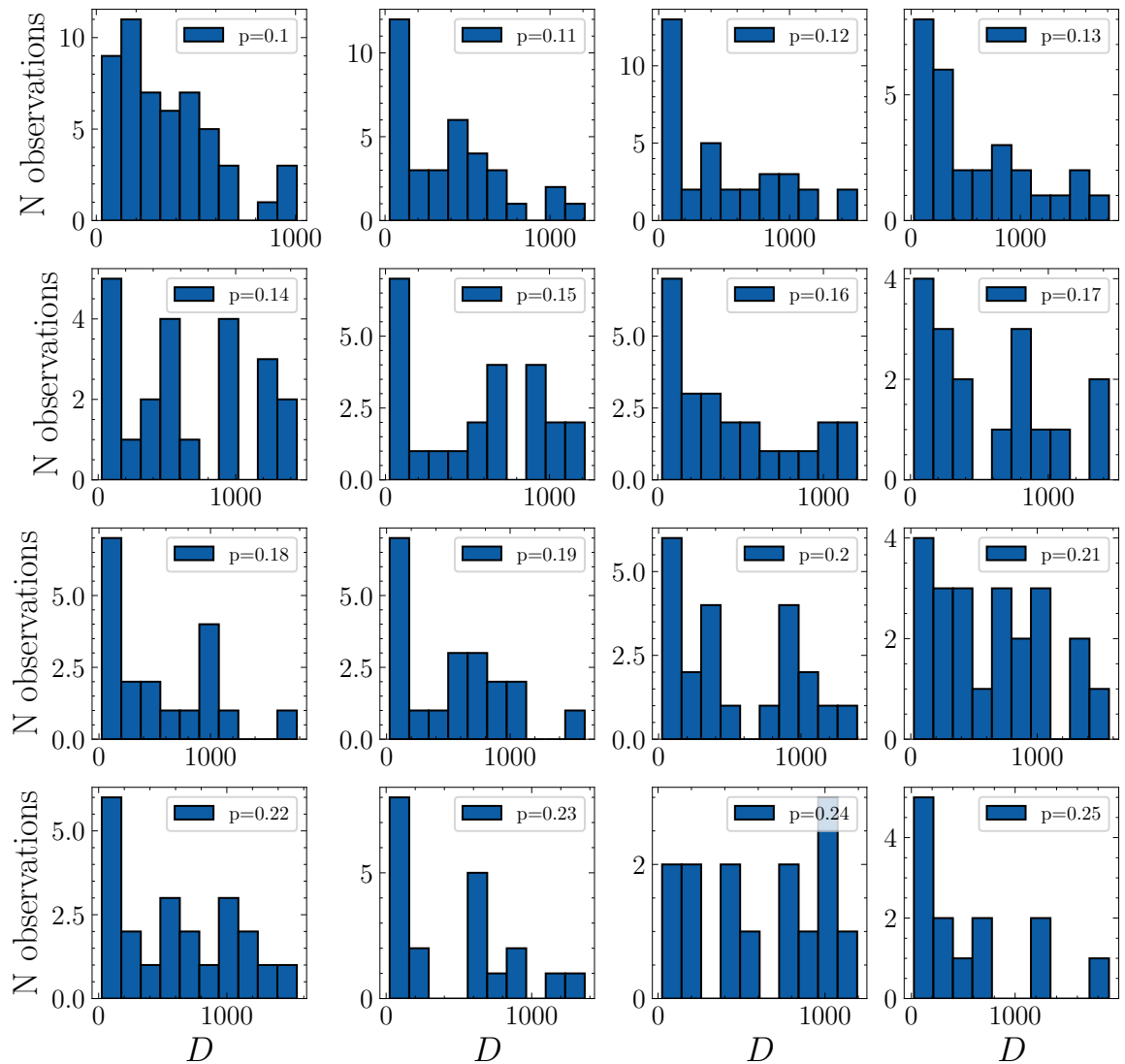


Figure B.35 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 300.0$.

Fonte: o autor (2022).

Histograms for the values of the Diffusion Coefficient $D > 5.0$ for all pressures p for $k = 500.0$

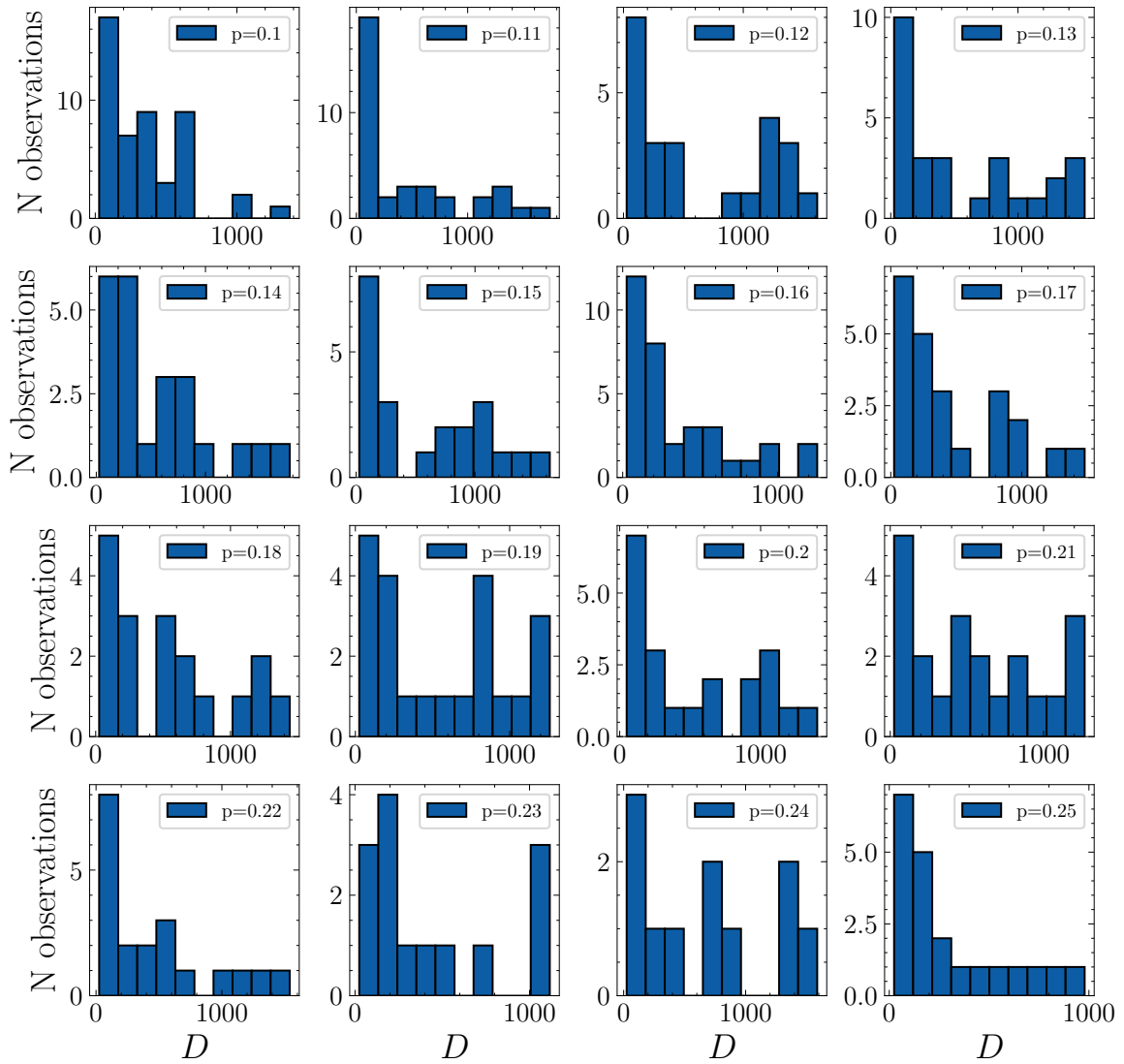


Figure B.36 – This figure provides the histograms for the observed values of the diffusion coefficient larger than 5 for every value of p for $k = 500.0$.

Fonte: o autor (2022).

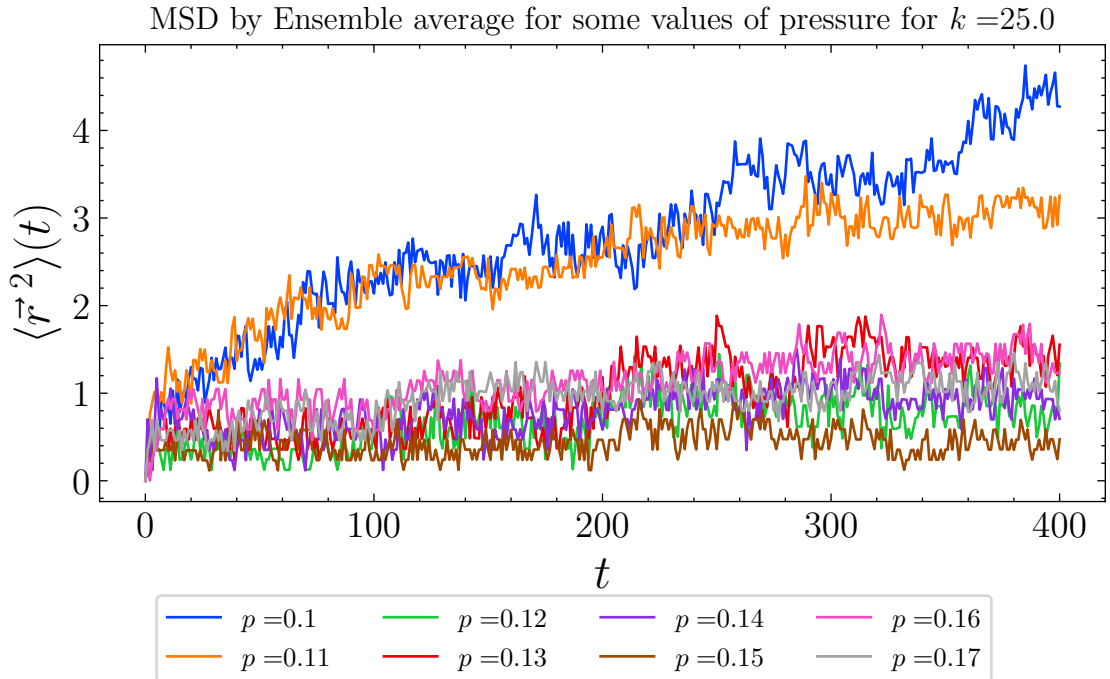


Figure B.37 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 25.0$ for all 401 timesteps.

Fonte: The author.

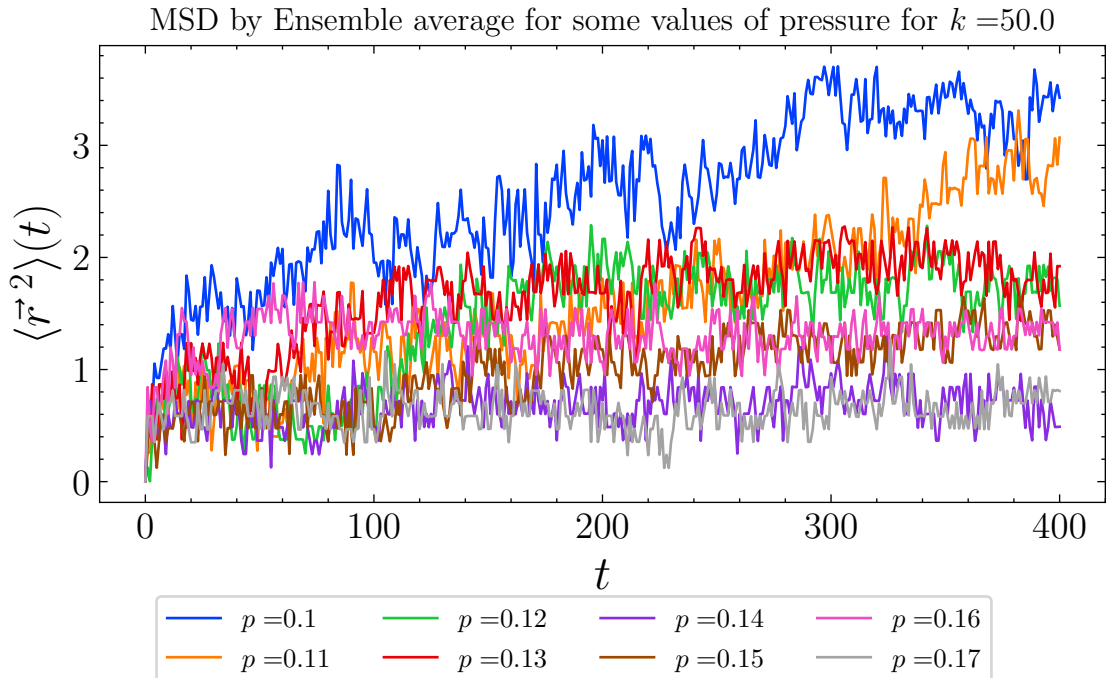


Figure B.38 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 50.0$ for all 401 timesteps.

Fonte: The author.

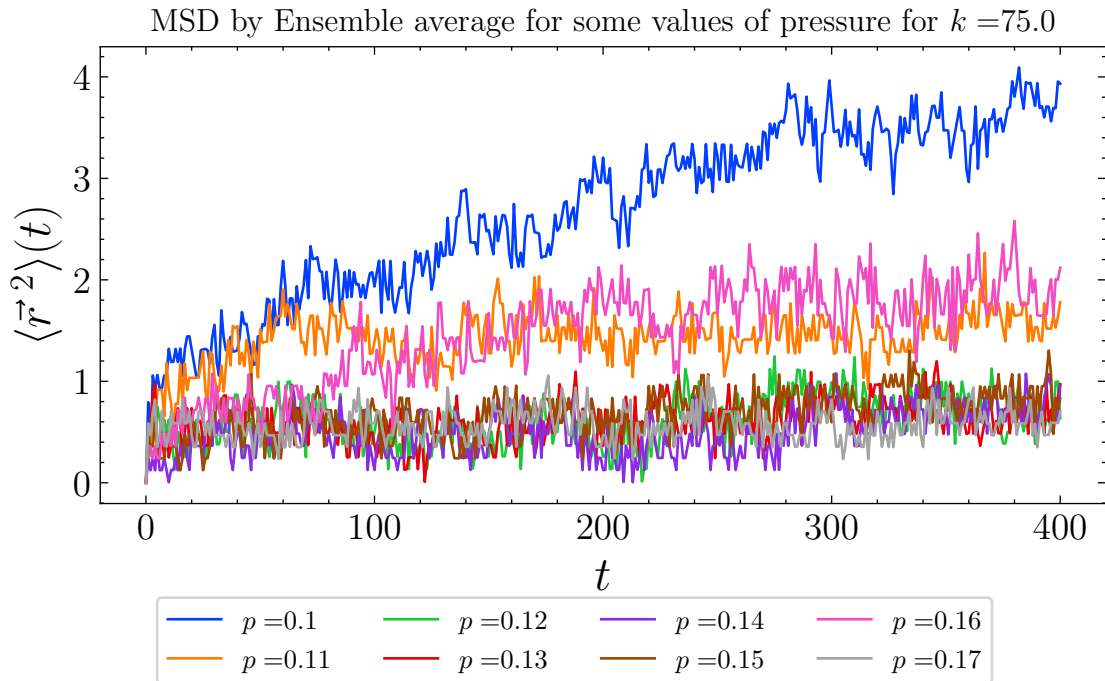


Figure B.39 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 75.0$ for all 401 timesteps.

Fonte: The author.

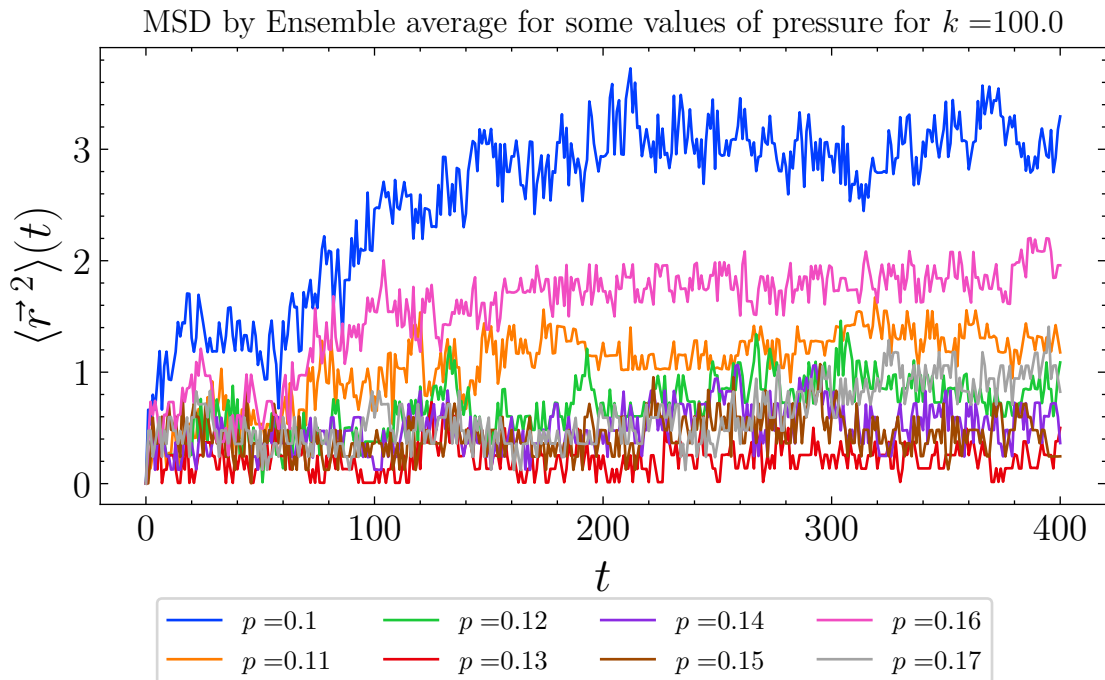


Figure B.40 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 100.0$ for all 401 timesteps.

Fonte: The author.

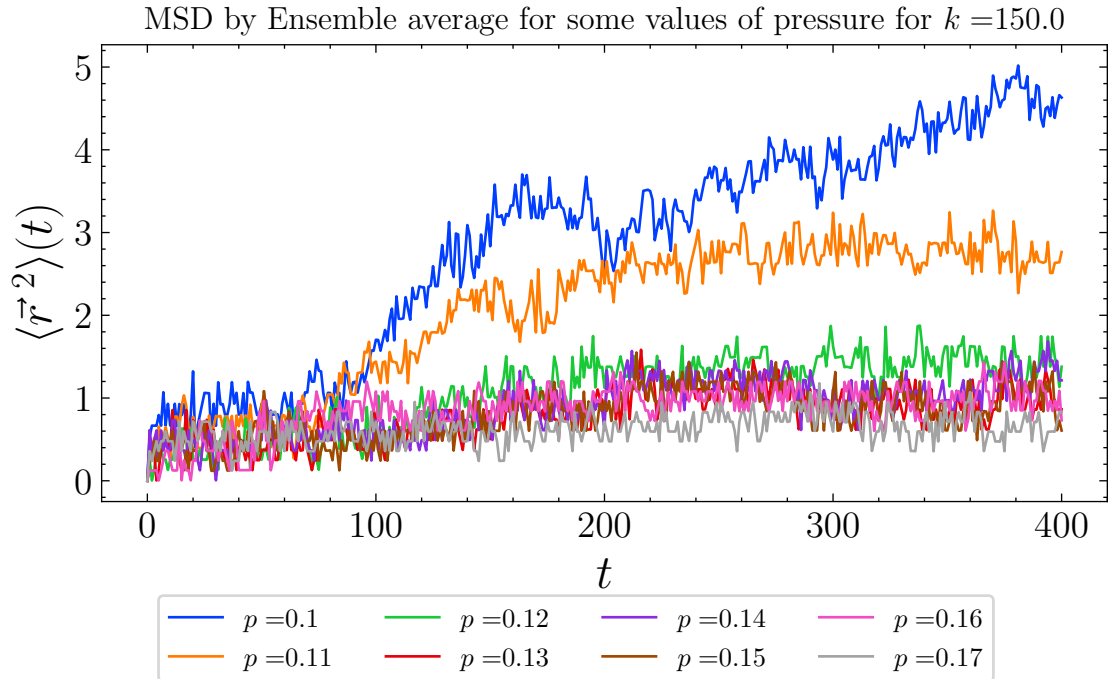


Figure B.41 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 150.0$ for all 401 timesteps.

Fonte: The author.

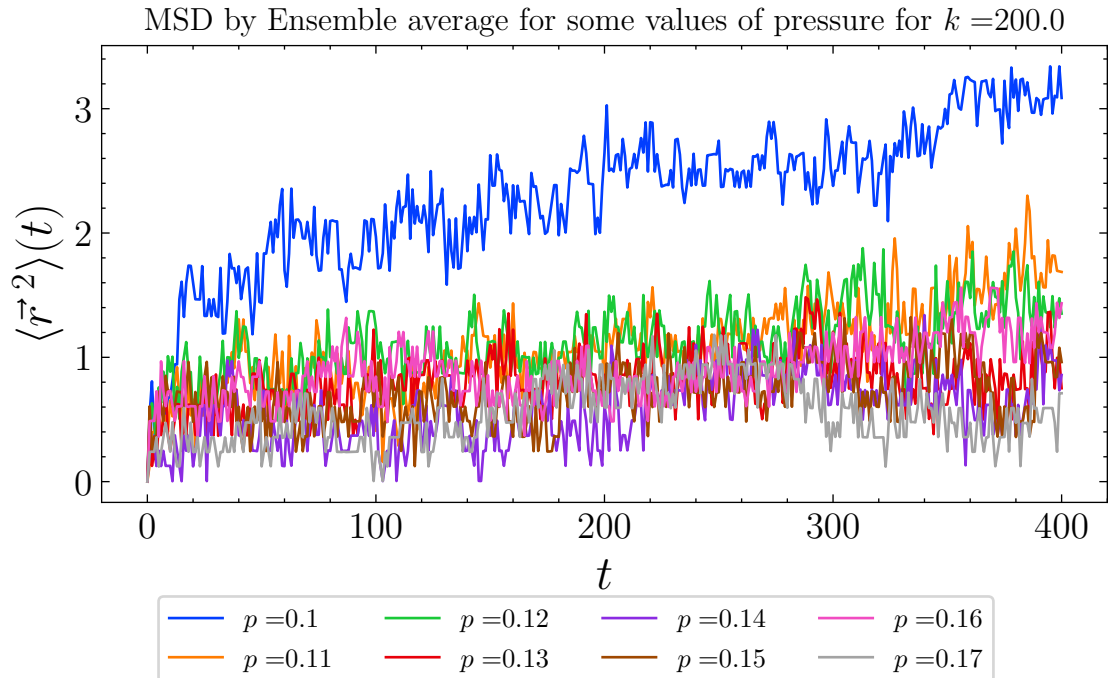


Figure B.42 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 200.0$ for all 401 timesteps.

Fonte: The author.

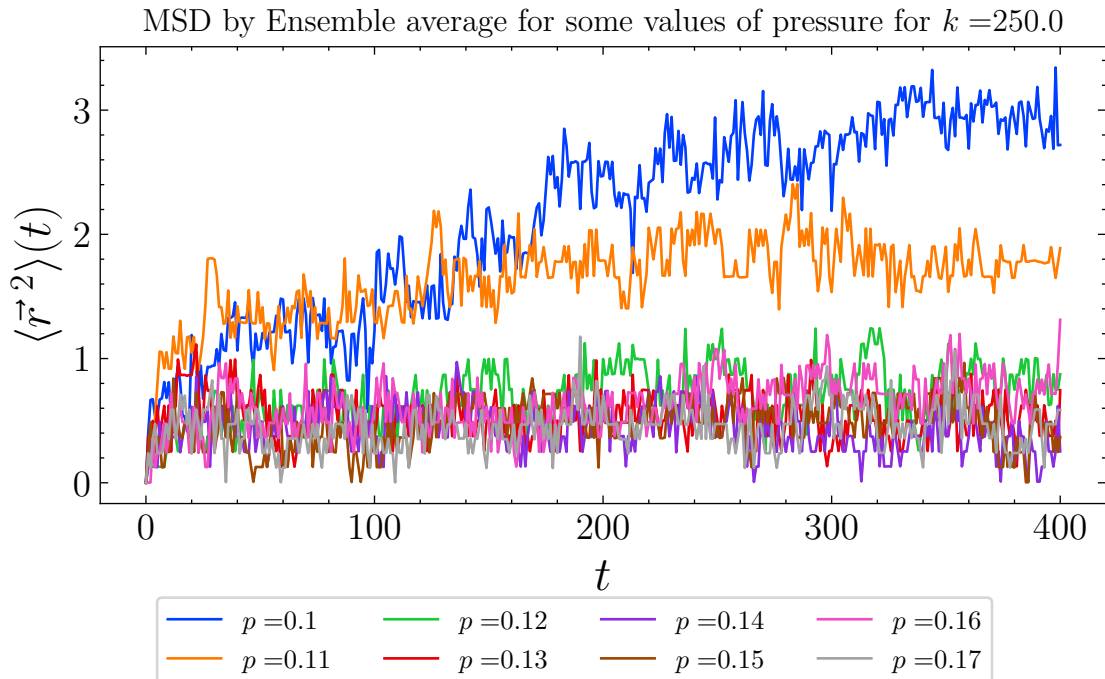


Figure B.43 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 250.0$ for all 401 timesteps.

Fonte: The author.

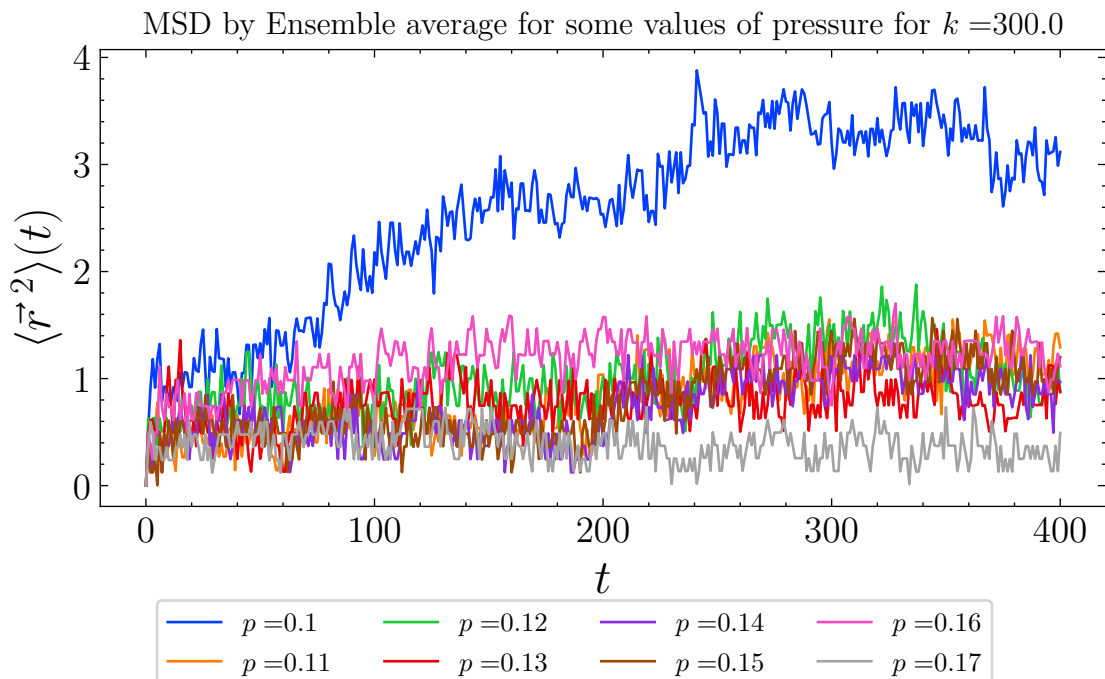


Figure B.44 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 300.0$ for all 401 timesteps.

Fonte: The author.

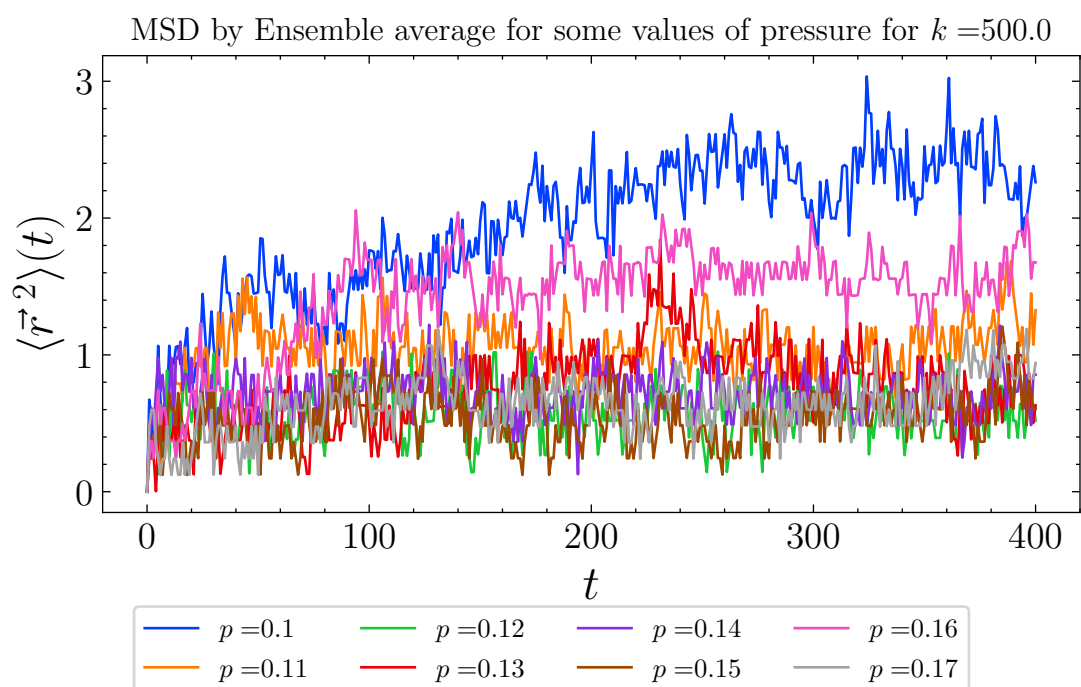


Figure B.45 – This figure provides the MSD by Ensemble average for pressure values of $p = 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16$ and 0.17 for $k = 500.0$ for all 401 timesteps.

Fonte: The author.