Project report

Application of Tensorflow, K-nearest neighbor and DBSCAN algorithms to particle laden turbulence

Authors :

Anis BNINI

Benoit GILLES

Yahia BRAIHEMAT

Mamoudou KOUME

Master 1, Applied Mathematics, Statistics Data Science's Course

Under supervision of :

Kai SCHNEIDER

Professor at Aix-Marseille University

19th April 2021

Acknowledgements

We thank and express our gratitude to our supervisor Kai SCHNEIDER for all the assistance and availability granted in the realization of this project report. We would like to thank Thibault OUJIA, who helped us acquiring the data. We would also like to thank Keigo Matsuda, who provided the data, and the Mesocentre and its entire personnel. Our thanks also go to Aix-Marseille University as well as to those responsible for the master's degree in Applied Mathematics, Statistics, for having us allowed to put into practice our acquired knowledge in a spirit of rigor and initiative to produce this research. May all those who have contributed directly or indirectly to the development of this project find our warm thanks here.

Abstract

We analyzed a total of 110 text files that contain the spatial distribution of particle laden in turbulence, these files stem from a numerical simulation done by a supercomputer known under the name "Earth Simulation" and based in Japan. Our objectives were to characterize and identify the clusters formed from the droplets in turbulence and to shed the lights about how it is conditioned by the Stokes number and Reynolds Number that define these droplet particles . We also attempted from those files to implement an image classification algorithm and to generate synthetic data.We confirmed with DBScan that the number of clusters is negatively correlated with the Stokes number. With a convolutional network built with Tensorflow, we could determine the Stokes Numbers with an accuracy of 79%, on the other hand, KNN clustering guarantees a 100% accuracy on determining the Reynolds number of a sample. Both OpenCV and DBScan show similar results at identifying the number of clusters. Overall, our results showcase the major role that plays the two parameters : Stokes and Reynolds number in determining the shape of the clusters, these two parameters, with the help of KNN clustering and neural networks, can be predicted from an image of a sample flow with a high level of confidence, thus directly determining the number of clusters.

Acronyms and abbreviations

AMICC : Aix-Marseille Intensive Computing Center

API: Application programming interface

CNN: Convolutional neural network

DBSCAN : Density-Based Spatial Clustering of Applications with Noise

DNS : Direct Numerical Simulations

eps : Epsilon

GAN: Generative adversarial network

KNN : K-Nearest Neighbors

minPts : Minimum number of points required to form a cluster

Ng : Number of Grid points (in cubic resolution)

OpenCV : Open Computer Vision

PCA: : Principal component analysis

Re : Reynolds number

St : Stokes number

TF : TensorFlow

WCSS : Within Cluster Sum of Squares

Contents

1 Introduction

Droplets-laden in turbulence are a natural phenomenon governed by the principles of fluid mechanics. It is commonly observed in the environment for example in the form of atmospheric aerosol particles, rainfall or during the occurrence of a cyclones. It it also encountered in very wide processes of the industry, noticeably and among many others in wastewater treatment or in a fluidized bed. It is therefore primordial to have a great knowledge of the behavior of particle-laden turbulence, whether it is to improve the efficiency and productivity while reducing the cost in the industry, or to cope better with pollution and some natural disasters regarding the environment.[1][2]

A lot of research has been carried out so far on the topic, however due to the limitations of the experimental tools and the complexity of the dynamics of these particles , there remains many significant points outstanding noticeably regarding the dispersion of these particles under turbulence, the literature assume that they tend to aggregate to one another in some specific regions forming what we call clustering regions. There is also the opposite effect that happens leading to zones -almost- void of particles. These clustering regions if existent vary in size and shape. [3]
The particles are too of diverse size, there are two physical dimensionless numbers that can characterize their behaviour in turbulence known as the Stokes number (St) and the Reynolds number (Re). The way the particles cluster depends strictly on these two numbers.

Throughout this study and with the use of multiple machine learning algorithms and libraries (KNN- DBScan and TensorFlow) on the numerically simulated data, we have set to ourselves the objective of improving our understanding about the formation of these clusters and predict the behavior of these droplets in turbulence given some of their physical properties. In the first chapter, we focused on identifying these clusters and their size as well as the influence of both the Stokes and Reynolds number on them using the k-nearest neighbors algorithm. The second chapter covered an identical problematic but with a different approach, using DBSCAN this time. Finally in the last chapter we intended to implement an image classification algorithm that predicts the St and Re numbers of droplets-laden particles based on the image of their spatial distribution, we also tried to generate synthetic spational data of particle laden in turbulence with Tensorflow.

2 General framework of the study

2.1 Context and origin of the data

≪Found abundantly in natural and engineered systems, small particles can alter fluid turbulence due to thermal feedback and momentum. Particles of different sizes, shapes and other variables can control the degree of interaction with the fluid. Namely, their inertial characteristics, gravitational settling and mass fraction are sources of feedback against fluid flow[4]≫. For example, radar remote sensing can provide estimates of the microphysical properties of clouds and precipitation particles [5]. The data used in this study are direct numerical simulation of particle-laden homogeneous isotropic turbulence by solving Navier-Stokes equations on supercomputers :

$$\begin{aligned} \frac{\partial u_i}{\partial x_i} &= 0, \\ \frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{x_j} &= -\frac{1}{\rho_a}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + F_i \end{aligned} \tag{2.1}$$

where u_i is the flow velocity in the i^{th} direction, p is the pressure, ρ_a is the air density, ν is the kinematic viscosity, and F_i is the external forcing term. The time integration was calculated by the second-order Runge–Kutta scheme [5].

Droplets movements were simulated by tracking Lagrangian point-particle tracking using the equation below :

$$\frac{\mathrm{d}v_i}{\mathrm{d}t} = -\frac{v_i - u_i}{\tau_p} + g_i \tag{2.2}$$

where v_i and g_i are the particle velocity and gravitational acceleration in the i^{th} direction, τ_p is the droplet relaxation time, which is given by :

$$\tau_p = \frac{\rho_p}{\rho_a}\frac{2r_p^2}{9\nu} \tag{2.3}$$

The data have been kindly provided by Dr. Keigo Matsuda. Access to those data was granted after request by the mesocentre AMICC at Aix-Marseille university, one of the high-performance computating centers in France.

2.2 Data description

In this study we analyze direct numerical simulation data of particle laden turbulence within turbulent flows in a periodic box. The data were obtained by solving the Navier-Stokes equations including inertial point particles on the Earth simulator, the supercomputer at JAMSTEC in Yokohama, Japan. We consider inertial point particle for different Reynolds numbers, which characterize the turbulence intensity and different Stokes numbers which characterize the inertia of the particles. Low Stokes numbers correspond to light particles, while large Stokes numbers correspond to heavy particles. Note that both parameters, the Reynolds and the Stokes numbers are non-dimensional numbers.

We have 110 files given in text format. Each of these files contains the particle positions at a given time instant in either a three-dimensional cube, or in a 2 dimensional thin slice.

2.3 Data visualization

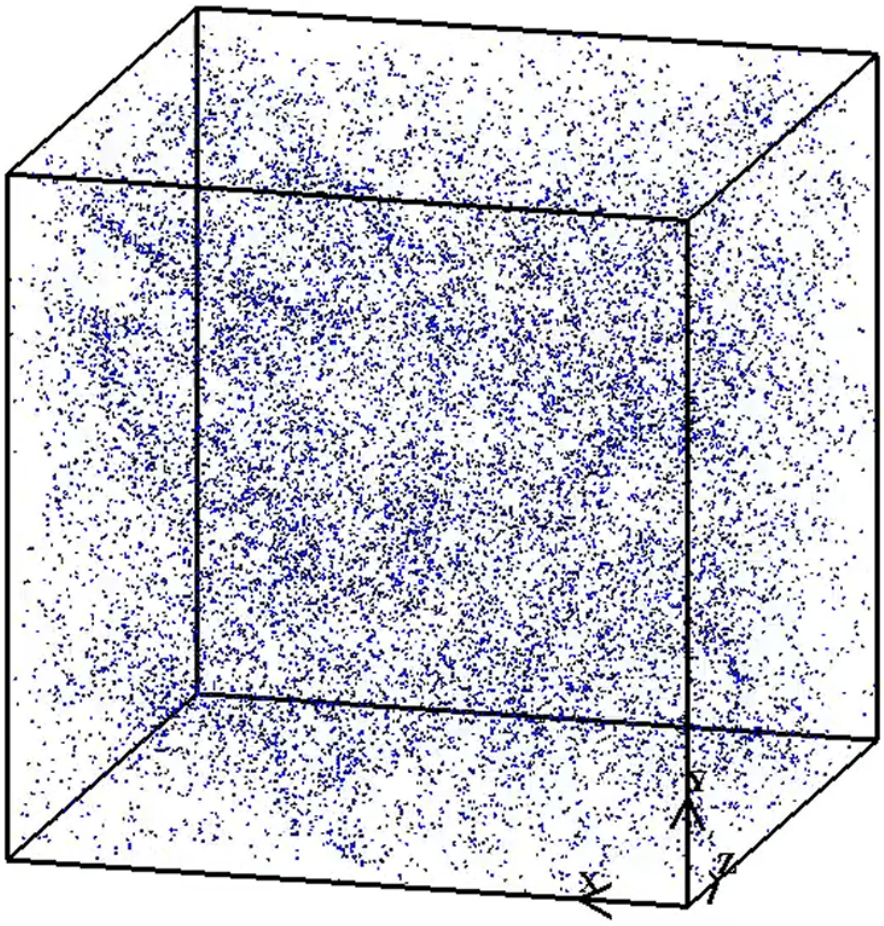


Figure 2.1: Representation of 3D data for $Re_\lambda = 91$ And $St = 1$ at a given time instant

2.4 Explanation of parameters

2.4.1 Reynolds number

In these experiments carried out in 1883 with the flow of a fluid in a straight cylindrical pipe, Reynolds proved the existence of two flow regimes: laminar and turbulent. He found, using different fluids and varying the flow rate and pipe diameter, that the parameter that determined whether the flow was laminar or turbulent was a dimensionless number. Later, this number will be called the reynolds number.

The reynolds number is defined by the relation

$$Re = \frac{v\ D}{\nu} \tag{2.4}$$

v = characteristic velocity of the flow, D = pipe diameter, ν = Kinematic viscosity

The critical Reynolds number for internal flow is:

$$\begin{cases} \text{if } Re < 2000, & Laminar flow \\ \text{if } 2000 < R_e < 4000, & Transient flow \\ \text{if } Re > 4000, & laminar flow \end{cases}$$

Here the figure below, we have the different cases according to the value of the Reynolds number.

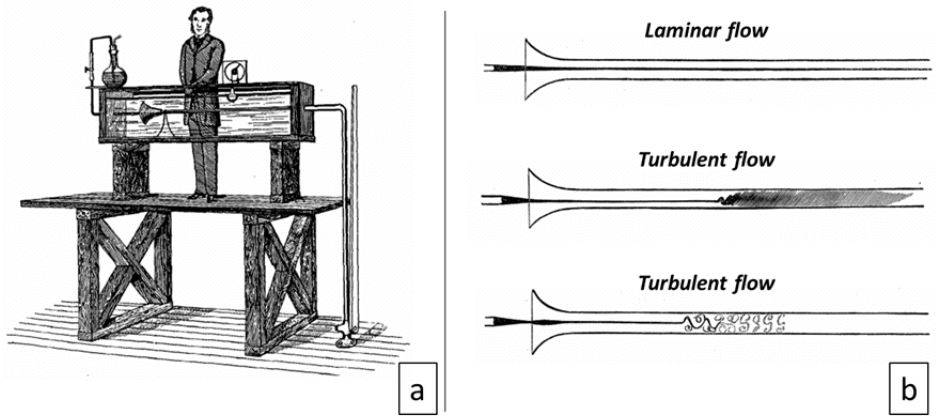


Figure 2.2: Reynolds number and nature of flow

Source : https://www.simscale.com/docs/simwiki/numerics-background/what-is-the-reynolds-number

The Reynolds number thus characterizes the turbulence intensity of the flow, flows with $Re < 1$ are dominated by viscous effects and flows with $Re >> 1$ are highly turbulent and dominated by inertial effects, reflected in vortices and a random behavior.

The computational domain was defined on a cube with edges of length $2\pi L_0$, where L_0 is the representative length scale. Periodic boundary conditions were applied in all three directions. The domain was discretized uniformly into N_g^3 grid points.

The turbulence Reynolds number is calculated based on the Taylor microscale define as [6] :

$$Re_\lambda = \frac{l_\lambda \, u_{rms}}{\nu}$$

where u_{rms} is the RMS value of the velocity fluctuations and l_λ is the Taylor microscale and the kinematic viscosity was set to $1.5 \times 10^{-5} \; m^2 \; s^{-1}$.

The resolutions (N_g) were chosen to satisfy $k_{max} \, l_\eta \approx 2$, where k_{max} is the maximum wavenumber given by $k_{max} = N_g(2L_0)^{-1}$ and l_η is the Kolmogorov scale, in [6] DNS experiments the nondimensional energy dissipation rate was essentially the same for all the flows.

The DNS was performed for two turbulent flows, each with a different value of the Taylor-microscale-based turbulent Reynolds number :

N_g	$L_0(m)$	u_{rms}	Re	Re_λ	$k_{max} l_\eta$
512	0.0400	0.345	909	204	2.02
1024	0.0666	0.499	2220	322	2.06

Table 2.1: Computational parameters for DNS and the statistical results

The number of droplets was set to 1.5×10^7 and 5×10^7 for $Re_\lambda = 204$ and $Re_\lambda = 322$, respectively.

2.4.2 Stokes number

The Stokes number (St) is a dimensionless number used in fluid dynamics to characterize the behavior of a particle in a fluid. It represents the relationship between the kinetic energy of the particle and the energy dissipated by rubbing with the fluid.

This number is named after George Gabriel Stokes, an Irish physicist and mathematician.

It is defined as:

$$St = \frac{\rho_p \, d_P^2 \, v}{\nu \, L_c} \tag{2.5}$$

ρ_p = Particle density, d_p^2 = Characteristic length of particle, v = Fluid speed, ν= Dynamic fluid viscosity, L_c= Characteristic length

This number is used to determine the behaviour of a particle in a fluid in the face of an obstacle, including whether the particle will bypass the obstacle by following the movement of the fluid or whether it will hit the obstacle [7] and [8].

2.5 Clustering

2.5.1 Definition

Basically, a cluster is a collection of elements. Every set is distinct from the others. So each element of a cluster has strong similarities with the other elements of the same cluster, and must be different from the elements of other clusters [9]. Data clustering is a method in data analysis that aims to divide a set of data into different homogeneous «packets », in the sense that the data of each subset share common characteristics, which most often correspond to criteria of proximity (computer similarity) that we define by introducing measures and classes of distance (usually Euclidean distance) between objects.

2.5.2 Cluster properties

The criteria to be met by an unsupervised learning algorithm are less obvious to define than in the case of supervised algorithms, where there is a clear task to accomplish. This does not prevent the existence of measures of the performance of an unsupervised algorithm [9]. A cluster is said to be relevant if it satisfies the following two important properties :

- **internal cohesion** (objects belonging to this cluster are as similar as possible)
- **external insulation** (objects belonging to other clusters are the most distant possible)

To verify these properties, we associate the cluster with notions of measurement (distance and similarity) by looking for its :

- **density** (mass of objects per volume unit)
- **shape** (concave, convex, hyperspheric...)
- **dimension** (size and radius/diameter)
- **variance** (degree of dispersion of objects in space)
- **separation** (compared to other cluster)

To check the requirements (homogeneity and separation) of a clustering, we can measure the silhouette coefficient which makes it possible to assess whether a point belongs

to the right cluster. For a given point x, the silhouette coefficient is given by [10] :

$$s(x) = \frac{b_l(x) - a_k(x)}{max(a_k(x), b_l(x))}$$

where :

- $a_k(x)$ is the average distance from x to all other points in the cluster k;
- $b_l(x)$ is the smallest value that $a(x)$ could take, if x were assigned to another cluster $l \neq k$.

Also, the average separation of the clusters taken by pair can be expressed as follows :

$$S = \frac{2}{k(k-1)} \sum_{k=1}^{K} \sum_{l=k+1}^{K} d(\mu_k, \mu_l)$$

where : $d(\mu_k, \mu_l)$ is the separation of two clusters k and l seen as the distance between their centroids (barycenter of points of a cluster).

2.5.3 Clustering methods

There are two main categories of clustering [9] :

- **Hierarchical methods**
  The goal is to form a hierarchy of clusters. Depending on the top-down or bottom-up approach chosen, we look for clusters that are specific to a certain number of objects considered like similar.
- **Partitioning methods**
  Here, the goal is to form a partition of the space of objects, according to a certain criterion function, each partition then representing a cluster in this category. Among these methods, we have :
  - K-means
  - DBSCAN
  - clustering based on neural network, etc.

3 Application of the KNN/KMeans algorithms to particle-laden turbulence

In this part, we describe the two different algorithms, K Nearest Neighbors and KMeans, and how to use them for our problem. The main goal of the project is to identify and to characterize the clusters in particle laden turbulence and to analyze the influence of the Stokes number and the Reynolds number of the flow. Each goal will require one algorithm, and one strategy.

3.1 KMeans

3.1.1 Definition

KMeans is an unsupervised learning algorithm that partitions the data in clustering problems into K clusters. It uses the Euclidean distance to calculate the proximity between points, and tries to minimize the squared Euclidean distance between points and the centroid of the clusters. The goal is to find the best centroids, and then allocates every data point to the corresponding cluster, by keeping the WCSS as low as possible.

3.1.2 How does the KMeans algorithm work ?

The first step in the KMeans algorithm is to define K random centroids. After they have been generated, the algorithm does many iterations to optimize the position of the centroids. Let us imagine one cluster contains five data points, and is defined by one centroid. In the next iteration, the centroid location will becomes the mean of the location of all the points in the current cluster. This way the algorithm tries to stabilize itself while reducing the $WCSS$, which is defined in the next part. It either stops when the algorithm has reached a stable position, which means the centroids are perfectly located or when the number of iterations has reached a predefined limit.

3.1.3 Deciding the number of clusters

The elbow method

The first method we used to decide how many clusters we need to consider is the elbow method. It uses the $WCSS$ as its criteria, with $WCSS = \sum_{i=1}^{N_c} \sum_{x \in C_i} d(x, \tilde{x}_{C_i})^2$,

where N_c corresponds to the number of clusters, d is the Euclidean distance, and $\tilde{x}_{C_i}$ is the centroïd of the cluster i [11]. After plotting the $WCSS$ depending the number of cluster, we look for an elbow curve and the location of a bend in the plot is generally considered as an indicator of the appropriate number of clusters.

We created the following example of a result using the elbow method to find the optimal number of cluster in a flow with St equal to 0.5 and Re equal to 2220 :

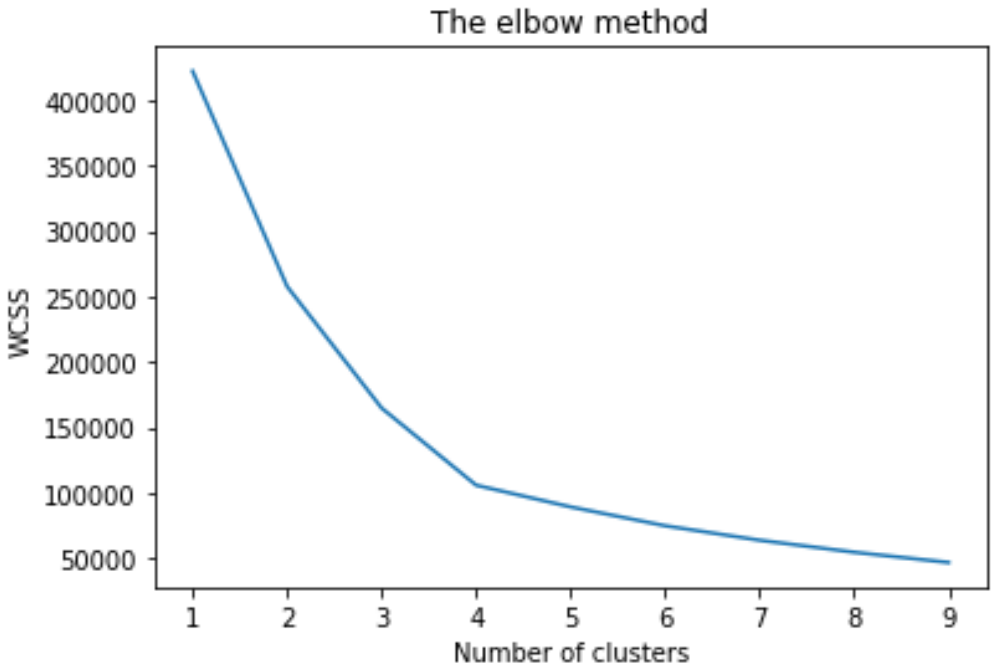


Figure 3.1: WCSS in function of the number of clusters, for a flow with $Re = 2220$ and $St = 0{,}5$

In the figure 3.1, we can observe the $WCSS$ as a function of the number of clusters. In this scenario, both values three and four are correct for the number of cluster as we can see that we can reduce the number of clusters from five to four and also from four to three without increasing the $WCSS$ too much. We limited the maximum number of clusters to nine due to a huge prediction time. On top of that, we assumed that since we obtained an elbow curve while having a range of one to nine for the number of clusters it was not necessary to go any further.

The Gap Statistic Method

For the Gap Statistic method, we need to choose the value of K that maximises the gap statistic.

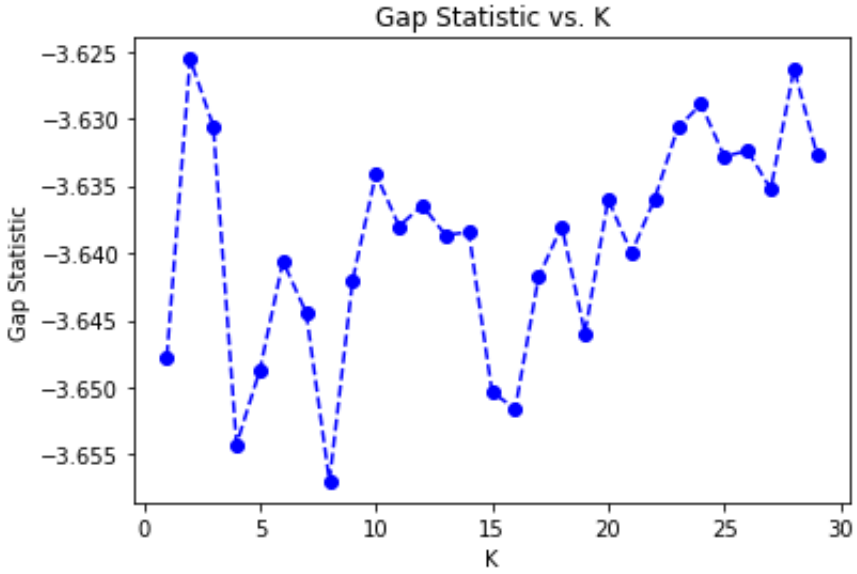


Figure 3.2: Example of a result using the gap statistic method, $Re = 2220$ and $St = 0{,}5$

As we can see in the figure 3.2, the Gap statistic peaked when K is equal to two. Thus it indicates that two should be the optimal number of clusters.

The Silhouette method

As written on Wikipedia, «Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object has been classified.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

Assume the data have been clustered via any technique, such as k-means, into K clusters.

For a data point $i \in C_i$ (data point i in the cluster C_i), let $a(i) = \frac{1}{|C_i|-1} \sum_{j \in C_i, i \neq j} d(i,j)$ be the mean distance between i and all other data points in the same cluster, where $d(i,j)$ is the distance between data points i and j in the cluster C_i (we divide by $|C_i|-1$ because we do not include the distance $d(i,i)$ in the sum). We can interpret $a(i)$ as a measure of how well i is assigned to its cluster (the smaller the value, the better the assignment). We then define the mean dissimilarity of point i to some cluster C_k as the mean of the distance from i to all points in C_k (where $C_k \neq C_i$). For each data point $i \in C_i$, we now define $b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$ to be the smallest (hence the min operator in the formula) mean distance of i to all points in any other cluster,

of which i is not a member. The cluster with this smallest mean dissimilarity is said to be the neighboring cluster of i because it is the next best fit cluster for point i. »

We then define the silhouette score as follows :

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases} \quad [12]$$

The silhouette value varies between -1 and 1. A high value for the silhouette score for a number of cluster equal to K indicates that we should choose K as the optimum value for the number of clusters for our sample.

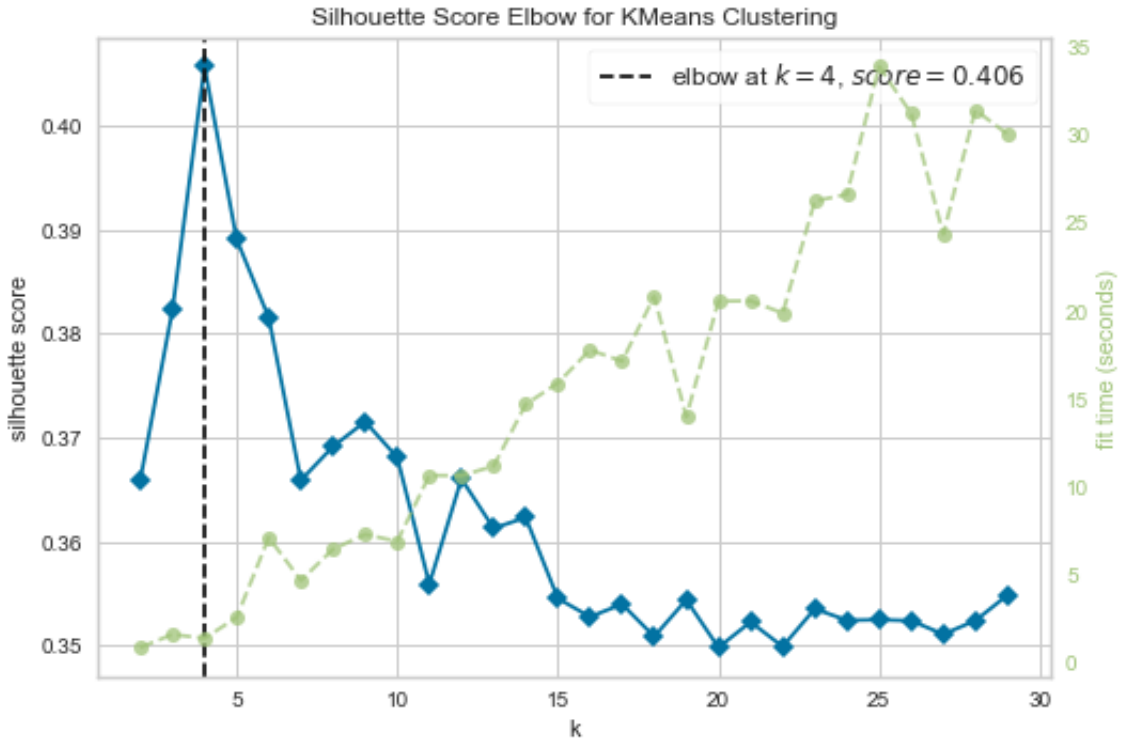


Figure 3.3: Example of a result using the silhouette method, $Re = 2220$ and $St = 0{,}5$

In the example shown in figure 3.3, the optimal number of clusters is four as we observe a peak for the silhouette score when the value of K is equal to four.

Cluster detection with the OpenCV Library

The main problem with the last three methods is the fact that the number of clusters they give is too small. They basically divide the image created from one sample into two or four clusters. Thus, no information was obtainable for the clusters. On top of that, the computing time to obtain those results (i.e. to find the optimal number of clusters) for only one sample was quite expensive, approximately 20 minutes for one sample.

We then decided to find another way to detect clusters. We thought about creating a dendrogram, but after our first attempt we had a memory issue, thus making this solution not viable. After some research on the Internet, we decided to try to use the OpenCV library and to detect the number of objects (i.e. clusters) in the images created from our 2D samples.

≪OpenCV is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products[13].≫

Since python usually shows rectangular images, we needed to resize the images created to visualize the data due to the fact that the data are 2d slices of a cubic box, so the side lengths are the same. We went from rectangular images to square ones and following is an example of a resized image.

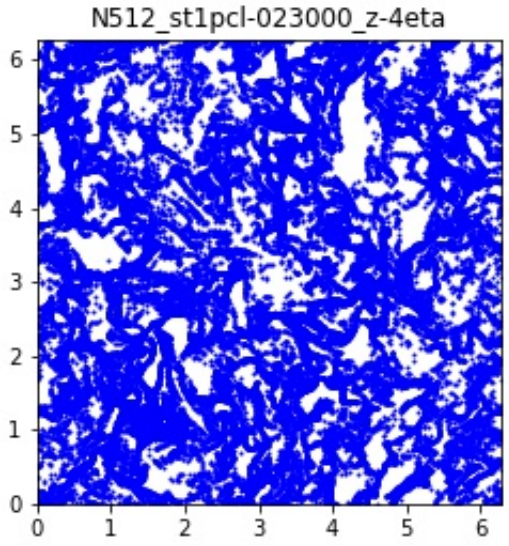


Figure 3.4: Square image of a 2D slice of a sample flow, with $Re = 909$ and $St = 1$

Figure 3.4 shows a sample flow with the characteristics $Re = 909$ and $St = 1$. We can observe voids and high density zones. Even if there are no major differences between the two representations, the number of clusters detected with the OpenCV library varied between rectangular images and square images. Those figures show the number of objects detected for the different samples for square images :

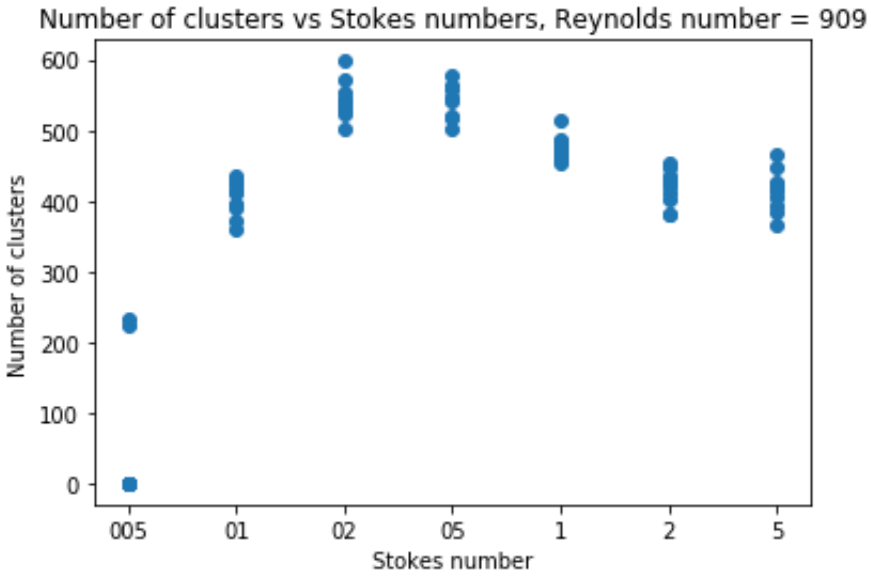


Figure 3.5: Number of clusters detected with the OpenCV library, for $Re = 909$, as a function of St

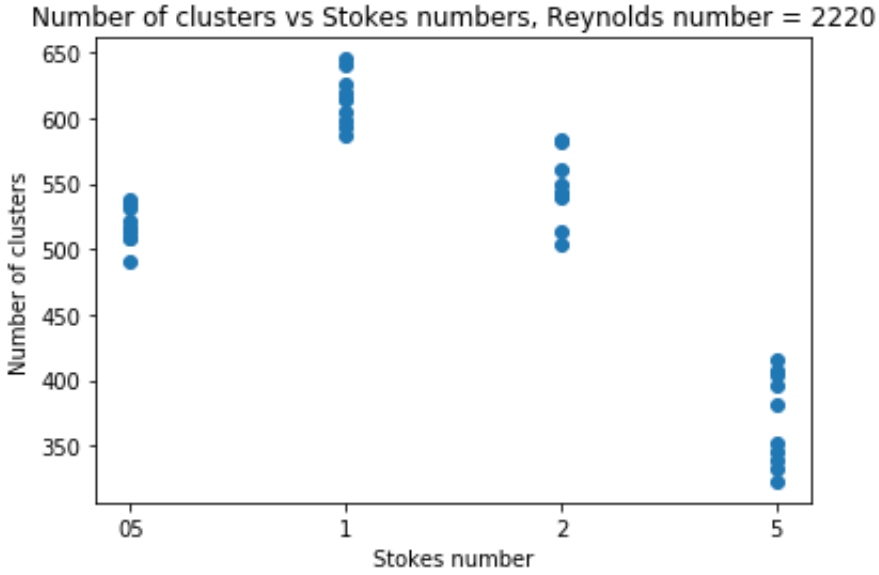


Figure 3.6: Number of clusters detected with the OpenCV library, for $Re = 2220$, as a function of St

Figure 3.5 shows the number of clusters detected with the OpenCV library for the 2D sample flows with Re equal to 909. We observe something interesting as our method detects only one cluster for some samples with a Stokes number equal to 0,05. It can be explained since a flow with this Stokes number has a high density of particle, thus making it impossible to visually detect many clusters in our problem. It could be interesting to analyse the distribution of those results with a higher range of Stokes numbers. The number of clusters seems to peak for flows with a Stokes number equal to 0,2 with a maximum value for the number of clusters detected equal to 600.

On the contrary, while analyzing the figure 3.6 which represents the number of clusters detected with the OpenCV library for the 2D sample flows with Re equal to 2220, the maximum number of detected clusters in one image peaks at 650, for a Stokes number equal to 1. Those different results may be a consequence of the imprecision of the object detection feature of the OpenCV library, for a huge number of points.

3.2 K Nearest Neighbors

3.2.1 Definition

The KNN algorithm is described as follows by Venkatesh Umamaheswaran : ≪Contrary to the Kmeans algorithm, the K Nearest Neighbors algorithm is a supervised classification algorithm. It takes a bunch of labeled points and uses them to learn how to label other points. To label a new point, it looks at the labeled points closest to that new point which are its nearest neighbors, and has those neighbors vote. So whichever label, the most of the neighbors have is the label for the new point [14].≫

Here, we know the number of cluster K. Since we are going to apply the KNN algorithm to classify our data regarding their characteristics, we can choose either the number of different Reynolds numbers (2), or the number of different Stokes numbers (7). Our entry data will be the 1D arrays from our initial data, and the result will be either the Stokes number, or the Reynolds number associated to the 1D array.

First, we separate our data into training and testing data, with the ratio 40/60, and we create our model. After fitting our training data to the model, we fit the testing data, and test the accuracy of our model : For each input data, we look whether the output given by the model corresponds or not to the real output, and then count the correct number of output and divide it by the total number of input data.

3.3 Results

3.3.1 KMeans

Here is an example of an image representation of a sample flow with the different clusters colorized :

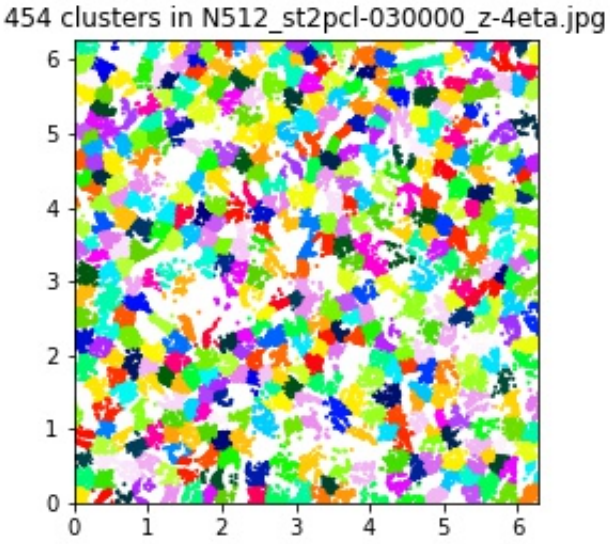


Figure 3.7: Representation of colorized clusters in a sample flow with $Re = 909$ and $St = 2$, colors randomly generated to separate clusters

We can see various shapes and sizes for the clusters in the figure 3.7, each color representing one cluster. Please note that the white zones correspond to voids. Other colors has no meaning and were randomly chosen via a colormap from the library matplotlib on Python. After comparing different representations of flows with different characteristics, we did not find any interesting result. One thing to keep in mind is that we are conscious that the method using OpenCV to detect objects and to assign the detected number to the number of clusters is a visual method. It is not viable due to the fact that it depends on the focus of the image, and not the similar properties of multiple particles to detect clusters. However, this was the only method usable with this algorithm due to the results with the other three commonly used methods, giving a value for the optimum number of clusters between one and ten. Another result that may be interesting is the difference between the figures 3.5 and 3.6. If we detect higher numbers of objects while representing flows with a Reynolds number equal to 2220, it will be maybe easier to cluster images of flow according to their Reynolds number than according to their Stokes number. This result will be developed in the following part.

3.3.2 KNN

We started by clustering our samples according to their Reynolds number. After a first test, we obtained an accuracy of 60 percent approximately. It is not a bad result, but we thought we could do better. We tried to change the distribution of the training/testing data, but we merely gained few percent. So we pre-processed the data : instead of taking our initial data, which consists of a scatter plot of more than 70,000 points, we tried to plot the data with the hist2D function, which in our case, consists of separating the plot into rectangles and counting the number of particles inside each rectangle. Following is an example of a modified image with the hist2D function :

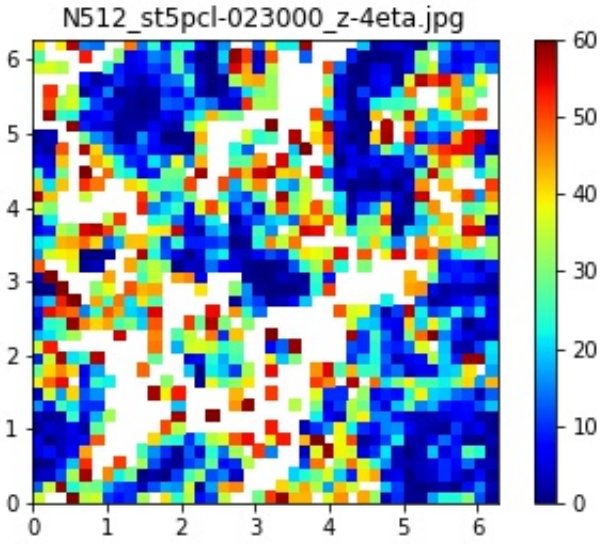


Figure 3.8: Representation of a sample flow with $Re = 909$ and $St = 5$ separated in 50*50 bins, each bin colored according to the number of particles inside

In the figure 3.8, we modified a normal representation (such as the one in the figure 3.4) with the hist2D function, from Python. We capped the number of particles in one square to 60, to avoid having scale problems. Following are some results obtained while printing the accuracy of each model. The accuracy is obtained by comparing the number of correct guesses from one model to the total number of tries. If we put 30 samples in the model, and only 15 were given the correct characteristics, the accuracy will be equal to 50 percent.

```
   ...:
   ...: print('KNN Accuracy: %.3f' % accuracy_score(y0_test,predictions))
KNN Accuracy: 0.106
```

Figure 3.9: Accuracy while training the model to determine the Stokes number of a sample flow, inputting the non modified square images

```
   ...: print('KNN Accuracy: %.3f' % accuracy_score(y2_test,predictions))
KNN Accuracy: 0.288
```

Figure 3.10: Accuracy while training the model to determine the Stokes number of a sample flow, inputting the modified square images with the hist2D function

```
   ...: print('KNN Accuracy: %.3f' % accuracy_score(y1_test,predictions))
KNN Accuracy: 0.576
```

Figure 3.11: Accuracy while training the model to determine the Reynolds number of a sample flow, inputting the non modified square images

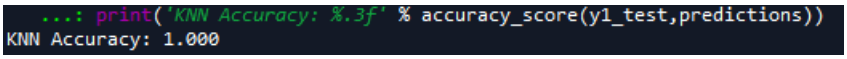


Figure 3.12: Accuracy while training the model to determine the Reynolds number of a sample flow, inputting the modified square images with the hist2D function

As shown in the figures 3.9 to 3.12, the accuracy jumped from 60 percent to 100 percent for the model trying to determine the Stokes number associated to a sample flow by pre-processing the data. However, while trying to classify the data according to their Stokes number, the best accuracy we could obtain was almost 30 percent, even after applying the hist2D function to the image representation of the sample flows. Trying other tools to pre-process the image representation of our data may be the key to a higher accuracy.

3.4 Pros/Cons of KMeans and KNN

Both KNN and KMeans are really easy to understand and to implement in Python. KNN allows to add data as it does not require training. However, both algorithms have a high prediction cost for large datasets. With more than 70,000 points for one data set, the time needed to calculate the distance between each point and their corresponding centroid become quickly high. The main methods for determining the optimal number of clusters do not work well too. On top of taking a lot of time, the results may range from two to 15 clusters. This number was not high enough to give information on clusters.

4 Application of the DBSCAN algorithm to particle-laden turbulence

Clustering algorithms belong to the class of unsupervised learning algorithms. There are different clustering algorithms among which : hierarchical clustering, K-means, and DBSCAN [10]. In this part of our study, the objective is to apply the DBSCAN algorithm to particle-laden turbulence in order to determine the number of clusters, percentage of noise points, cluster dimensions among others for each image grouping droplets scattered in a two-dimensional space and regroup them according to the values taken by Stokes number for each Reynolds number to be set *a priori*. Indeed, DBSCAN is a good alternative to k-means when we do not know how many clusters to expect in our data, but we know something about how points should be clustered in terms of density (distance between points in a cluster). The DBSCAN algorithm integrates a notion of cluster based on density makes it possible to discover clusters of arbitrary form [9]. Morever, the choice of this algorithm was also made from a practical point of view because the parameters which are necessary for it on the input side can be related to physical parameters for example distance between two points can be assimilated by their distances in space or in the plane depending on whether the data is represented in a two or three-dimensional space.

4.1 DBSCAN algorithm

4.1.1 Interest of DBSCAN

The method of grouping a set of spatial objects into groups called «clusters» is known as geospatial clustering. Many applications require spatial data management such as SDBS (Spatial database systems). An increasing amount of data is obtained from satellite images, X-ray crystallography or other automatic equipment. Thus, automatic overdrafts more and more knowledge is needed in spatial databases. Well-known clustering algorithms offer no solution for the combination of these requirements [15]. This is how the new clustering algorithm called DBSCAN appeared. DBSCAN (spatial clustering based on the density of applications with noise) algorithm allows the identification of classes, i.e. the grouping of the objects of a database into meaningful subclasses. Otherwise, many clustering algorithms do not allow solving these problems. DBSCAN which integrates a notion of cluster based on density makes it possible to discover clusters of arbitrary form. This algorithm requires only two input parameters so that the user can specify an appropriate value and proves to be a particularly efficient even for large spaces database.

4.1.2 Cluster's notion based on density

Hierarchical clustering and partitioning methods like K-means are very good for finding spherical or convex shaped clusters. However, they are also greatly affected by the presence of noise and outliers in the data. The goal of DBSACAN is to identify dense regions, which can be measured by the number of objects close to a given point [16].

To better understand this notion of density-based clustering, it is necessary to define these following terms [16] :

- **Direct density reachable :** a point M is directly density reachable from another point N if M is in the ϵ-neighborhood of N and N is a core point.
- **Density reachable :** a point M is density reachable from N if there are a set of core points leading from N to M.
- **Density connected :** Two points M and N are density connected if there are a core point P, such that both M and N are density reachable from P.
- **Core points :** Core data points have at least $minPts$ number of data points within their epsilon distance.
- **Border points :** Border data points are on the outskirts as they are in the neighborhood (ie. in epsilon distance of core point) but have less than the required $minPts$.
- **Outlier points :** These points are not part of a neighborhood (ie. more than epsilon distance) and are not border points. These are points located in low-density areas.

A density-based cluster is defined as a group of density connected points.

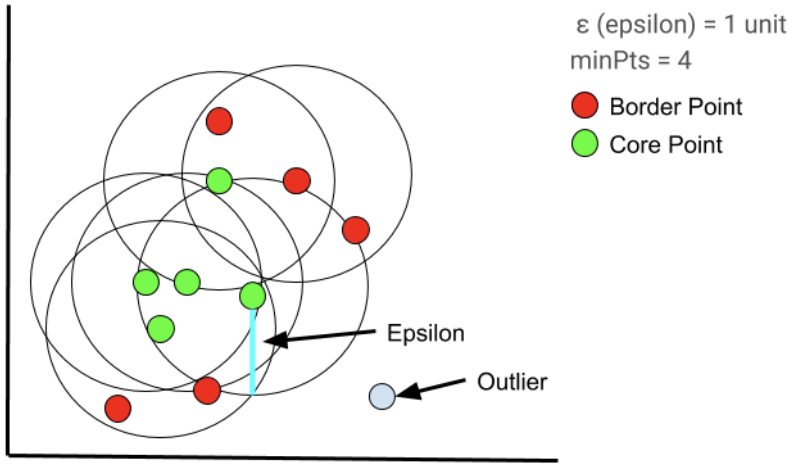


Figure 4.1: DBSCAN demo

Source : towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31

4.1.3 Identifying DBSCAN parameters

DBSCAN is an unsupervised algorithm that requires two hyper-parameters : ϵ (*eps*: epsilon) and the minimum number of points required to form a cluster ($minPts$). Ideally, we must know the appropriate *eps* and $minPts$ of each cluster and at least one point of the respective cluster. The parameter *eps* is the maximum radius of the neighborhood. It is generally determined by the problem to solve (e.g. a physical distance) and $minPts$ is the desired minimum cluster size [17]. By denoting d the distance from a point M to its $k-th$ nearest neighbor, then the d-neighborhood of M contains exactly $k+1$ points for almost all points M. It contains more than $k+1$ points if and only if several points have exactly the same distance d from M, which is unlikely. In addition, changing the value of k for a point in a given cluster will not result in large changes in the value of d. This only happens when the $k-th$ nearest neighbor of M for $k = 1, 2, 3 \ldots$ are located approximately on a straight line which is generally not obvious for a point in a cluster [15].

Then the optimal value for ϵ can be chosen by using a k-distance graph, plotting the distance (on the y-axis) to the $k = minPts$, by mapping each point (all the data points in the x-axis) to the distance of its $k-th$ nearest neighbor. By sorting the points of the database in ascending order of their $k-dist$ values, the graph obtained gives an idea of the density distribution in the database. Experiments in [15] have shown that k-dist graphs for $k > 4$ do not differ significantly from $4-dist$ graph and, moreover, they require much more calculations. Therefore, we can set the $minPts$ parameter to 4 for all datasets (two-dimensional data). Subsequently, optimal *eps* value (corresponds to critical change in the curve or «Elbow Method ») is obtained such that a «sufficiently large »part of the points have a distance to its nearest neighbor less than *eps* («sufficiently large »means, for example, 95% or 99% of the points).

Much like the «Elbow Method »used to determine the optimal epsilon value the $minPts$ heuristic isn't correct 100% of the time [18]. Usually the $minPts$ can be derived from the number of D dimensions in the dataset, such as $minPts \geq D+1$ (or $minPts = 2*D$) [17]. With $minPts \leq 2$, the result will be the same as for hierarchical grouping with the single link metric, with the dendrogram cut at height ϵ. Therefore, $minPts$ should be chosen at least 3. Also, for datasets with noise, higher values of $minPts$ can be chosen and will result in more meaningful clusters.

4.1.4 Abstract algorithm

The implementation of the DBSCAN algorithm requires several steps which can be summarized through the following pseudo algorithm :

1. Find the points in the ϵ (*eps*) neighborhood of every point, and identify the core points with more than $minPts$ neighbors.

2. Find the connected components of core points on the neighbor graph, ignoring all non-core points

3. Assign each non-core point to a nearby cluster if the cluster is an ϵ (*eps*) neighbor, otherwise assign it to noise

The DBSCAN algorithm can be used with any distance function (similarity functions or other predicates). The distance function (dist) can therefore be considered as an additional parameter. A pseudocode of the algorithm can be given as follows :

```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N := RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) := Noise                               /* Label as Noise */
            continue
        }
        C := C + 1                                          /* next cluster label */
        label(P) := C                                       /* Label initial point */
        SeedSet S := N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point Q */
            if label(Q) = Noise then label(Q) := C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed (e.g., border point) */
            label(Q) := C                                   /* Label neighbor */
            Neighbors N := RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check (if Q is a core point) */
                S := S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

Figure 4.2: DBSCAN pseudo-algorithm

Source : wikipedia.org/wiki/DBSCAN#Algorithm

4.1.5 Advantages and disadvantages of DBSCAN

1. **Advantages [17]**
    - Requires just two parameters
    - Does not require *a priori* specification the number of clusters like KMeans
    - Able to identify noise data while clustering and is robust to outliers
    - Able to find arbitrarily size and arbitrarily shaped clusters
    - Can separate high density data into small clusters
    - Can cluster non-linear relationships (finds arbitrary shapes)
    - The parameters $minPts$ and ϵ can be set by a domain expert, if the data are well understood
2. **Disadvantages [17]**
    - Very sensitive to epsilon (ϵ) and minimum points ($minPts$) parameters
    - Can suffer with high dimensional data
    - Struggles to identify clusters within data of varying density (cannot cluster data sets well with large differences in densities)

4.2 Clustering results

In this study, one of the objectives that we set ourselves is to use the DBSCAN algorithm in order to find clusters of arbitrary shape based on the notion of dense regions for all data files. As said previously, we have set the minimum number of points ($minPts$) equal to 5 and the value of eps corresponding to its optimal value already presented. We have done this for every dataset. In addition, we used the average of the optimal values at the different times (corresponding to the 10 files for a given Reynolds number and a given Stokes number), as being the optimal eps value corresponding to the fixed Reynolds and Stokes parameters.

4.2.1 Optimal epsilon value and minPts

The choice of the epsilon parameter is very important to perform the DBSCAN algorithm. We are based on the approach mentioned above in order to make an optimal choice. Take the example for a Reynolds number (Re) equal to 909 and a Stokes number equal to 0.1 at the a given time instant. By slicing the line so as to have approximately 95% of the points which are at a distance from their nearest neighbors less than eps, it is thus deduced that the value eps is equal to approximately 0.029. The approach method is applied in all the other datasets available to us.

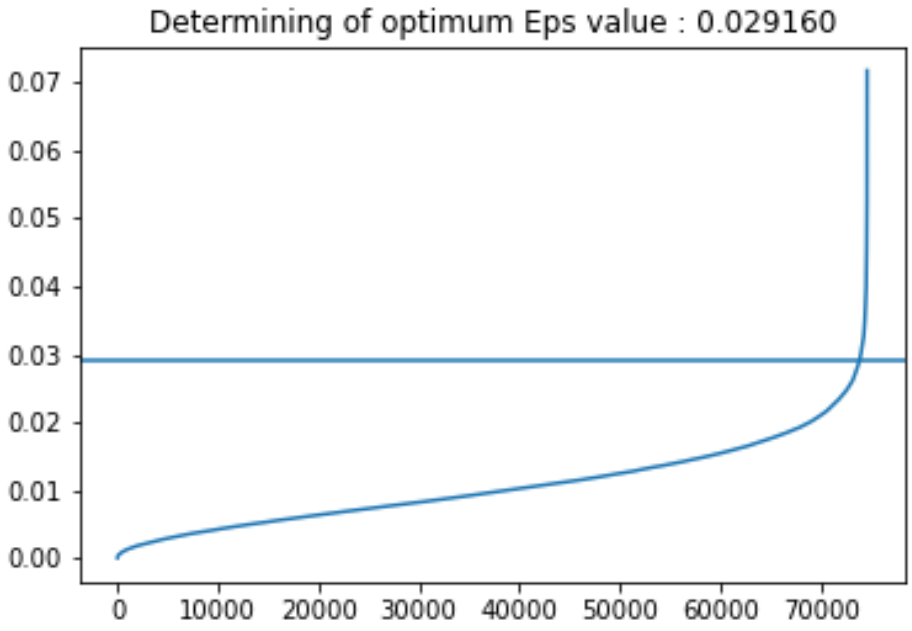


Figure 4.3: Optimal eps value for $Re = 909$ and $St = 0.1$ at the first instant

One way to choose the $minPts$ would be to multiply by two the number of dimensions of our dataset, i.e. 4. However, we arbitrarily set it to 5 despite the limited power of our machines to perform these calculations.

In order to refine our analysis according Stokes number (St) and Reynolds number

(Re), we calculated the average value of the optimal eps because, for each Re considered, we have 10 files (.pos) corresponding to 10 different positions of the droplets for each Stokes number for a fixed Re (909 and 2220).

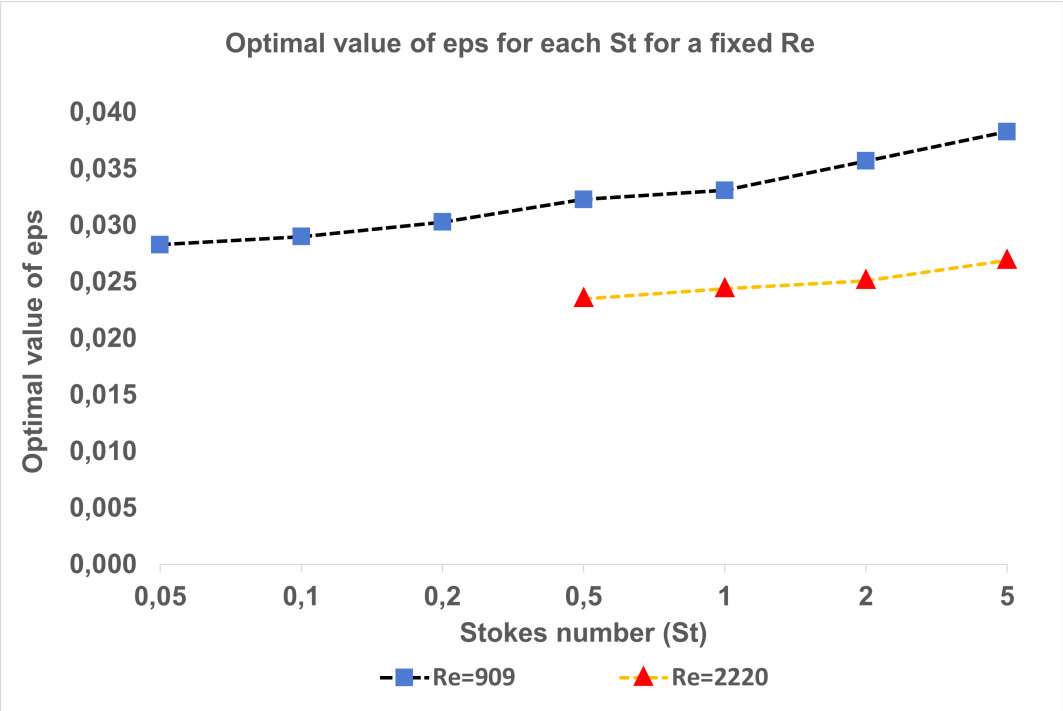


Figure 4.4: Optimal eps value for each Stokes number for a given Reynolds number

Analysis of the graph above shows an increasing and a quasi-linear relationship between the Stokes number and the optimal eps value for Re equal to 909. In addition, these values fluctuate around 0.03 with an amplitude less than 0.001 compared to confidence intervals (see appendix). Also, the analysis of the boxplots of the series of the optimal values of eps for each Stokes number, does not detect any outliers within two significant ones. Otherwise, the observation is almost the same for $Re = 2220$. The only significant difference is that the values of optimal eps turn around 0.02 with almost the same amplitude of variation. Also, theirs boxplot not only do not reveal the existence of atypical points but also shows a good distribution of observations. Thus, we can consider that this average constitutes a good indicator of the optimal ep value on the set of files considered.

4.2.2 Number of clusters

Observation of the graph below shows that the number of clusters is much higher when the Reynolds number is larger for any Stokes number. This suggests that the higher the Reynolds number, the larger the number of clusters.

For $Re = 909$, the average number of clusters decreases to reach its minimum value for $St = 5$. The decrease is much stronger for a Stokes number between 0.05 and 0.2. Beyond this value, the number of clusters decreases slightly. We also obtain similar

results for $Re = 2220$ with the only difference that the rate of decrease is approximately constant.

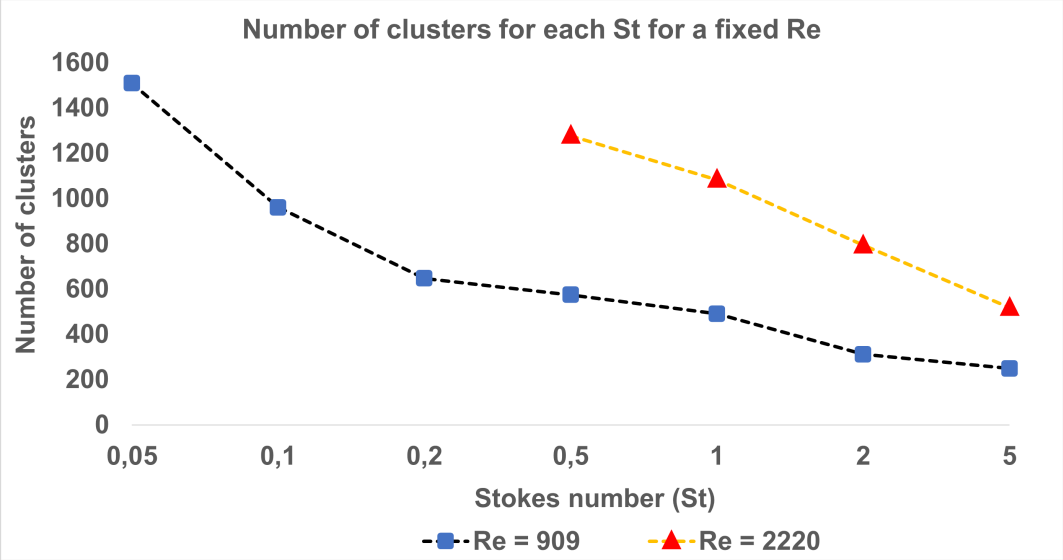


Figure 4.5: Number of clusters for each Stokes number for a given Reynolds number

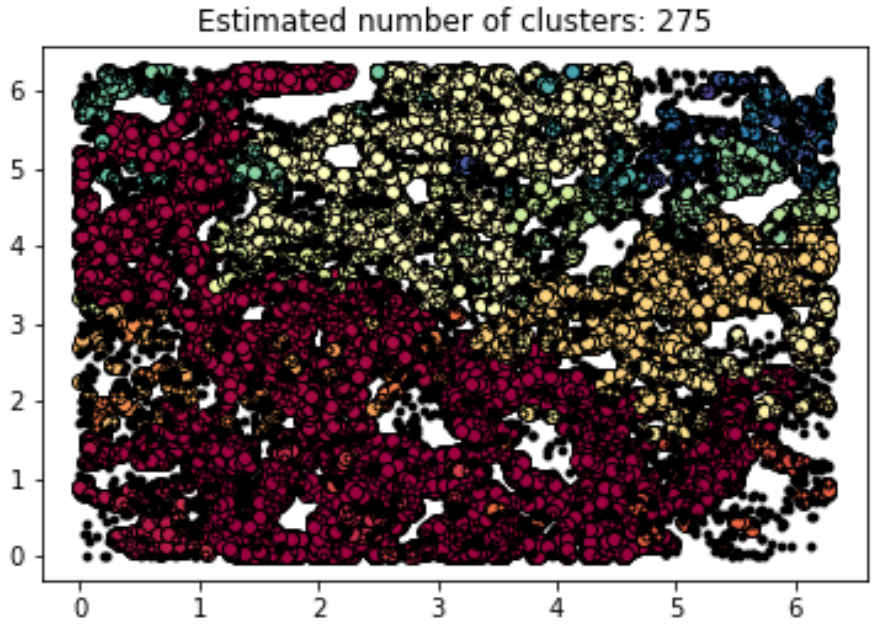


Figure 4.6: Representation of colorized clusters in a sample flow with $Re = 909$ and $St = 2$

4.2.3 Percentage of noise

As its name suggests, the DBSCAN algorithm is closely linked with the notion of noise. These noise points are points that do not belong to any of the clusters. The

percentage of noise is equal to the ratio of the number of noise points and the total number of points in the dataset.

We can notice that the percentage of noise points is slightly higher for $Re = 2220$ than for $Re = 909$. In both cases, the percentage decreases with the Stokes number. On the other hand, the decrease is faster between $St = 0.05$ and $St = 0.5$ for $Re = 909$ while the decrease is very weak for $Re = 2220$.

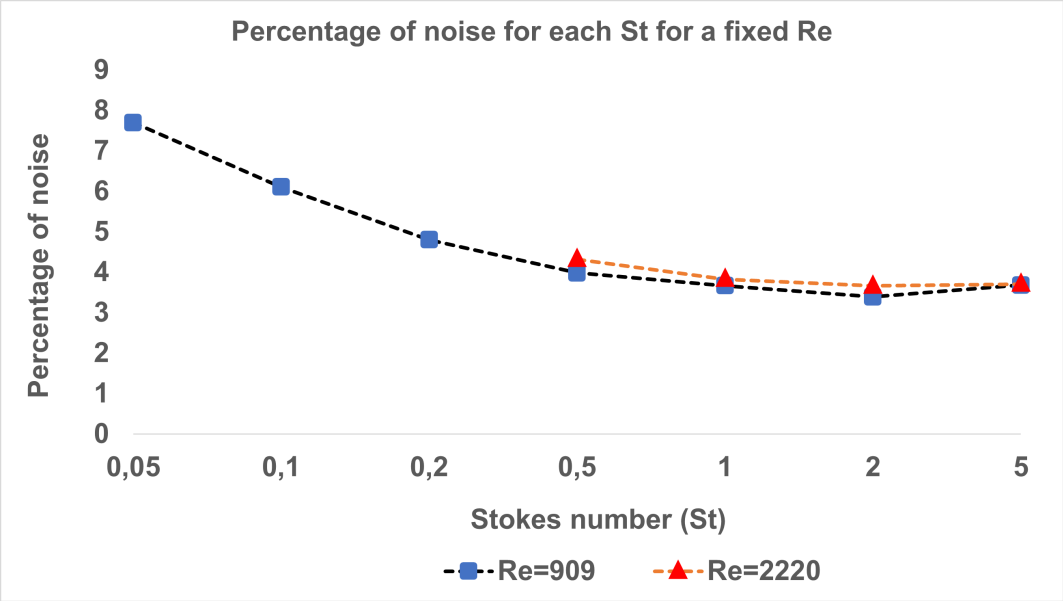


Figure 4.7: Percentage of noise for each Stokes number for a given Reynolds number

4.2.4 Cluster dimensions

To calculate the dimensions of the clusters we will try to find the circular or elliptical shape that covers the majority of the particles belonging to each cluster. Once the shapes are identified, we calculate their surfaces by determining the diameters of the ellipse (the major and minor axes) -or the diameter of the circle in the case of a circular shape-.

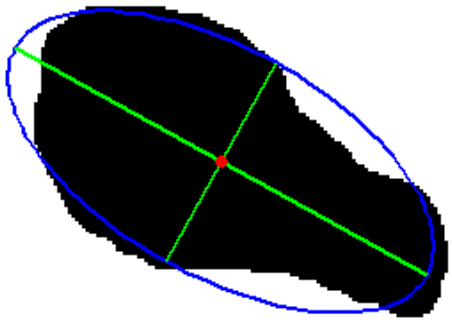
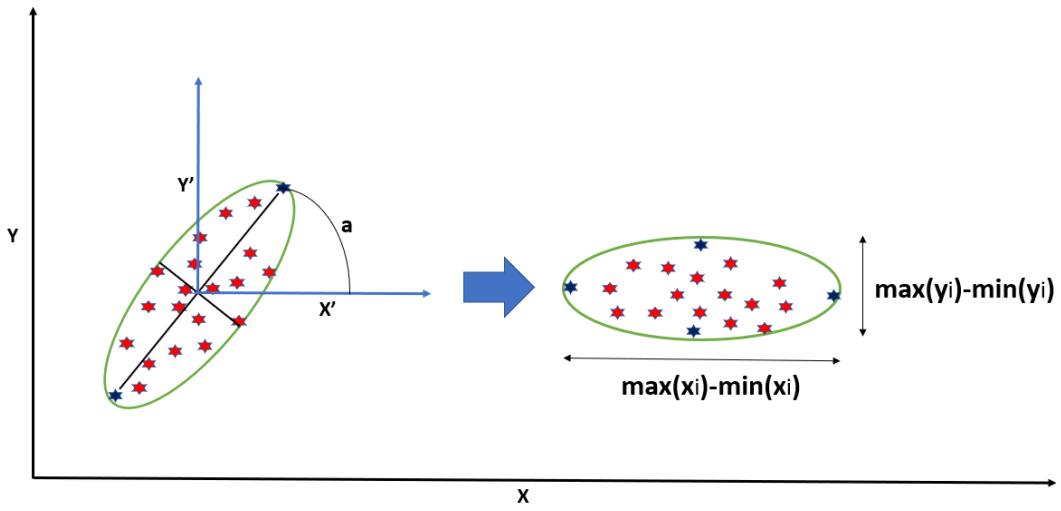


Figure 4.8: example of a cluster surrounded by an ellipse

Determining diameters is not always obvious. In the figure below, we notice that we can have an oblique ellipse, so to calculate the diameters we apply a change of reference frame - so that the center of the ellipse and that of the new reference frame are superimposed - afterward we rotate the ellipse to have an ellipse that forms a plane of axes parallel to the axes of reference, then the surface of the ellipse is given by :

$$S = \pi ab \quad with\ a = \frac{max(x_i) - min(x_i)}{2}\ and\ b = \frac{max(y_i) - min(y_i)}{2}.$$



Here is an example of a sample with the characteristics $Re = 909$ and $St = 0.05$ which shows a different clusters colorized for a time instant:

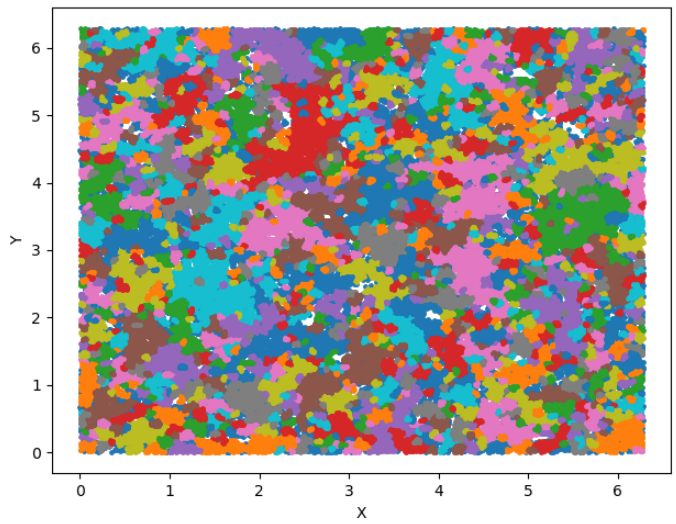


Figure 4.9: Representation of colorized clusters in one sample

Influence of the Stokes number and the Reynolds number on the dimension of clusters :
Since our data base contains 10 files (.pos) corresponding to 10 different time instant, we decided to compute the average value of the dimension of clusters for each St and Re, in this way it will be easier to study the Reynolds number variance according to the dimension of clusters.

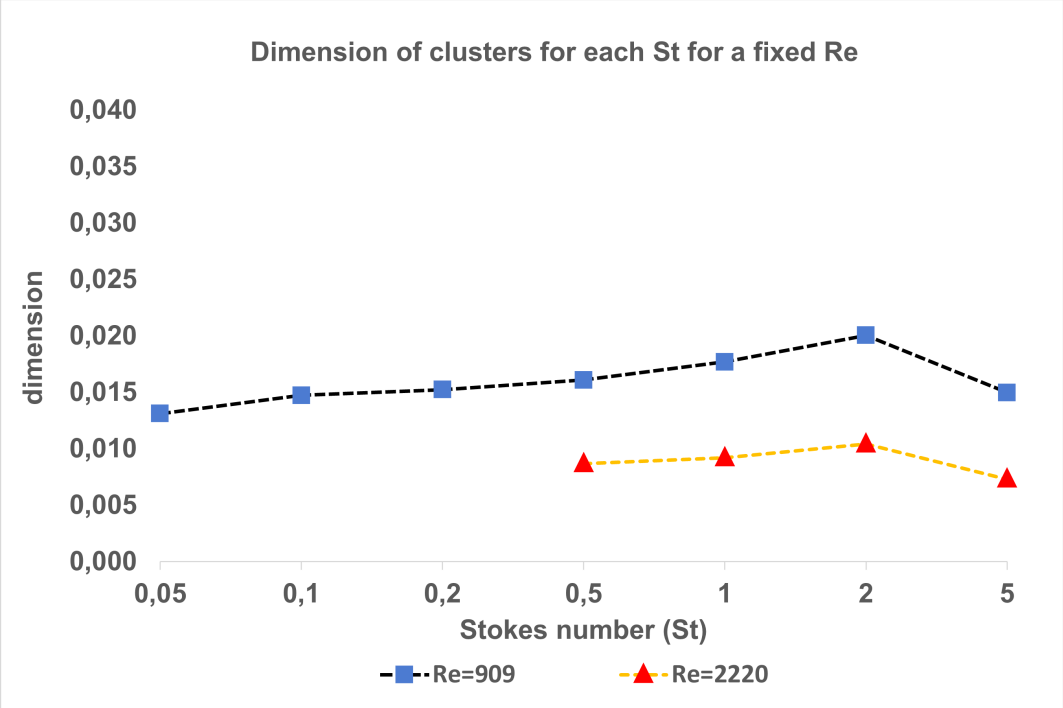


Figure 4.10: Dimension of clusters for each Stokes number for a fixed Reynolds number

The First thing we can observe in the graph above is that, when it comes to comparing the results obtained for each Reynolds number, the dimensions of the clusters of the data characterized by $Re = 909$ are larger than those of $Re = 2220$, so the clusters in the first case have more surface than the ones in the second case for a fixed Stokes number, as well as for all the Stokes numbers combined. Also, the largest mean surface of clusters is reached when $St = 2$ for the two Reynolds numbers, which is coherent with the previous result, since the noise points in case $St = 2$ are the lowest for both Reynolds numbers in comparison to the remaining Stokes numbers for each Reynolds number taken separately.

4.2.5 DBSCAN Cluster Evaluation : Silhouette method

As we have shown previously, this method measures the separability between clusters. The silhouette coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The silhouette coefficient for a sample is $\frac{(b-a)}{max(a,b)}$. It varies between -1 (worst classification) and 1 (best classification). [19]

We will therefore wish to have a high score (i.e. closest to 1) which would indicate

that there is a low intra-cluster average distance (tight clusters) and a large mean inter-cluster distance (well clusters separated).

This is how we apply this method on datasets in order to do the analysis according to the Stokes number and the Reynolds number. In other words, for a fixed Reynolds number, we would like to know how the silhouette coefficient varies according to the Stokes number. Therefore, after checking the assumptions on representativeness, we considered the average of the silhouette coefficients in each dataset for each given Stokes number and Reynolds number fixed *a prior*.

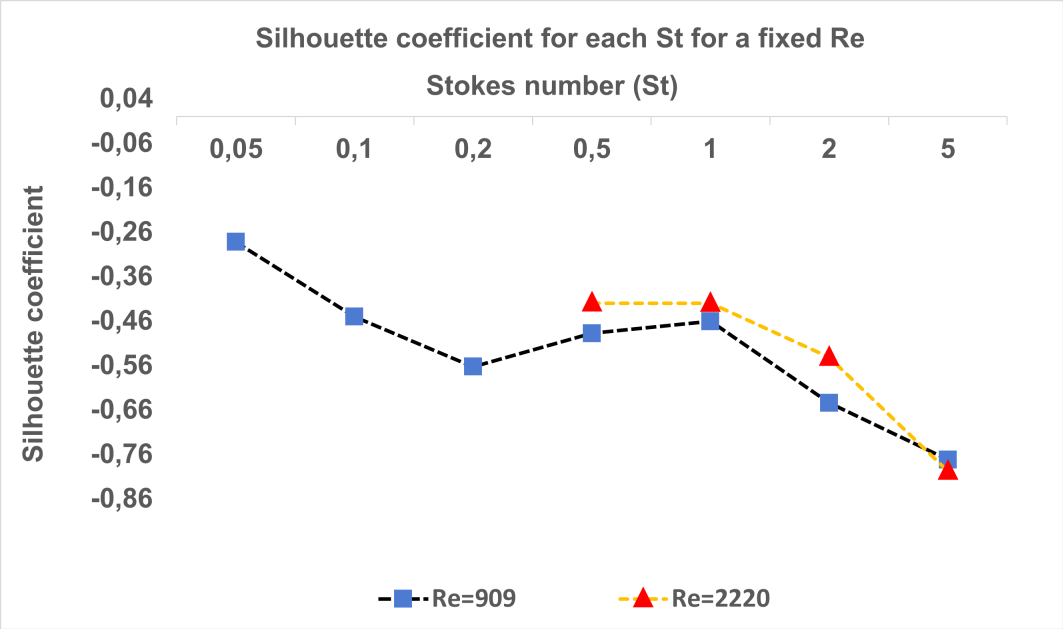


Figure 4.11: Silhouette coefficient for each Stokes number for a given Reynolds number

The first observation we can make on this graph is that all the silhouette coefficients are negative. They are also well above -1 (coefficients between -0.8 and -0.26). Thus, we can say through these results that we would have a good clustering all the more if many points belonging to given clusters were part of other neighboring clusters. In other words, the clutering applied may not be the best given the concepts of cluster separation and homogeneity.

Nevertheless, if we make the analysis according to the Stokes number and the Reynolds number, we note that the higher Stokes number, the lower the silhouette coefficient is for $Re = 2220$. While for $Re = 909$, the silhouette coefficient decreases for a Stokes number between 0.05 and 0.2, then increases between 0.2 and 1 ends up decreasing again between 1 and 5.

5 Application of Tensorflow to particle-laden turbulence

Tensorflow (TF) is an open source library developed by Google that enables the user to perform diverse machine learning tasks, it is often associated with the framework Keras which is an Application Programming Interface (API) that permits to implement the complex functions of TF in a way that is more accommodating for the user. One of the most popular uses of Tensorflow is to carry out some image classification tasks, this is usually done by implementing a Convolutional neural network (CNN) architecture.

In this section, our goal is to build a model that allows us to classify an image given as an input into its accordingly Stokes and Reynolds numbers. Furthermore, we try to generate synthetic data from the original files that contains the position of the droplets at a point of time, this will be done using Generative Adversarial Networks (GAN).

We'll attempt to achieve our two objectives by building multiple neural networks with the help of Tensorflow and the high level integrated API of Keras.

5.1 About Image Classification

5.1.1 Definition

In a generic definition, image classification is the process that takes an image as an input and outputs the predefined «classes» or the «labels» to which this particular image belongs to, it is used in medical data analysis for instance. This process requires to extract the features of the image (points, pixels, edges ...) in order to observe some patterns, we use the CNN architecture to conduct this task.

5.1.2 Convolutional Networks

«A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take as the input an image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.[22]»

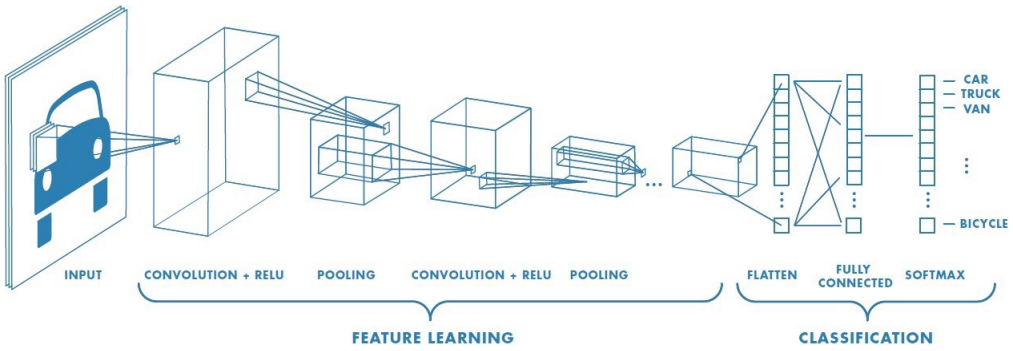


Figure 5.1: CNN Architecture

Source : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

The process of extracting the features of the input image is accomplished according to the following steps :

- **1)The convolutional layer:** We select a matrix called the filter that moves along the matrix that contains the pixel values of our input image, we obtain in the end feature maps in the form of a matrix that is smaller in shape than the input matrix.

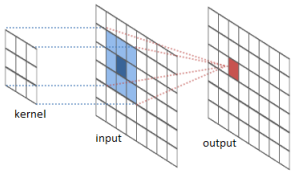


Figure 5.2: Image convolution with a filter

Source : https://jeanvitor.com/convolution-parallel-algorithm-python//

- **2) Relu (Activation function) :** A non linear function that activates our newly obtained matrix by increasing the non-linearity.

- **3) Pooling layer :** Reduces the dimension of the feature maps and prevents overfitting by saving only the relevant parts of the image. Later on, we'll use the Max Pooling layer that outputs the maximum value from the portion of image covered by the filter.

- **4) Flattening layer :** The last layer of the CNN, stores the values in a long 1-dimensional vector.

5.2 Classifying our Data

5.2.1 Data pre-processing and parameterization

We have initially a total of 110 square images of our droplets and 7 classes that correspond to the 7 values of Stokes Number that characterize the droplets of our dataset. We put the images in separate files according to their classes.

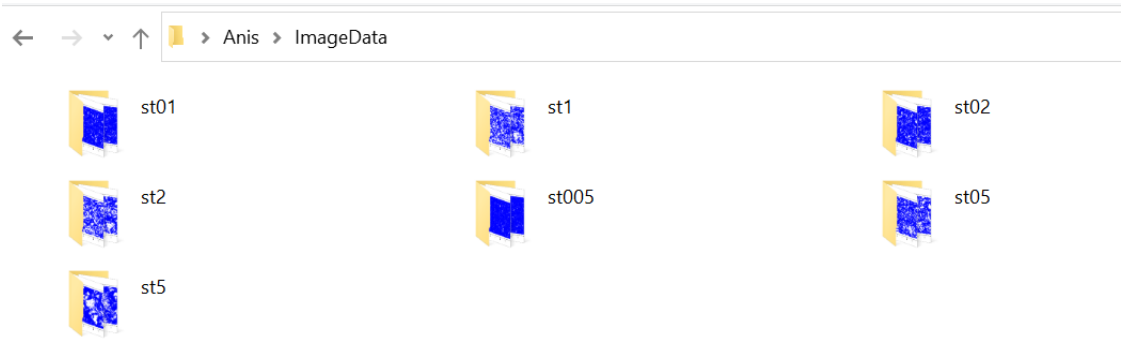


Figure 5.3: The classes of our dataset

We arbitrarily chose to withdraw the image «N512_st005pcl-021000_z-4eta.png» representing a droplet's position with the Stokes and Reynolds numbers respectively equal to 0.05 and 909 in order to use it for a prediction test after we have trained the model. We set the batch size at 16 and choose to use 70% of the images for training and 30% for validation.

```
Found 109 files belonging to 7 classes.
Using 77 files for training.
Found 109 files belonging to 7 classes.
Using 32 files for validation.
```

Figure 5.4: Partition of our dataset

5.2.2 Data augmentation

Given the small dataset available, it was preferable to expand it by using techniques of Data Augmentation so as to increase the performances and accuracy of our model but also to avoid the issues concerning overfitting. In particular we've decided to use the layer «tf.keras.layers.experimental.preprocessing.RandomFlip» that flips each image horizontally and vertically.

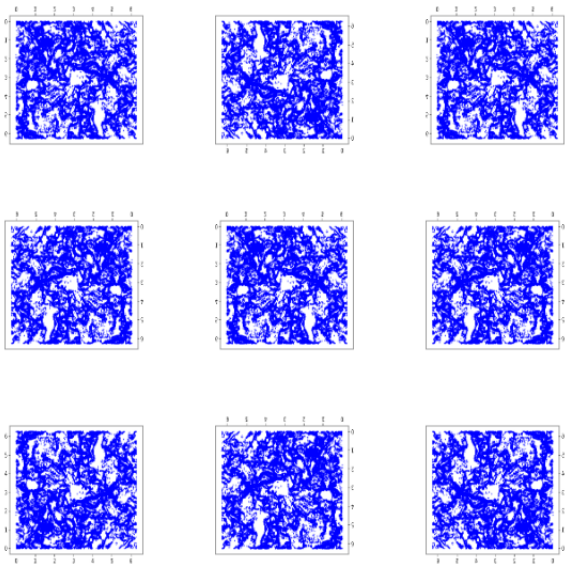


Figure 5.5: Data augmentation applied to the same image multiple times

5.2.3 Building the model

After going through these preliminary steps, it is time to build an architecture for our CNN. We adopted a model composed of 3 convolutional layers between pooling layers of the Maxpool type, finally we attach an activation layer at the end followed by a flatten layer to convert the output into a one dimensional vector.

Layer (type)	Output Shape	Param #
rescaling_107 (Rescaling)	(None, 180, 180, 3)	0
conv2d_159 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_159 (MaxPoolin	(None, 90, 90, 16)	0
conv2d_160 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_160 (MaxPoolin	(None, 45, 45, 32)	0
conv2d_161 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_161 (MaxPoolin	(None, 22, 22, 64)	0
flatten_53 (Flatten)	(None, 30976)	0
dense_105 (Dense)	(None, 128)	3965056
dense_106 (Dense)	(None, 7)	903

Total params: 3,989,543
Trainable params: 3,989,543
Non-trainable params: 0

Figure 5.6: Architecture of our model

5.2.4 Training and visualising the model

At first we realize our training with a large epoch to determine where the overfitting happens, we first set it at 100 :

Figure 5.7: Training of our model after 100 epochs

The plot shows the training loss function and validation loss function down trending together until around 13 epochs. The validation loss function then starts abruptly increasing while the training loss function is still steadily decreasing. We conclude that the model starts overfitting from there on. After that point, the model efficiency begins to weaken. It is therefore primordial to put an early stoppage before we pass that point. We re-train out data with the number of epochs set at 13.

Figure 5.8: Training of our model with epoch=13

The model is still not the most optimal, we observe signs of overfitting (the validation accuracy is on certain epochs above the training accuracy, sudden increase of the validation error on the 7th epoch). However, the validation loss is even with the training loss, steadily decreasing and they're both seemingly converging to 0. We also note that there isn't a big margin between the training accuracy and validation accuracy. At the end of the training we obtain an accuracy of 79%.

5.2.5 Prediction

We can now use the final model to predict the class of an untrained image, such as the N512_st005pcl-01000_z-4eta.png that we have spared from training. Not only we obtained the class or the Stokes number (0.05 in this case, the model got it right and the prediction was correct), we also managed to obtain the correctness of this prediction known as the confidence which was valued at 90.40%.

```
...: print(
...:     "This image most likely belongs to {} with a {:.2f} percent confidence."
...:     .format(class_names[np.argmax(score)], 100 * np.max(score))
...: )
This image most likely belongs to st005 with a 90.40 percent confidence.
```

Figure 5.9: Prediction of the class of our image

5.2.6 Classifying by the Stokes and Reynolds number

Similarly to what we did before, we have applied the same algorithm while considering this time that we have 12 classes (or labels) instead of 7. We take into account in the class both the Stokes number and the Reynolds number.

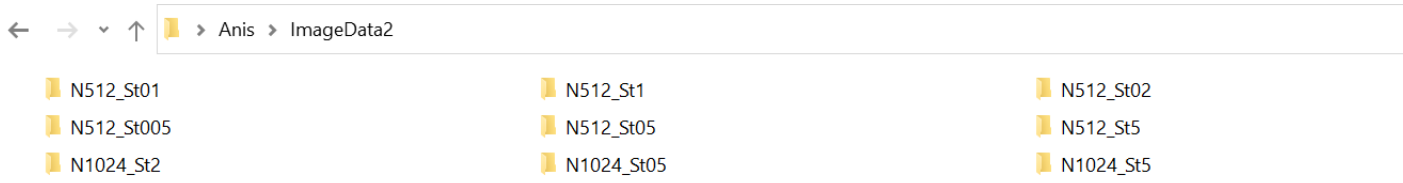


Figure 5.10: The classes of our dataset

We put aside the image «N512_st5pcl-028000_z-4eta.png» and we train the rest of the images, after executing the algorithm we obtain a correct prediction once again but with a lower confidence estimated at 57.20%.

```
...: print(
...:     "This image most likely belongs to {} with a {:.2f} percent confidence."
...:     .format(class_names[np.argmax(score)], 100 * np.max(score))
...: )
This image most likely belongs to N512_St5 with a 57.20 percent confidence.
```

Figure 5.11: Prediction of the class of our image

5.3 Generating Synthetic Data

With the intention of ameliorating our machine learning model for image classification, we've resorted to generating synthetic data for the purpose of data augmentation to have a larger training set. There are different deep-neural network algorithms well suited for this task, we propose to use one of them known as the generative adversarial networks (GANs), which are a type of neural network that lies under the field of unsupervised training we'll implement with Tensorflow and Keras.

5.3.1 Definitions

What is Synthetic Data?

Synthetic data is «any production data applicable to a given situation that are not obtained by direct measurement»[23]. It is created with the help of algorithms in a way it mirrors the statistical properties of an original data and reproduces the distribution with the main goal of producing realistic replicates. It is widely used as an anonymization technique to preserve the privacy, it can also be very useful to solve data scarcity like in our case.

Generative adversarial network

GAN is a machine learning framework based on unsupervised learning that is able to generate new data that mimics the original data and saves it structure. GANs are composed of two neural networks that work in an adversarial style :

- **The generator** the model that produces the new data that preserves the statistical characteristics of the original one.
- **The discriminator** the model that distinguishes between real and imitated data.

How it works : «The generative network generates candidates while the discriminative network evaluates them. The contest operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data distribution of interest, while the discriminative network distinguishes candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminative network... Independent backpropagation procedures are applied to both networks so that the generator produces better samples, while the discriminator becomes more skilled at flagging synthetic samples» [24]

5.3.2 Building and Training our GAN model

We're willing to generate synthetic data from the txt file corresponding to the particle position at a given time for St=5 and Re = 1024, the txt file is named «pcl-0150000_z-4eta.txt».

First of all we set a standard batch size of 32. The generator function takes some random noise as an input to produce the images. The architecture of this function is represented by an input layer that characterize the noise and 4 activation layers with 128,256,512,3 neurons .

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(32, 32)]                0
_________________________________________________________________
dense_24 (Dense)             (32, 128)                 4224
_________________________________________________________________
dense_25 (Dense)             (32, 256)                 33024
_________________________________________________________________
dense_26 (Dense)             (32, 512)                 131584
_________________________________________________________________
dense_27 (Dense)             (32, 3)                   1539
=================================================================
Total params: 170,371
Trainable params: 170,371
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.12: Architecture of the Generator

As for the discriminator, the architecture defined by an input layer, 4 activation layers with 512,256,,128,1 neurons with the last layer having a Sigmaoid activation instead of relu, eventually we add 2 dropout layers to prevent overfitting.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(32, 32)]                0
_________________________________________________________________
dense_24 (Dense)             (32, 128)                 4224
_________________________________________________________________
dense_25 (Dense)             (32, 256)                 33024
_________________________________________________________________
dense_26 (Dense)             (32, 512)                 131584
_________________________________________________________________
dense_27 (Dense)             (32, 3)                   1539
=================================================================
Total params: 170,371
Trainable params: 170,371
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.13: Architecture of the Discriminator

The architecture we've built was inspired from a paper about fraud detection in banking using GANs [25]

After establishing the architecture, we train the GAN model up to 5000 epochs.

```
4980 [D loss: 0.536116, acc.: 73.44%] [G loss: 1.127634]
4981 [D loss: 0.475874, acc.: 73.44%] [G loss: 0.954669]
4982 [D loss: 0.766204, acc.: 65.62%] [G loss: 1.047146]
4983 [D loss: 0.522794, acc.: 65.62%] [G loss: 1.283856]
4984 [D loss: 0.655954, acc.: 67.19%] [G loss: 0.951234]
4985 [D loss: 0.572167, acc.: 67.19%] [G loss: 1.139402]
4986 [D loss: 0.479146, acc.: 76.56%] [G loss: 1.067703]
4987 [D loss: 0.538064, acc.: 70.31%] [G loss: 1.237918]
4988 [D loss: 0.504940, acc.: 76.56%] [G loss: 1.223475]
4989 [D loss: 0.517938, acc.: 75.00%] [G loss: 1.076315]
4990 [D loss: 0.548773, acc.: 76.56%] [G loss: 1.294502]
4991 [D loss: 0.552965, acc.: 67.19%] [G loss: 1.172060]
4992 [D loss: 0.497968, acc.: 71.88%] [G loss: 1.108571]
4993 [D loss: 0.507878, acc.: 68.75%] [G loss: 1.133773]
4994 [D loss: 0.498458, acc.: 68.75%] [G loss: 1.137892]
4995 [D loss: 0.492085, acc.: 71.88%] [G loss: 1.115393]
4996 [D loss: 0.577287, acc.: 65.62%] [G loss: 1.032875]
4997 [D loss: 0.476150, acc.: 76.56%] [G loss: 1.107633]
4998 [D loss: 0.570038, acc.: 68.75%] [G loss: 1.197875]
4999 [D loss: 0.484288, acc.: 73.44%] [G loss: 1.058782]
5000 [D loss: 0.572047, acc.: 67.19%] [G loss: 1.159346]
generated_data
```

Figure 5.14: Training of our GAN model

The model seems to have stabilized and we obtain an accuracy of about 70% on average after 5000 epochs.

5.3.3 Evaluating the quality of our synthetic data

We save the synthetic data our generator has published on a csv file

Generated Data

	0	1	2
0	5.5359	4.646327	0.024666833
1	2.4212472	2.9091396	-0.009225523
2	5.8401623	6.961732	-0.014788685
3	5.627727	3.7431374	-0.08713798
4	3.786219	2.9356134	-0.015868066
5	2.795875	3.6795409	0.017181108
6	3.8900506	3.4232237	-0.0020933207
7	3.3250597	4.5564632	0.0143206585
8	3.0746832	4.266151	0.008310655
9	3.5071926	3.9229038	-0.001823647
10	3.8599782	2.7320428	-0.010307303
11	3.6781526	3.3689187	0.01881116
12	4.0671496	4.101487	-0.006715907
13	5.2010036	3.1669717	-0.073041946
14	4.2579513	2.5951412	-0.07483977

Figure 5.15: The head of our New data, contains 135000 line in total

Aiming to compare the quality of our generated data, we'll use for the rest of this section the package table-evaluator on python [26] to have indications about how close we are to the actual data.

We compare the mean and the standard error between the generated (on the y-axis) and original data (on the x-axis)

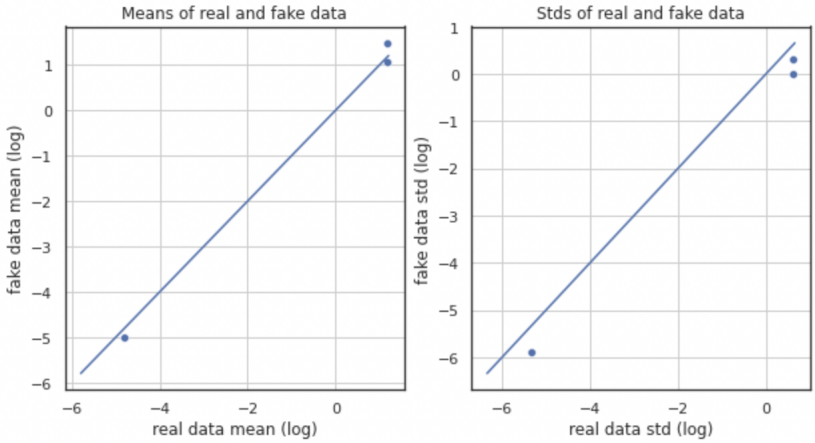


Figure 5.16: Mean and SD of our data

We observe a straight line. The equation that characterizes this line is the trivial

$$x = y \tag{5.1}$$

On both the mean and the standard error, the data we generated has successfully retained these two statistical properties of the original data.

However this is not enough, we should look up thoroughly other details, beginning by the cumulative sum of the position of the particles (on the x, y and z axis)

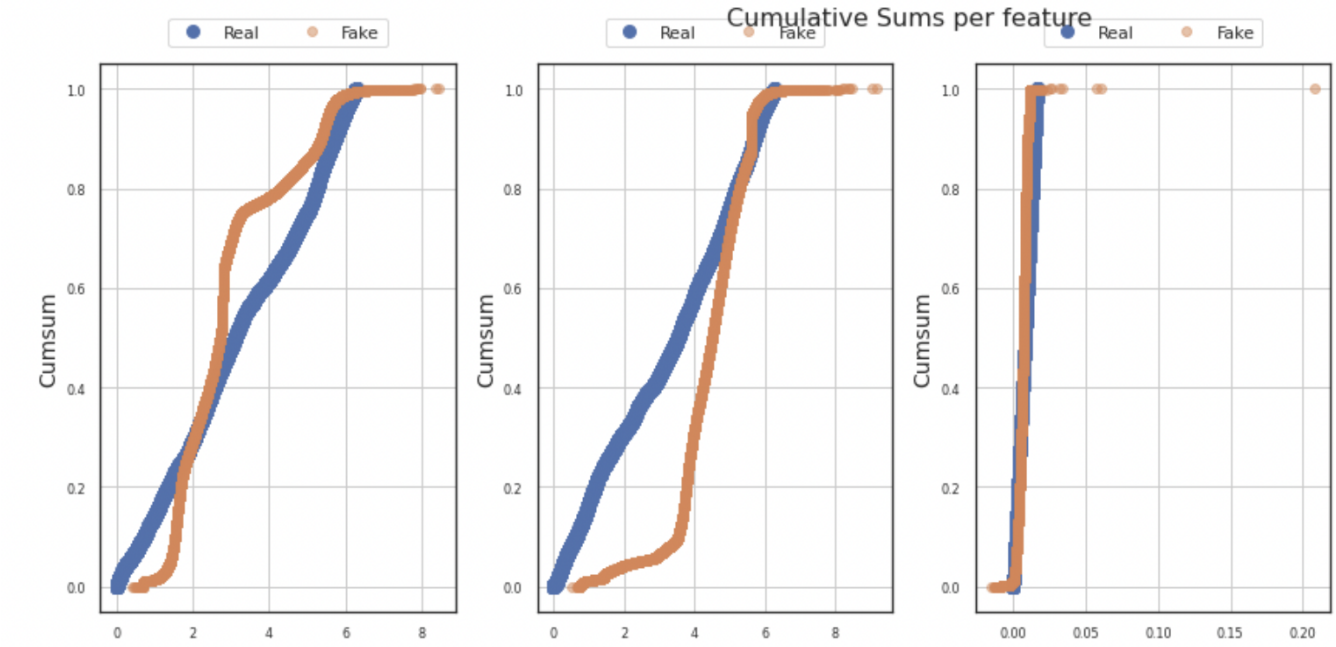


Figure 5.17: Cumulative sum of our data

The repartition of particles in the z-axis compared to the original one is decent. This is unfortunately not the case for the x and y axis in which we note many irregularities,

especially regarding the small values. We do also register that the position of the particles in the x and y axis can take values way superior than in the original data. This newly generated data doesn't seem to recreate the original data exactly.

Finally we have a first two component analysis to have a better overview of the positioning of the 135 000 particles of our generated data

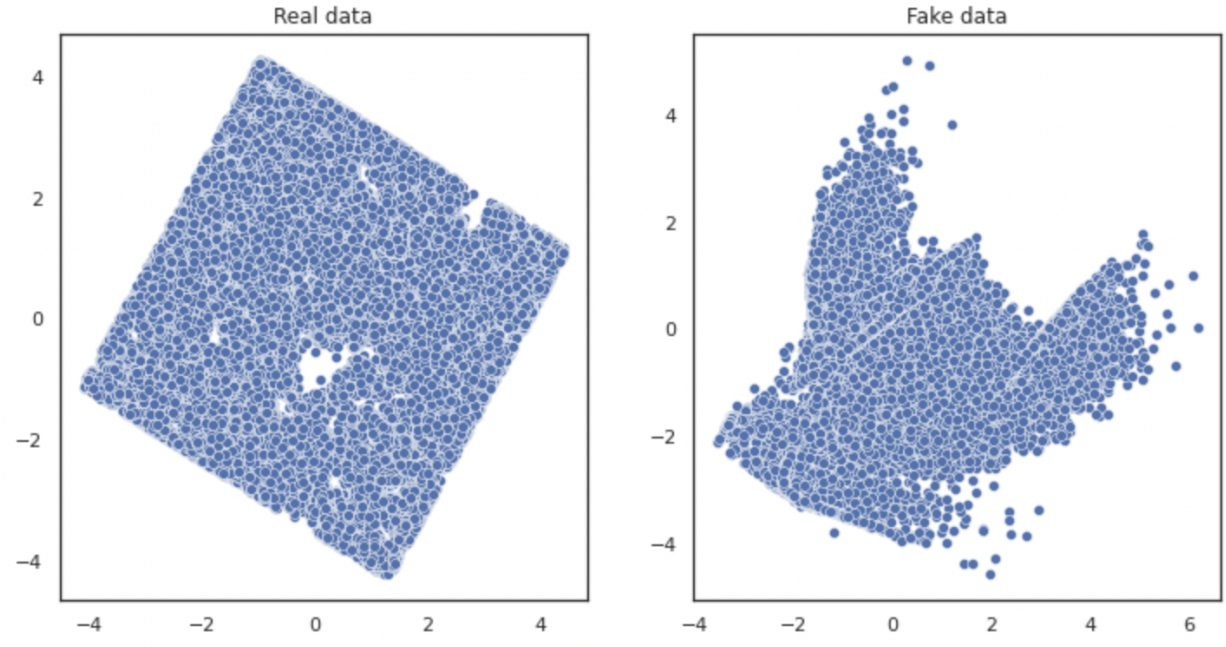


Figure 5.18: Two-dimensional PCA

Despite many shared properties, there are visibly many flaws in our obtained data, our model requires a surirgical precision for the generated data to be used as supplementary in the training set, exploiting this generated data for image classification as a training set will only hinder the process.

6 Conclusion

This research work on particle-laden turbulence has enabled us to apply three different machine learning methods (Tensorflow, KNN and DBSCAN) in order to form clusters but also to be able to analyze the dependence that exists between the Reynolds number and the Stokes number in the interactions that exist between the polydisperse cloud droplets. These particles in turbulence are from a three-dimensional direct numerical simulation of particle-laden isotropic turbulence [6].

We managed to obtain some results when clustering the data using both KMeans and KNN algorithms. While KMeans was useful when trying to partition particle in one sample flow, we could not obtain precise information about the clusters created via the different methods tested. On the contrary, using KNN allowed us to determine which Reynolds number was associated to a fluid : we trained our model with the training data and achieved an accuracy of 100 percent while inputting the remaining testing data.

Regarding the DBSCAN results, after using KNN to compute optimal epsilon values and having formed the clusters, we focused on the aspects of these clusters in particular their size, dimension, percentage of noise as well as their homogeneity and separation. The main goal is to see how the Stokes number and the Reynolds number vary according to these properties. What we can draw from this analysis is that the formation of clusters is closely related to these two physical parameters. Nevertheless, DBSCAN algorithm presents some limits at the level of the silhouette coefficient which presents low values thus translating a low homogeneity of the clusters independently of the physical parameters.

As for Tensorflow, the trained model for image classification based on the Stokes number reached an accuracy rate of 79% and passed the prediction test despite being slightly overfitted. Besides, after training our GAN model, we generated artificial data that maintained most of the properties of the original data, but due to an accuracy of only 70%, it wasn't sufficient enough to use it for data augmentation.

7 Appendix

Boxplots for DBSCAN algorithm

In the following, we highlight the results of the boxplots in order to account for the symmetry, the dispersion or the centrality of the distribution of the values associated with the two physical parameters which are the Reynolds number and the Stokes number. Also, these boxplots will allow us to account for the presence of extreme values in the data series.

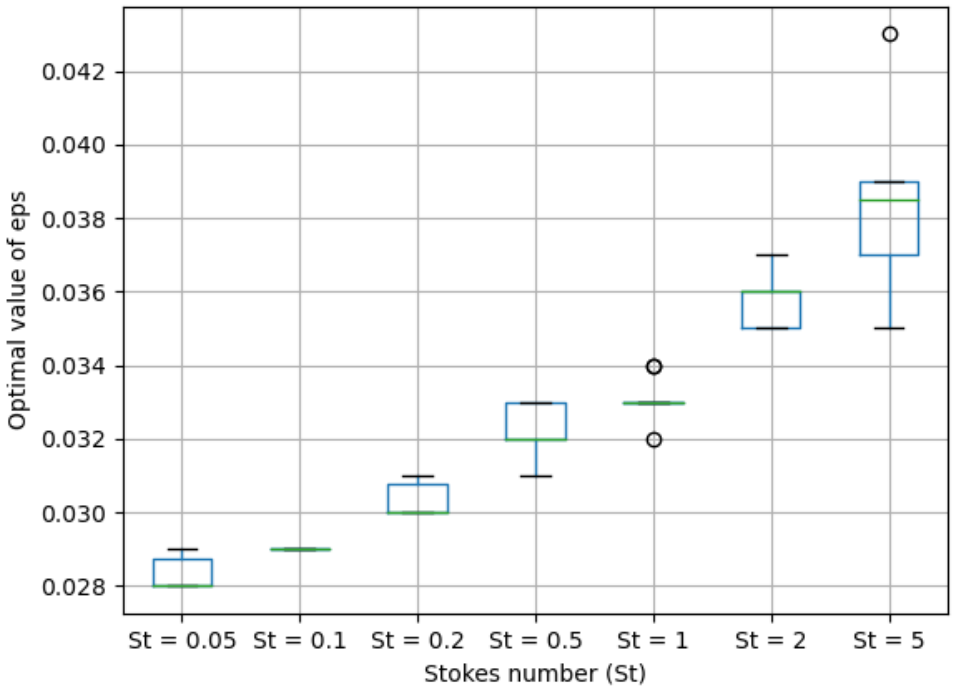


Figure 7.1: Box-plot of optimal *eps* of each St for $Re = 909$

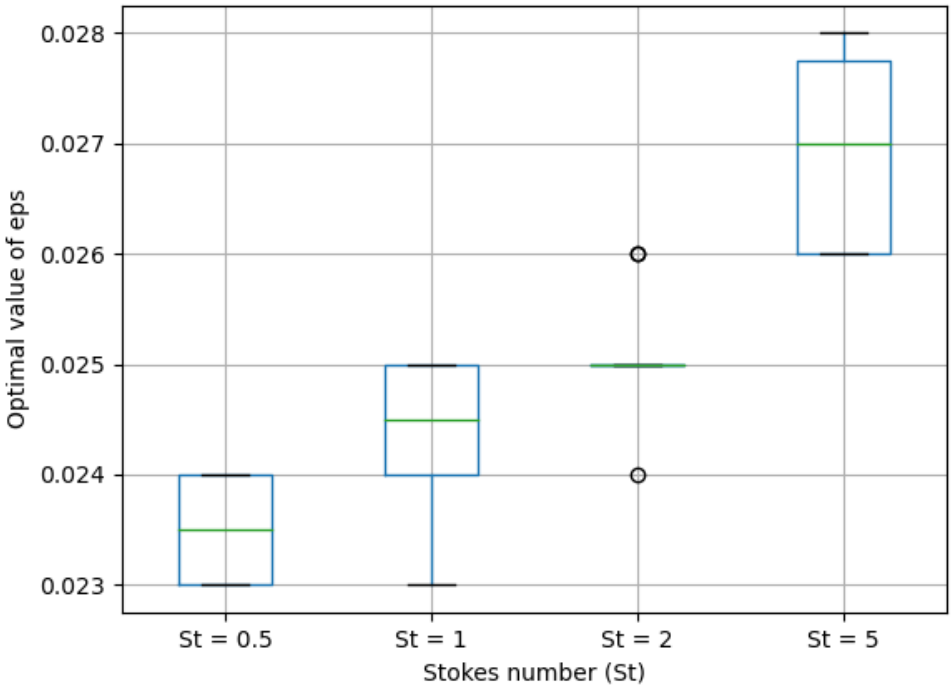


Figure 7.2: Box-plot of optimal *eps* of each St for $Re = 2220$

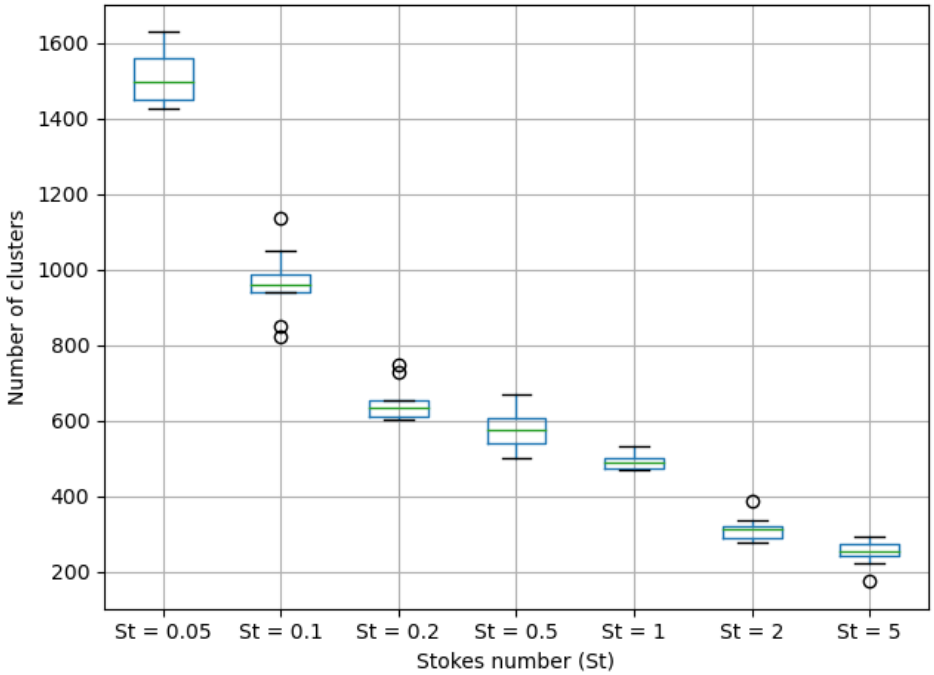


Figure 7.3: Box-plot of number of clusters of each St for $Re = 909$

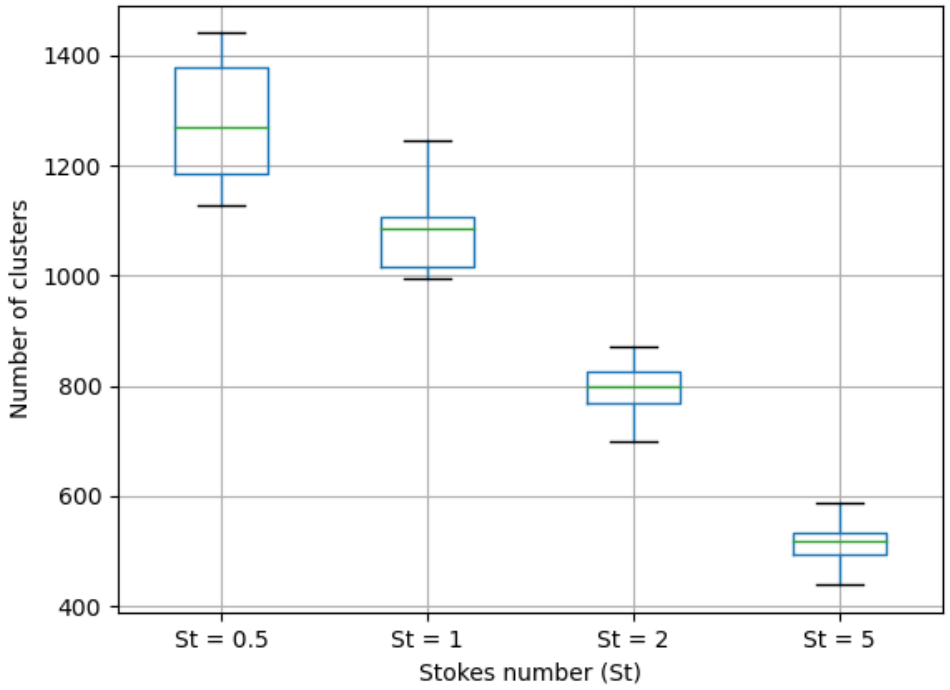


Figure 7.4: Box-plot of number of clusters of each St for $Re = 2220$

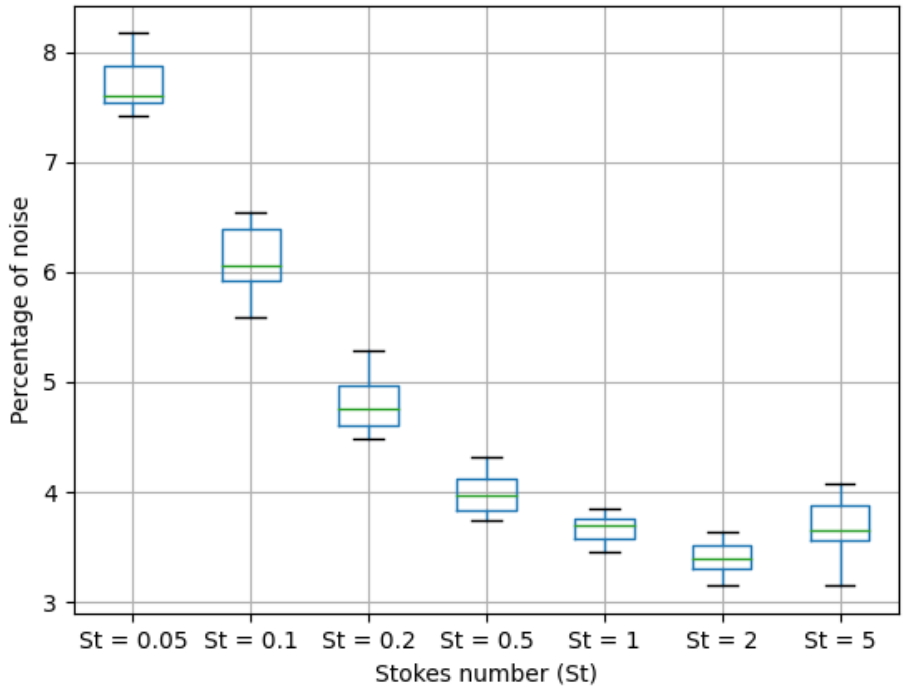


Figure 7.5: Box-plot of percentage of noise of each St for $Re = 909$

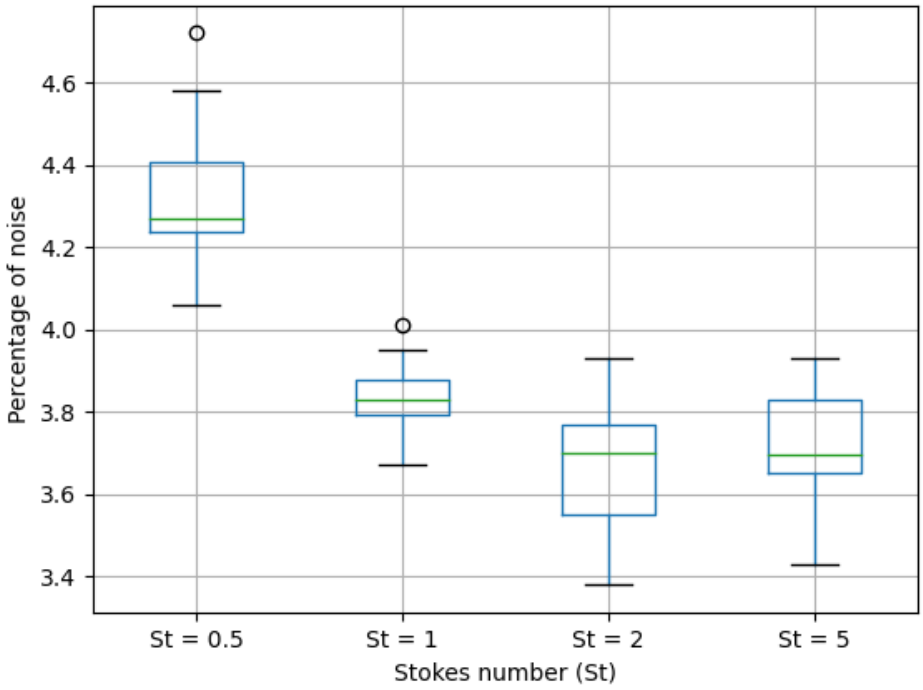


Figure 7.6: Box-plot of percentage of noise of each St for $Re = 2220$

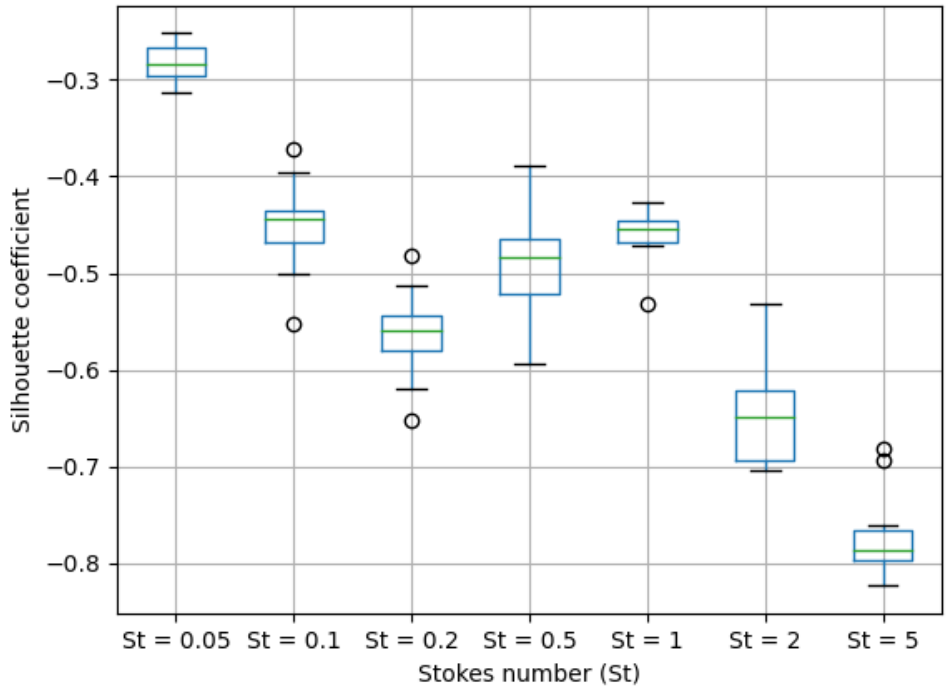


Figure 7.7: Box-plot of silhouette coefficient of each St for $Re = 909$

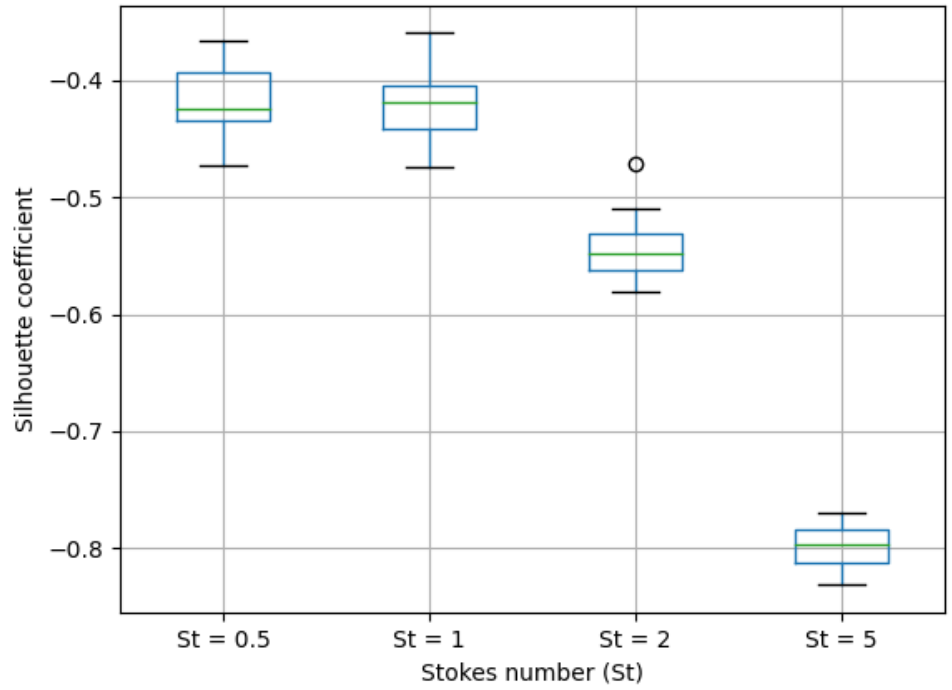


Figure 7.8: Box-plot of silhouette coefficient of each St for $Re = 2220$

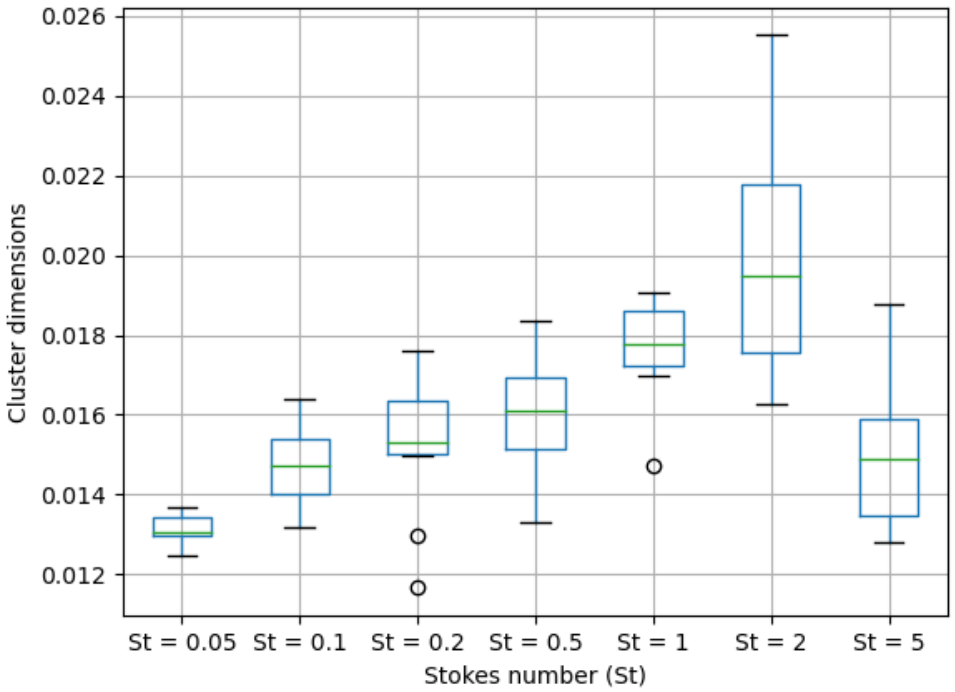


Figure 7.9: Box-plot of dimension of cluster of each St each St for $Re = 909$

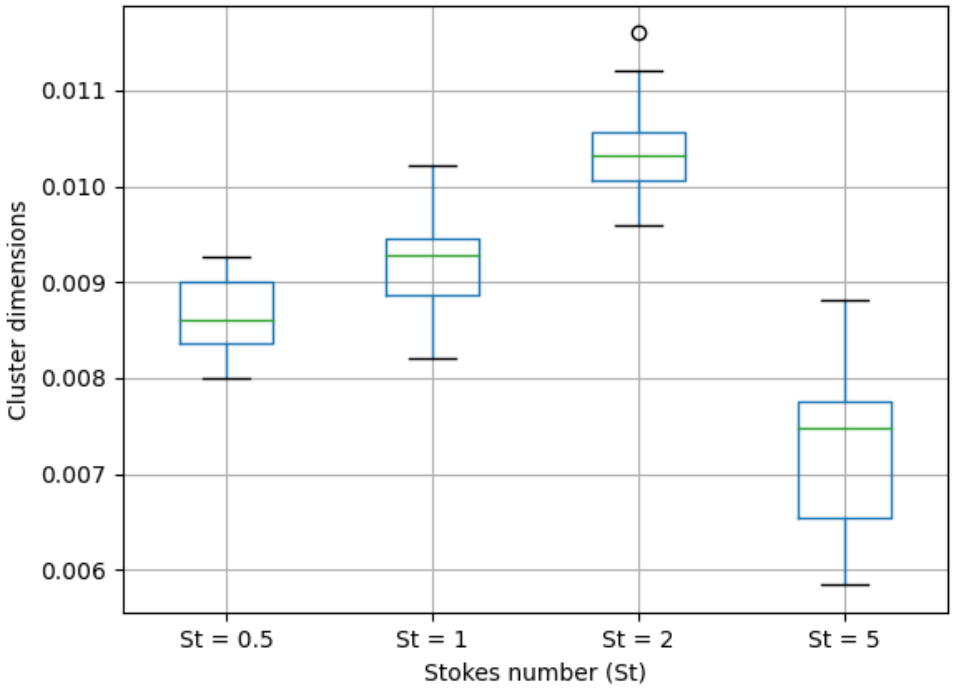


Figure 7.10: Box-plot of dimension of clusters of each St for $Re = 2220$

Confidence intervals for DBSCAN algorithm

In order to quantify the area of uncertainty on our estimators, we have calculated the confidence intervals for these last ones using the normal distribution. This allows us to know the range of values within which we are 95% certain of finding the true value we are looking for. It is calculated as :

$$Confidence\ Interval = \bar{x} \pm t \times (\frac{s}{\sqrt{n}}) \tag{7.1}$$

where:

$\bar{x}$: sample mean
t: t-value that corresponds to the confidence level (95%)
s: sample standard deviation
n: sample size

The confidence interval program in python is as follows :

```python
def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1)
    return [m-h, m+h]
```

Figure 7.11: Python confidence interval program

N_g/St	**0.05**	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
909	[0.0279, 0.0286]	0.029	[0.0299, 0.0306]	[0.0318, 0.0328]	[0.0327, 0.0335]	[0.0352, 0.0362]	[0.0368, 0.0398]
2220	X	X	X	[0.0231, 0.0238]	[0.0238, 0.0249]	[0.0246, 0.0255]	[0.0262, 0.0275]

Table 7.1: Confidence interval of optimal *eps*

N_g/St	**0.05**	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
909	[0.0279, 0.0286]	0.029	[0.0299, 0.0306]	[0.0318, 0.0328]	[0.0327, 0.0335]	[0.0352, 0.0362]	[0.0368, 0.0398]
2220	X	X	X	[0.0231, 0.0238]	[0.0238, 0.0249]	[0.0246, 0.0255]	[0.0262, 0.0275]

Table 7.2: Confidence interval of number of clusters

N_g/St	**0.05**	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
909	[0.0279, 0.0286]	0.029	[0.0299, 0.0306]	[0.0318, 0.0328]	[0.0327, 0.0335]	[0.0352, 0.0362]	[0.0368, 0.0398]
2220	X	X	X	[0.0231, 0.0238]	[0.0238, 0.0249]	[0.0246, 0.0255]	[0.0262, 0.0275]

Table 7.3: Confidence interval of percentage of noise

N_g/St	**0.05**	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
909	[-0.297, -0.267]	[-0.487, -0.414]	[-0.598, -0.528]	[-0.528, -0.446]	[-0.481, -0.440]	[-0.686, -0.603]	[-0.806, -0.738]
2220	X	X	X	[-0.445, -0.394]	[-0.446, -0.395]	[-0.565, -0.518]	[-0.812, -0.783]

Table 7.4: Confidence interval of silhouette coefficient

Re/St	**0.05**	**0.1**	**0.2**	**0.5**	**1**	**2**	**5**
909	[0.0128, 0.0134]	[0.0140, 0.0154]	[0.0139, 0.0165]	[0.0149, 0.0172]	[0.0167, 0.0186]	[0.0177, 0.0223]	[0.0135, 0.0163]
2220	X	X	X	[0.0083, 0.0089]	[0.0087, 0.0096]	[0.01, 0.0108]	[0.0066, 0.0079]

Table 7.5: Confidence interval of dimensions of Cluster

Python programs

Some pieces of python programs used to compute the DBSCAN algorithm.

```
foldernames = glob.glob("TERData/*") #TERData contient les données et doit être dans le répertoire de travail
filenames = [] #contiendra les noms des différents répertoires des données
for i in foldernames:
    filenames.append(glob.glob(i+"/*.txt")) #Permet de récupérer une liste constituée de listes contenant
                                            #les noms de tous les fichiers, classÃ©s par nom de dossier
tabnames = [] #contiendra les nom des données
tabval = [] #contiendra les données (toutes les données), donc liste de dataframe
tabval1 = []


for i in filenames:
    for j in i: #on parcourt toutes les donnÃ©es
        pos1 = j.find('\\') #on va rÃ©cup un nom pour chaque donnÃ©e
        pos2 = j.find('\\pcl')
        sousChaine1 = j[pos1:pos2]
        pos3 = j.find('pcl')
        pos4 = j.find('.txt')
        sousChaine2 = j[pos3:pos4]
        # sousChaine commence par \, on va supprimer le premier caractère
        sousChaine = sousChaine1 + sousChaine2
        #premier caractère supprimé
        sousChaine = sousChaine[1:]
        #on ajoute le nom des donnÃ©es Ã©tudiÃ©es dans la table tabnames
        tabnames.append(sousChaine)
        #on lit les data associées au nom dans la table filenames
        dataset = pd.read_csv(j, delimiter = "  ", header = None, engine='python' )
        X=dataset.iloc[:,0:2].values
```

Figure 7.12: Python program for recovering all files

```
neigh = NearestNeighbors(n_neighbors=5)
# Ajustez l'estimateur des voisins les plus proches de l'ensemble de données d'entraînement
nbrs = neigh.fit(X)
# Trouve les K-voisins d'un point.
# Renvoie les indices et les distances des voisins  de chaque point
distances, indices = nbrs.kneighbors(X)
# Tri croissant des distances suivant la première colonne
distances = np.sort(distances, axis=0)
# On recupère la colonne 1
distances1 = distances[:,1]
#On cherche le coude
count_dist=int(99/100*len(distances1))

Eps_optimal=float(distances1[count_dist,])

#on ajoute les distances dans la table tabval
tabval.append(distances1)
#on ajoute les valeurs optimales des epsilons dans la table tabval1
tabval1.append(Eps_optimal)

if not(os.path.isfile(path+'/'+sousChaine+".png")):
    plt.plot(distances1);       # Tracer de la courbe croissante
    plt.title('Determining of optimum Eps value : %f' % Eps_optimal)
    plt.savefig('VizData_Eps2/'+sousChaine+'.png') #on sauvegarde la figure dans un sous dossier
    plt.clf() #on ferme la figure
```

Figure 7.13: Python program to determine optimal epsilon

```
#Ici, on programme l'algorihtme DBSCAN
db=DBSCAN(eps = Eps_optimal, min_samples=5)
y_pred = db.fit_predict(X)
#plt.scatter(X[0],X[1],c = y_pred) #Graphique avec les clusters
#plt.show()

print(y_pred)
print(db.core_sample_indices_)

# y_pred contient pour chaque element du dataset, le cluster associé. Par exemple
# le troisième et le deuxieme element du dataset apartiennent au cluster “0”,
# le premier au cluster “39”, le quatrieme au cluster “1”. les elements qui ont “-1”
# sont considérés comme du bruit.

# db.core_sample_indices_ contient les indices des elements qui sont considérer comme noyaux.
# c’est à dire les points qui ont au moins 5 (min_samples) voisins qui sont à une distance
# inférieure ou egale à 0.03 (eps).

# identify core samples
core_samples = np.zeros_like(y_pred, dtype=bool)
core_samples[db.core_sample_indices_] = True
print(core_samples)

# declare the number of clusters and the number of noises,
n_clusters_ = len(set(y_pred)) - (1 if -1 in y_pred else 0)
n_noise_ = list(y_pred).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhoette Coefficient: %0.3f" % metrics.silhouette_score(X, y_pred))
```

Figure 7.14: Python program to compute DBSCAN algorithm

Bibliography

[1] "Study of tiny droplets could have big impact on industrial applications". In: (2012). URL: https://www.princeton.edu/news/2012/02/22/less-more-study-tiny-droplets-could-have-big-impact-industrial-applications.

[2] "Understanding tiny droplets can make for better weather forecasts". In: (2016). URL: https://partner.sciencenorway.no/forskningno-norway-ntnu/understanding-tiny-droplets-can-make-for-better-weather-forecasts/1433409.

[3] *Wall accumulation and spatial localization in particle-laden wall flows*. 2012. URL: https://www.researchgate.net/profile/Francesco-Picano/publication/236634353_Wall_accumulation_and_spatial_localization_in_particle-laden_wall_flows/links/00b49518984e008206000000/Wall-accumulation-and-spatial-localization-in-particle-laden-wall-flows.pdf.

[4] David Richter and al. "Rayleigh-Benard turbulence modified by two-way coupled inertial, nonisothermal particles, Physical Review Fluids". In: (2018).

[5] Keigo Matsuda and Ryo Onishi. "Turbulent enhancement of radar reflectivity factor for polydisperse cloud droplets". In: (2019). URL: https://acp.copernicus.org/articles/19/1785/2019/acp-19-1785-2019.html.

[6] MATSUDA and al. *Influence of microscale turbulent droplet clustering on radar cloud observations.* Atmos. Sci.71(10), 3569–3582., 2014.

[7] *Nombre de Stokes*. 2020. URL: https://fr.wikipedia.org/wiki/Nombre_de_Stokes.

[8] ANTONINOFERRANTE Aand SAID ELGHOBASHI. *Reynolds number effect on drag reduction in a microbubble-laden spatially developing turbulent boundary layer*. Cambridge University Press, J. Fluid Mech. (2005) vol. 543, pp. 93–106., 2005.

[9] Benjamin DEVEZE and Matthieu FOUQUIN. *DATAMINING C4.5 - DBSCAN*. 2004.

[10] https://penseeartificielle.fr/clustering-avec-lalgorithme dbscan/. "Clustering avec l'algorithme DBSCAN". In: (2019).

[11] Jiwon Jeong. "Who Is Your Golden Goose?: Cohort Analysis". In: (2019). URL: https://towardsdatascience.com/who-is-your-golden-goose-cohort-analysis-50c9de5dbd31.

[12] *Silhouette (clustering)*. 2021. URL: https://en.wikipedia.org/wiki/Silhouette_(clustering).

[13] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[14] Venkatesh Umamaheswaran. “Comprehending K-means and KNN Algorithms”. In: (2018). URL: https://becominghuman.ai/comprehending-k-means-and-knn-algorithms-c791be90883d.

[15] Martin Ester and al. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. Institute for Computer Science, University of Munich, 1996.

[16] Statistical tools for high-throughput data analysis. “DBSCAN: density-based clustering for discovering clusters in large datasets with noise - Unsupervised Machine Learning”. In: (). URL: http://www.sthda.com/english/wiki/wiki.php?id_contents=7940.

[17] Wikipedia. “DBSCAN”. In: (2021). URL: https://en.wikipedia.org/wiki/DBSCAN.

[18] Kelvin Salton. “How DBSCAN works and why should we use it?” In: (2017). URL: https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80.

[19] Shritam Kumar Mund. “How does DBSCAN clustering algorithm work?” In: (2019). URL: shritam.medium.com/how-dbscan-algorithm-works-2b5bef80fb3.

[20] Indraneel Dutta Baruah. “Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python”. In: (2020). URL: https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad.

[21] URL: https://creativecommons.org/licenses/by-sa/3.0/deed.fr.

[22] Sumit Saha. “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”. In: (2018). URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[23] “Synthetic data”. In: (2020). URL: https://en.wikipedia.org/wiki/Synthetic_data.

[24] “Generative adversarial network”. In: (2020). URL: https://en.wikipedia.org/wiki/Generative_adversarial_network.

[25] Anubha Pandey-Deepak Bhatt-Tanmoy Bhowmik. *Limitations and Applicability of GANs in Banking Domain*. 2020. URL: http://ceur-ws.org/Vol-2692/paper1.pdf.

[26] 2020. URL: https://package.wiki/table-evaluator.