READ ME

_1_run_interpolation_rotation.py

INPUTS TO CHANGE :

- Folder_fig, Folder_data : folders where you want your data and your figures to be stored (an under folder will be created for each case)
- D : size of the cylinder
- Wavelet_size

Read the path and find the folders we want to analyze (calibration plate, PIV, Prof)

Calls the functions from *_1_def_velocities.py* and *_1_GIF_3D.py*

Extract both the frames of the PIV and the profilometry data

Creates the proper coordinates system by sliding the PIV frame for it to match with the calibration plate (select the point along the ruler 30cm after the cylinder according to the ruler)

Run a loop which will deal with each dataset the exact same way

- create folders to store the data and the figures where we want
- access the PIV data and then extract U, V, and W and the Tot=[U,V,W] from the data
- access the profilometry data and extract elevation from the PROF data
- interpolates the 2 data sets on the slided PIV grid and return ITPP_elevation interpolated
- ; and saves them
- rotates the datasets to have the wake parallel to the x-axe according to the mean U velocity, cleans the borders, and centers the wake along the y-axe ; and crops the frames at the same size, with a shared frame x_mm, y_mm(in mm) and saves everything (it rotates the dataset of the same angle on the time resolved and NOT time resolved datasets)
- it calculates the mean of each (U,V,W)
- it saves u,v,w (fluctuating velocities)
- it computes and saves the vorticity wz
- it plots the enstrophy mean(wz**2) (those plots are saved as PDF)
- it computes the svd, for the NOT TimeResolved cases, on tot (concatenation of u,v and w) and on h and on wz and then separates them and plot the ten first modes of the velocity in each direction (those plots are stored as PDF), it also plots the singular values (stored as PDF)
- saves a 3D GIF of the surface elevation (PROF)

- computes the gradients of the profilometry datasets and stores them
- applies the wavelet transform on the PROF data (wavelet size to determined)

_1_def_velocities	
BIG	Read the data and extract U,V and W
rotation	Rotate the data so that the wake is parallel to the x-axe
clean	Remove the weird borders created by the rotation
centered	Make sure that the center of the wake is in the center of the y-axe
get_PIV_coordinates	Get the frame data, x(=x_mm) and y(=y_mm) in meters
velocities2	Rotates and cleans U,V,W and h and the frame according to Umean (calling rotation, clean and centered)
SVD	Computes the SVD on Tot
Blurred	Blurres the data to avoid the gradient to go crazy
rotz	Computes the rotational wz of the flow = the vorticity
Mexican hat	Computes the Mexican hat formula to create the wavelet
W_transform	Apply the wavelet transform on a dataset according to a specific wavelet
PLOTS	
show_frame	Plots the 2D matrix with colors, pictures of the wake mostly
modes2	Plots the 10 1st modes extracted by the SVD of u,v and w
energy	Plots the evolution of the singular values
modes1	Plots the 10 1st modes of the vorticity and profilometry

_2_run_scales.py

INPUTS TO CHANGE :

- Folder_fig, Folder_data : folders where you want your data and your figures to be stored (an under folder will be created for each case)
- D : size of the cylinder
- mu

Read the path and find the folders we want to analyze only the PIV

Calls the functions from *_2_def_scales.py* and *_2_def_function.py*

Run a loop which will deal with each experimental case the exact same way

- Calls x_mm,y_mm, u,v, U
- Computes and stores Rux (autocorrelation function of u according to x)
- Computes and stores sfux (structure function of u according to x)
- Plots Rux(r) for different position in the wake

- Plots sfux(r) with an automatic fitting following ar**(2/3) in the inertial range(which is automatically detected : where the regression fits the best)(inertial range found as the average eof those slopes for every points IN the wake)
- This fit enables to compute Epsilon =(a/C)**(3/2) where C=2, Epsilon is also plotted in space and depending on x for different y
- Computes the integral scale Li which is the surface under the curve of Rux before it crosses zero; Li is also plotted depending on x
- Li_fit the same but Rux is approximated by an exponential fit; plotted
- Computes Cepsilon =(Epsilon*Li)/(urms**3) ;plotted
- Plots the mean in bins of Cepsilon to see the tendency
- Computes and plots the different scales of turbulence Lambda(taylor), eta(Kolmogorov)
- Computes and plots the Reynolds number depending on Lambda(=ReynoldsL)
- Plots Li/Lambda depending on ReynoldsL
- Plots Cep depending on ReynoldsL
- Plots Cep depending on the inverse of ReynoldsL
- Plots average bins for the last 3 ones to see tendencies

_2_def_function	Autocorr_fct_rx	Computes the structure fct
	Structure_fct_rx	Computes the autocorrelation fct
_2_def_scales	regression	Calls inertial_regression(makes the fit) and inertial(finds the inertial range) Computes the dissipation rate Espilon = E
	Li and Li _fit	Computes the integral scale Li and Li_fit
	Cep	Computes the proportionality coeff Cepsilon
	Cep_mean	Computes bins of Cep for clearer graphes
	lambdas	Computes Taylor scale Lambda=L
	F_Reynolds	Computes the Reynolds number depending on the Taylor scale= ReynoldsL
	Kolomogorov_Scale	Computes the length(eta), time(tau_eta) and velocity(u_eta) scale of Kolmogorov
PLOTS	drawing	Draws several curves for each point in A depending on r
	curves	Draws several curves for different y centered depending on x in mm
	drawing_log	Draws several logarithmic curves for each point in A depending on r
	show_frame	Plots the 2D matrix with colors, pictures of the wake mostly
	wake_lines	Draws lines along the wake to see what exactly we plotted before
	curves5	Plots curves of values depending on x in mm

	curves2	Plots points of values depending on ReynoldsL
	moyenne	Creates bins to have average values and clear graphs

_3_run_vortex.py

Read the paths of the PIV and the PROFilometry and find the folders we want to analyze, only the timeResolved ones

Calls the functions from *_3_def_vortex.py* (same as *def_lambda.py*)

Run a loop which will deal with each dataset the exact same way

- loads the frame, u, v, U ,V and h
- Computes the Lambda 2 criterion on the data (with blurring the data before computing the gradient of scheme 5)
- Calculates from that the adimentionalized Lambda (named with a "_")
- Determines the threshold to apply to the lambda criterion on the adimentionalized data (it is calculated with the curved of the percentage of area occupied by vortices depending on the value of the threshold, the threshold match with the point where the curve goes higher than a chosen percentage (here 0.45% but can be change in the arguments) the goal is to have this threshold just after the bump)
- Then Lambda and its threshold are stored
- Creates a dictionary where the keys are the name of the vortex (a number) and the value is its different positions in the different snapshot [t,(x,y)]
- Stores this dictionary after removing the vortices that lives for less than 3 snapshots
- Computes the PDF of the time length of the vortices detected.
- Computes the wavelet transform W of h for a given wavelet length
- Determines the threshold to apply to W (only keeps 1% of the structures)
- Creates a new dictionary where the keys are the name of the vortex (a number) and the value is its different positions in the different snapshot [t,(x,y)]
- Stores this dictionary after removing the vortices that lives for less than 9 (3x3) snapshots
- Creates a GIF of the vortices motion underneath the surface with white circles circling the vortices detected and green circles for the vortices detected at the surface

blurred	Blurres the picture with a gaussian filter with a window 3by3

grad5	It is the gradient method used to compute lambda
lambda2	Computes lambda with its mathematical formula from Jeong et Hussain (1995)
area_pourcentage2	Plots the curve of the percentage of the frame occupied by the vortices depending on the thresholding (in average in time) And determined where the thresholding by choosing a thresholding percentage (you have to choose this percentage)
vortex_center	Collects the centers of the vortices in the frame thanks to the scipy.local_minimum method (3*3 boxes)
detection	Creates the dictionary to follow the vortices from the PIV in time using **methods // then if no vortex is found within the search radius re at time t + dt, then we "skip" a moment and restart the search at time t + 2dt. The new search location will be 2(du+dv) downstream of the initial observation.
Mexican_hat and W_transform	Computes the wavelet transform on h, the surface elevation field
detection2	Creates the dictionary to follow the vortices from the PROF in time using **methods
PDF2	Plots a PDF depending on time
update	Update function used to animate the GIF (adding the circles green or white)
gif_vortex	Use FuncAnimation to creates the GIF at the real speed and storing it (15 frame each second)

[Numerically, vortex structures are initially detected from the local minima of the λ2 field for subsurface flows, or from the field resulting from the wavelet transform for surface flows. Each detected vortex is recorded in a dictionary-like structure, its key is a unique number identifier **n. The corresponding value is an array containing the time of occurrence t (identified by the snapshot number) and the spatial coordinates (x, y) of the detection, denoted [t, (x, y)].

The temporal tracking method is based on a predictive search based on the displacement of the vortex with the average velocity field. For each vortex detected at the previous time t − dt, an estimated position at time t is calculated by translating the distance du+dv where du = ⟨U ⟩dt and dv = ⟨V ⟩dt. A search circle of radius re is then centered on this predicted position. If one or more new vortices detected at time t are located inside this circle, the one with the minimum distance from the predicted position is selected. This new vortex is then considered as the temporal continuity of the vortex identified by **n**, and its position (x', y') is added to the trajectory of the vortex, in the form [t, (x', y')].] (in that function, t0 is the first time the vortex we are going to link the new detection to as appeared in the frame. In other word we are going through the dictionnary to find the vortex to which linked this new detection but not the whole disctionnary only the vortices that appeared not earlier than 20 frames before, this what this condition mean. Because we have shifted the previous position we are searching

for what was the name (identification number) that previous vortex to add this new detection in the right list)

!! be carefull, to open the stored dictionary :

```
data = np.load("dico_vortex.npz", allow_pickle=True)
vortex = data["vortex"].item()
```

Bonus :

Strict_wave.py

Loads h the surface elevation and the frame and convert it in meter

INPUTS TO CHANGE :

- fr, the acquisition frequency
- U the mean velocity of the flow
- g the gravity (not supposed to change…)
- alpha the criterion of cropping to be stricter on the wave removal

Compute H the fourier transform of the surface elevation
Computes the water characteristics
Computes the dispersion relation curves with those characteristics
Plots those curves on the FFT of h in the frame (Kx,w) averaged over Ky and for Ky=0
Create a mask along those curves to remove the waves from the spectrum, the alpha criterion can make those curves stricter to remove the wave noise
Apply the symmetric of this mask according to the FFT properties
Constructs the image back with the wave removed
Apply that method on every frame with a loop
Creates a GIF of this new surface elevation without any waves
Creates a GIF of this new surface elevation without any waves with a threshold applied

Water_prop	Calculates the water properties depending on the temperature
Show_frame	Plots a np.array with a log scale (for the fourier transform results)

Graphs_4cases_adimensionalized.py

Loads everything we need and create lists to make loops
!!Be careful with the storing path, to be changed
Plots the structure function sfux(r) for the 4 cases at the same position in the wake with their corresponding inertial slopes
Plots the 4 u_rms 1 for every case with the same scale on the same picture
Plots the calibration plate and mean U raw (without any modification)

Plots the autocorrelation function Rux(r) for the 4 cases at the same position in the wake

Plots the points clouds for the 4 cases of Cep depending on Re_lambda