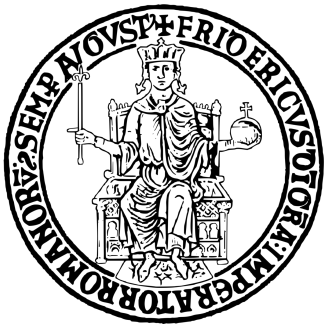
UNIVERSITÀ DEGLI STUDI DI NAPOLI

"FEDERICO II"



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA IN INGEGNERIA AEROSPAZIALE

CLASSE DELLE LAUREE IN INGEGNERIA INDUSTRIALE (L-9)

Bachelor's degree dissertation

Vertical liquid jet: effect of the contact angle and data-driven model reduction.

Supervisor:

Prof. Matteo Chiatto

Advisor:

Dott. Antonio Colanera

Candidate:

Russo Andrea

N35003980

Academic Year 2023/2024

Contents

Abstract

This study examines the dynamics of gravitational thin liquid films in two main configurations. The initial setup utilizes a conventional approach, performing a modal decomposition analysis through Proper Orthogonal Decomposition (POD). The subsequent setup, instead, assesses the influence of inlet contact angles (40 and 140 degrees) on the flow dynamics using the same method. Applying the POD technique allows us to create a decomposed model, facilitating the identification of physically relevant structures. The analysis is conducted via BASILISK software using a two-phase Volume of Fluid (VOF) approach. The purpose of this research is to enhance our comprehension of these phenomena and assess whether any noticeable changes in flow patterns are evident while applying a contact angle. Findings reveal that the traditional setup operates as anticipated, displaying a sinuous-varicose flow behavior. However, varying the contact angle seems to have a limited impact on the original configuration; additional research is needed to clarify these results.

Chapter 1

Introduction

This paper seeks to explore and further understand the behavior of gravitational thin liquid sheets across two fundamental setups. In the first scenario, we explore a conventional setup and perform a modal decomposition analysis. In the second, our goal is to determine if the presence of a contact angle significantly impacts the flow. For the conventional setup, we are going to analyze the flow field behavior using a mathematical method called Proper Orthogonal Decomposition (POD). This technique allows us to develop a decomposed model of the phenomenon, facilitating the extraction of physically significant features, and ultimately enabling the creation of a desired reconstruction. The POD technique has been extensively discussed in the works of K. Taira *et al.* [11], Steven L. Brunton *et al.* [4], and executed by A. Colanera *et al.* [5]. It was A. Arote *et al.*, in the previously referenced articles ([2], [3]), who first introduced its application to liquid sheets.

The application of modal decomposition analysis through the POD technique marks a crucial advancement in the research of liquid sheets, originating in the mid-1900s and widely applied across scientific and industrial fields like fuel atomization, coating deposition, papermaking, and more. Using the POD technique on the liquid sheet allows for the identification of dominant spatial structures in the flow.

The analysis is conducted using the BASILISK software [http://basilisk.fr], employing a two-phase Volume of Fluid (VOF) approach. Pertaining to this topic, reference is made to previous applications of the VOF technique in simulating gravitational liquid sheets carried out by A. Della Pia *et al.* [7] [8] and A. Colanera *et al.* [5]. Like this research, these investigations begin with the incompressible linearized Navier-Stokes equations. The water-air interface analysis is also addressed in the works of A. Arote *et al.* [2] and [3].

As previously said this paper also investigates the impact of different contact angles on flow field behavior. It examines two specific configurations: one with sinusoidal forcing at the inlet and one without. In the simulation a Weber number of 2 and a Reynolds number of 413 are implemented, placing the configuration in the supercritical regime (We $>$ 1).

The primary aim of this study is to enhance our comprehension of the impacts and magnitudes of various contact angles on thin viscous sheets, and to determine if significant alterations in the flow are detectable. The selected angles, 40 and 140 degrees, are chosen to emphasize the contrasts between low and high contact angles. This analysis is supported by the work of Afkhami *et al.* [1], and it implements a BASILISK library based on their findings.

Researching gravitational liquid sheets enhances our understanding of waterfall dynamics. Grasping waterfall flow is essential for foreseeing unpredictable oscillations and averting overflows that might cause noise and vibrations across vast areas. The resultant noise can be a nuisance and a source of potential accidents, potentially leading to dangerous vibrations in nearby building window frames or structural harm to dams.

Chapter 2

Numerical Simulation

2.1 Physical Domain

In figure 2.1, a schematic representation of the physical domain, the adopted numerical grid, and a sample snapshot of the volume fraction C are presented. The right and left interfaces are denoted as $y^+(x,t)$ and $y^-(x,t)$, respectively, as referenced in [2.1a]. Given the geometry of the interface, the centerline shape, denoted as $l(x,t)$, and the liquid sheet thickness, denoted as $h(x,t)$, are computed according to.

$$\begin{bmatrix} l(x,t) \\ h(x,t) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & -1 \end{bmatrix} \begin{bmatrix} y^+(x,t) \\ y^-(x,t) \end{bmatrix} \tag{2.1}$$

The setup involves a forced liquid sheet flow characterized by a fully developed parabolic axial velocity profile at the inlet. This profile undergoes perturbation due to a sinusoidal forcing with variable amplitude in the lateral velocity component. Consequently, Dirichlet boundary conditions are enforced at the inlet for $y \in [-H/2; H/2]$.

$$u = \frac{3}{2}U\left(1 - \left(\frac{2y}{H}\right)^2\right), \tag{2.2}$$

$$v = \hat{A}U\sin(2\pi f_f(t - t_s))\vartheta(t - t_s), \tag{2.3}$$

$$C = 1, \tag{2.4}$$

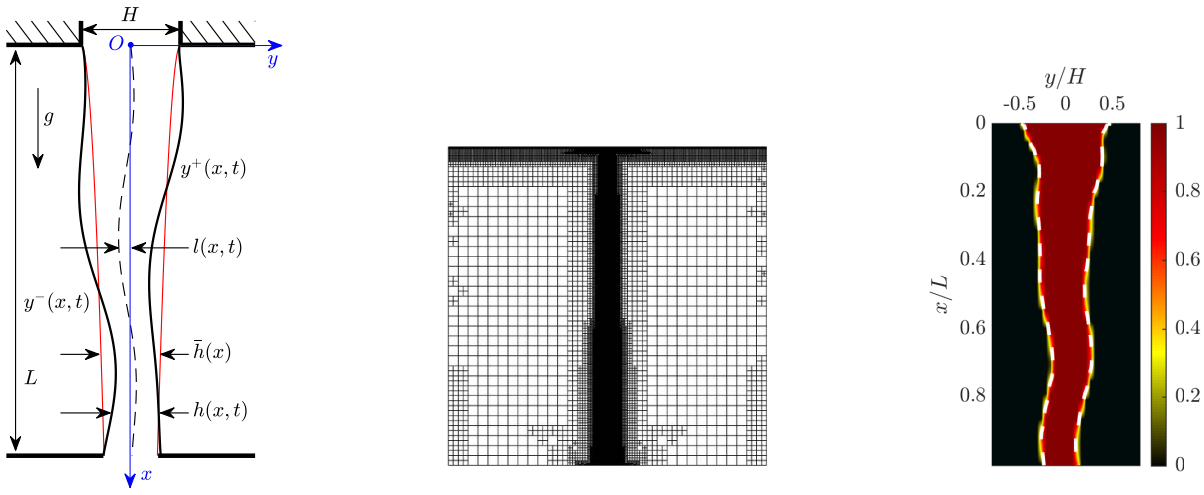
Where U and H represent the mean axial velocity and the thickness of the liquid sheet at the inlet, respectively. $\hat{A}$ denotes the dimensionless amplitude of the forcing lateral velocity

perturbation, f_f is the forcing frequency, and $\vartheta(t-t_s)$ is the Heaviside function that comes into effect at the forcing starting time t_s. For the remaining portion of the inlet side ($|y| > H/2$), no-slip conditions are applied. Neumann boundary conditions are imposed on the right and left edges for all variables, while standard outflow conditions are set at the lower edge.

$$\frac{\partial u_i}{\partial x} = 0, \tag{2.5}$$

$$\frac{\partial C}{\partial x} = 0, \tag{2.6}$$

$$p = 0. \tag{2.7}$$



(a) Domain sketch (b) Numerical grid (c) snapshot of C

Figure 2.1: Typical numerical arrangement and sample output of the liquid sheet

The computational domain is a square with dimensions of $L \times L$. The mesh employed is a quadtree-structured grid, as illustrated in panel 2.1b, maintaining maximum refinement within the rectangular region $0 < x/L < 1$ and $1 < y/H < 1$, encompassing the entire liquid sheet. Outside this region, grid cells are dynamically refined based on user-defined adaptation criteria. For a detailed explanation of the adaptive grid refinement strategy, readers are directed to van Hooft et al.'s work [12].The modal analysis specifically focuses on the region $0 < x/L < 1$ and $1 < y/H < 1$, discretized with a grid of 1025×41 uniform cells. Numerical data are collected at intervals of 5×10^3 s over a period of $T = 12$ s. Dimensionless parameters are introduced for subsequent analysis, including the Reynolds number ($Re = \rho_l UL/2(\mu_l)$), the Froude number ($Fr = U^2/(gL)$), the Strouhal number ($St = fH/U$), and the density ratio ($r_p = \rho_a/\rho_l$).

2.2 Volume Of Fluid approach

This work is describing a numerical investigation that has been conducted using the BASILISK open-source code. BASILISK is commonly used for fluid dynamics simulations, and in this case, it utilizes a single-phase formulation and the volume-of-fluid (VOF) approach.
In the VOF approach, the two-phase flow is treated as a single-fluid flow. The volume fraction, denoted as C, is introduced as a field variable. The volume fraction has values of 1 in the inner phase (representing one phase of the fluid) and 0 in the ambient phase (representing the other phase). The interface between the two phases is defined when the volume fraction is 0.5.
In this context, the simulation incorporates models for density ρ and viscosity μ

$$\rho = \rho_a = (\rho_l - \rho_a)C, \tag{2.8}$$

$$\mu = \mu_a = (\mu_l - \mu_a)C, \tag{2.9}$$

where subscripts a and l denote the ambient and liquid phases, respectively.
Therefore, based on equations (2.8) and (2.9), the density and viscosity remain constant within each phase and undergo variations across the separating interface, positioned where $0 < C < 1$. Throughout this study, the interface is associated with the isoline $C = 0.5$. It is important to note that this representation is a simplification; in reality, the transition from the external environment to the fluid occurs gradually. The volume fraction C is governed by the following advection equation.

$$\frac{\partial C}{\partial t} + \frac{\partial C u_i}{\partial x_i} = 0 \tag{2.10}$$

The momentum equations and the divergence-free condition (represented in index notation) are:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.11}$$

$$\rho(\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j}) = -\frac{\partial \rho}{\partial x_i} + \frac{\partial [\mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})]}{\partial x_j} + \sigma k n_i \delta_S \tag{2.12}$$

In this equation the surface tension is modelled as an impulsive force located at the interface. The governing equations, akin to those used in incompressible flows, are solved using the projection method. In the initial step, a temporary velocity field is determined by neglecting

the pressure gradient. Subsequently, in the second step, this field is projected onto a space of divergence-free velocity fields by incorporating the requisite pressure gradient information. The Volume-of-Fluid (VOF) method is employed to track the interface, solving Eq.(2.10) for the advection of the volume fraction. Once the velocity field is established, a one-dimensional scheme is applied to advect the volume fraction field along each coordinate direction. Local volume fraction fluxes are computed based on local velocities, and the geometric reconstruction of the interface is accomplished within each cell by utilizing the volume fraction value specific to that cell. The interface in each cell is represented as a segment described by the equation $n_1 x_1 + n_2 x_2 = c$, where n_i represents the dimensionless partial derivative of C with respect to the corresponding spatial coordinate x_i , and c denotes the shortest distance from the segment to the current coordinate and the underlying coordinate system.

Chapter 3

Contact angle

3.1 Physics description

By definition, the contact angle, represented by the symbol C, is the angle that forms where the liquid interface meets the solid surface. To further elaborate, it represents the angle between the tangent at the liquid-vapor interface and the tangent at the solid-liquid interface at their point of intersection.
According to the Young Equation, the contact angle is defined through the analysis of the thermodynamic equilibrium among three phases: gaseous, liquid, and solid.
In particular, denoting with γ_{sg}, γ_{sl} e γ_{lg} the surface energy of the three interfaces, the Young Equation states:

$$\gamma_{sg} - \gamma_{sl} - \gamma_{lg}\cos(\theta_c) = 0. \tag{3.1}$$

However, a century afterward, Gibbs suggested an alteration to Young's equation. It intended to account for the influence of the droplet's volume on θ_c; it also manages to account for the excess energy at the interface of the three-phase boundary, by introducing a line tension k. The equation becomes as follow:

$$\cos(\theta) = \gamma_{sg} - \frac{\gamma_{sl}}{\gamma_{lg}} + \frac{k}{\gamma_{lg}\alpha} \tag{3.2}$$

It's notable that at micro-nano scales, even the modified Young's equation fails, as seen by empirical observations. For this thesis, the modified Young's equation will suffice. Our goal is to further explore the impact of varying contact angles on unsteady viscous liquid sheet flows.

3.2 Application of contact angle

The code utilizes the contact.h library from Basilisk, which is inspired by M. Renardy's work *et al.*[10] in turn. The library in question enables the use of height functions and VOF tracers to achieve the desired contact angle. To get a better a grasp of this concept, we need to introduce height functions, denoted by Hfs. Hfs can be used to calculate the curvature of the interface and its normal. The way in which it is done is by storing in the field the distance of the interface from the center of cell, and then using these values to obtain n and χ.

$$n = [h_x, -1] \tag{3.3}$$

$$\chi = h_{xx}/(1 + h_x^2)^{\frac{3}{2}} \tag{3.4}$$

where h_x and h_{xx} are obtained with a centered finite differences second order scheme. What the code does is modify the height-function so that the interface adopts a shape consistent with our specified contact angle.

As we'll observe, this field will be passed to tracers simply by setting the height variable of the tracers field.

The necessity for employing HFs stems from the fact that gradient-based methods used to determine the normal to the interface and the curvature (essentially $\nabla \cdot n = \chi$ and $\frac{\nabla \times f}{|\nabla \times f|} = n$) introduce spurious currents (the strength of which might overshadow any actual velocities) and, more critically, fail to converge with mesh refinement.

The normal and curvature play a crucial role in discretizing the equation:

$$F_{st} = \sigma \cdot K \cdot \delta \cdot n \tag{3.5}$$

In (3.5), F_{st} denotes the superficial tension force, and δ represents Dirac's Delta. This formulation was developed based on the CSF Model. Looking at the code, it is all quite straightforward:

```
1 //We define the field containing the height-function.
2 vector h[];
3
4 //We modify the height-function field using the $contact_angle()$ macro
```

```
   , thus imposing the contact angle.\\
h.t[left] = (fabs(y) >= my_H*0.5) ? contact_angle(theta0*pi/180.) : 0.;

//In our case, we are imposing only outside the fluid, so outside the
   width of the flow, which is my_H.
```

Listing 3.1: Contact Angle Implementation

The `contact_angle()` macro is defined in the contact.h library as follows:

```
#define contact_angle(theta)

(val(_s) == nodata ? nodata : val(_s) + (orientation(val(_s)) ? -1. :
   1.)/tan(theta));
```

Listing 3.2: Snippet from contact.h library

Where `_s` represents the state of the height-function for the cell. It can be either complete or assume any value between 0 and 1: it's considered complete if both cell ends are identified, otherwise it's zero or one if one end is located, and between zero and one if only the interface is detected.

If both ends of the cell are identified, this indicates that the cell is not situated at the interface, and the value of `_s` is assigned `nodata`. In this instance, in fact, it remains unchanged, and the height function is also set to `nodata`. Otherwise, once the orientation is considered, we add `+- 1/tan(theta)`, where 1 is the dimension of our mesh, given that all measurements are normalized.

Since the standard contact angle (without imposing any ourself) is set to 90 degrees, `h0` and `h1` would have been the same. By adding `1/tan(theta)` we're imposing that the first order derivative of Hf is exactly `tan(theta)` (in this case the ghost value beneath the surface line is set equal to the one above it). It can be easily understood by looking at the figure 3.1.

This is required only for the left boundary, of course, as that is where the contact angle is imposed.

To achieve the required angles (from 30 degrees to 150 degrees), defining only the tangential component of `h[]` suffices. For more acute angles, the normal component would also be necessary.

```
//We then have to pass the information to the tracers field.
```

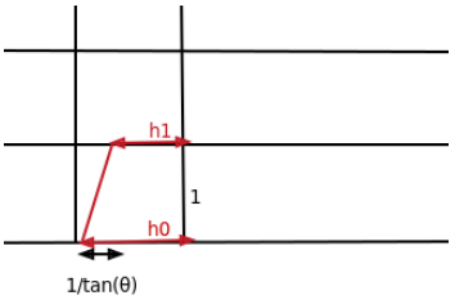
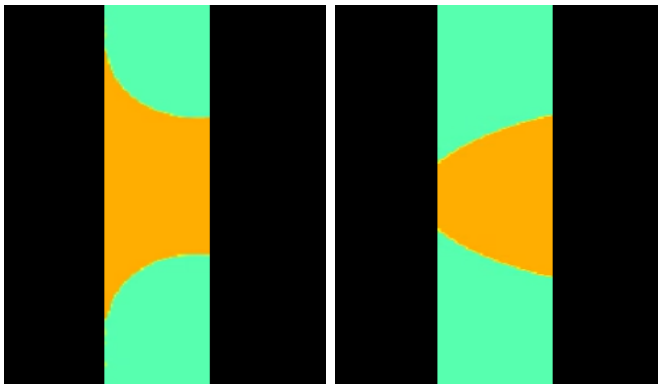


Figure 3.1: Height function on the left boundary

```
$f.height = h;$
```

Listing 3.3: Contact angle code

To further prove the presence of the contact angle, snapshots in fig.3.2 are taken from a configuration with gravity and initial velocity set to zero. The impacts in these cases are clearly visible in these snapshots, displaying contact angles of 30 and 135 degrees, respectively.



(a) 30 degree configuration (b) 135 degree configuration

Figure 3.2: Closer Examination of the Inlet Region in a Liquid Sheet with Different Contact Angles

3.2.1 Code

```
/*
Russo Andrea && Anastasio Domenico.
*/

//We're including all the libraries needed to solve the problem.
```

```
6  //The first one solves the incompressible, variable-density Navier-
       Stokes Equation.
7  #include "navier-stokes/centered.h"
8  //The second one lets us use a two-phase approach, usually used for two
       fluids separated by an interface.
9  #include "two-phase.h"
10 //The third one lets us account for surface tension.
11 #include "tension.h"
12 //The fourth one lets us know continuous information about the running
       solver.
13 #include "profile6.h"
14 //The fifth one lets us implement momentum-conserving VOF advection of
       the velocity component, needed for our two-phase solver.
15 #include "navier-stokes/conserving.h"
16 //The sixth one lets us implement the contact angle through a boundary
       condition as we'll see.
17 #include "contact.h"
18
19
20 //Defining problem variables.
21 #define my_rho_l 997.
22 #define my_rho_g 9.97
23 #define my_mu_g 0.0000184
24 #define my_sigma 0.00603
25 #define my_g 9.81
26 #define my_H 0.0015
27 #define my_L 0.075
28 #define my_U 0.492
29 #define my_r_rho (my_rho_g)/(my_rho_l)
30 #define my_r_mu (my_mu_g)/(my_mu_l)
31
32 //We define all the adimensional coefficients that will be needed.
33 #define my_RE (my_rho_l*my_U*my_H)/(2*my_mu_l)
34 #define my_WE (my_rho_l*pow(my_U,2)*my_H)/(2*my_sigma)
```

```
35 #define my_FR (pow(my_U,2))/(my_g*my_L)
36 #define my_eps (my_H)/(my_L)
37 #define my_p_ref my_rho_l*my_U*my_U
38
39 //This is the time at which we want the sinusoidal forcing at the inlet
       to start.
40 #define my_t_start 0.2
41
42 #define my_u_p 1.5*my_U
43
44 //This is the time at which the simulation will end.
45 #define my_tEnd 10.0
46
47 //These are the maximum and minimum level of refinement that will be
      used in the adapt_wavelet() function.
48 #define my_maxlev 11
49 #define my_minlev 5
50
51 //That's the power to which we will elevate 2, and that will be our
      initial grid size.
52 #define my_initlev 9
53 #define my_iter 5000
54 #define n_cells 512
55
56 //These are the amplitudes and the frequency of the sinusoidal forcing
      that will be applied at the inlet.
57 #define my_A 0.06
58 #define my_f 25
59
60 //This is the Strouhal number, which indicates the frequence of vortex
      shedding.
61 #define my_Stf (my_f*my_H)/my_U
62
63 //We define the strings containing the file's name.
```

```
64 char Nomefile3[50];
65 char Nomefile4[60];
66 char Nomefile5[60];
67 char Nomefile6[60];
68 char Nomefile7[60];
69 char Nomefile11[60];
70 char Nomefile20[60];
71 char Nomefile21[60];
72 char Nomefile22[60];
73 char Nomefile23[60];
74 char Nomefile24[60];
75 char Nomefile30[60];
76
77 //We define the strings containing the directories' name.
78 char Nomedir[60];
79 char Nomedir2[60];
80
81 //We initialize the field containing the height-function.
82 vector h[];
83 //We define the vectors containing all the viscosities and superficial
      tension coefficients for which we will iterate.
84 double mu_l_Vec[3] = {0.00089, 0.00032, 0.001};
85 double sigma_Vec[3] = {0.0725, 0.02, 0.04};
86
87 //We initialize the contact angle variable.
88 double theta0;
89
90 //We initialize the counter variables that will be used to iterate.
91 int k;
92 int z;
93
94 //All of these conditions are applied on the left, which should be the
      inlet, since the gravity is applied from left to right.
95 //We impose our boundary conditions.
```

```
96 p[left] = neumann(0.);
97
98 //We will impose parabolic (Poisseuille) flow along the x-directions,
      but only inside the liquid phase, so it will be between
99 //0.5*my_H and -0.5*my_H
100 u.n[left]  = fabs(y) <= 0.5*my_H ? dirichlet(-my_u_p*sq(y)/sq(0.5*my_H)
      +my_u_p) : dirichlet(0.);
101
102 //We will impose sinusoidal forcing at the inlet, but only inside the
      liquid phase.
103 u.t[left]  = fabs(y) <= 0.5*my_H ? dirichlet(my_A*my_U*sin(2.*pi*my_f*t
      )*(t > my_t_start)) : dirichlet(0.);
104
105 //We will impose that the liquid phase is only present between 0.5*my_H
       and -0.5*my_H, thus changing the value of the tracers to 1.
106 f[left]    = fabs(y) <= 0.5*my_H ? dirichlet(1.) : dirichlet(0.);
107
108 //On the right (which should be the lower edge), standard outflow
      conditions are imposed.
109 u.n[right] = u.n[] > 0. ? neumann(0.) : dirichlet(0.);
110 p[right]   = dirichlet(0.);
111 f[right] = neumann(0.);
112 u.t[right] = u.t[] > 0. ? neumann(0.) : 0.;
113
114 //For the right and left edges neumann conditions are imposed.
115 u.n[top] = neumann(0.);
116 u.t[top] = neumann(0.);
117 f[top] = neumann(0.);
118 p[top] = neumann(0.);
119 u.n[bottom] = neumann(0.);
120 u.t[bottom] = neumann(0.);
121 f[bottom] = neumann(0.);
122 p[bottom] = neumann(0.);
123
```

```
124 int main (int argc, char * argv[])
125 {
126
127   //We define the initial grid dimension.
128   init_grid (1 << my_initlev);
129
130   //We define the domain size.
131   size (my_L);
132
133   //We move the origin to the middle (along y-direction) of our box, on
        the left. The origin before was in the top left corner.
134   origin (0., -my_L/2);
135
136
137   //These are the densities that the solver will use for both of our
       phases.
138   rho1 = my_rho_l, rho2 = my_rho_g;
139   TOLERANCE = 1e-4;
140
141  //We make a for loop to solve for different contact angles, different
       liquid viscosities, and different superficial
142  //tension coefficients.
143  for (k = 0; k < 1 ; k++ )
144  for (z = 0; z < 1; z++ )
145  for (theta0 = 40; theta0 <= 140; theta0 += 100)
146  {
147   //The field containing the tracers will need the superficial tension
       coefficient to build the interface.
148   f.sigma = sigma_Vec[z];
149
150   //This is the viscosity the field will use.
151   mu1 = mu_l_Vec[k];
152
153   //We modify the height-function field using the contact_angle() macro
```

```
 , thus imposing the contact angle.
154 h.t[left] = (fabs(y) >= my_H*0.5) ? contact_angle(theta0*pi/180.) :
     0.;
155
156 //We then have to pass the information to the tracers field.
157 f.height = h;
158
159 //We give the two directories a name based on the Reynolds and Webers
     .
160 sprintf(Nomedir,"Re_%0.f_We_%0.f", my_H*my_rho_l*my_U/(mu_l_Vec[k]*2)
     ,(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]));
161 sprintf(Nomedir2,"Re_%0.f_We_%0.f/Snapshot_theta_%0.f", my_H*my_rho_l
     *my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]),
     theta0);
162
163 //We make the directories.
164 if (mkdir(Nomedir, 0777) == -1)
165       printf("Error");
166   else
167       printf("\nDirectory created\n");
168
169 if (mkdir(Nomedir2, 0777) == -1)
170       printf("Error");
171   else
172       printf("\nDirectory created\n");
173
174 //We make the names for the different files that are going to be
     created, the first six are a zoom at the inlet for each variable.
175 sprintf(Nomefile3,"Re_%0.f_We_%0.f/ux_jet_theta_%0.f_zoom.mp4",my_H*
     my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
     sigma_Vec[z]),theta0);
176 sprintf(Nomefile4,"Re_%0.f_We_%0.f/uy_jet_theta_%0.f_zoom.mp4",my_H*
     my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
     sigma_Vec[z]),theta0);
```

```
177   sprintf(Nomefile5,"Re_%0.f_We_%0.f/ux_theta_%0.f_zoom.mp4",my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
       sigma_Vec[z]),theta0);
178   sprintf(Nomefile6,"Re_%0.f_We_%0.f/uy_theta_%0.f_zoom.mp4",my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
       sigma_Vec[z]),theta0);
179   sprintf(Nomefile7,"Re_%0.f_We_%0.f/f_theta_%0.f_zoom.mp4",my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
       sigma_Vec[z]),theta0);
180   sprintf(Nomefile11,"Re_%0.f_We_%0.f/grid_%0.f.mp4",my_H*my_rho_l*my_U
       /(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]),theta0
       );
181   //We then name all the .mp4s that will contain the fields.
182   sprintf(Nomefile20,"Re_%0.f_We_%0.f/ux_jet_theta_%0.f.mp4",my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
       sigma_Vec[z]),theta0);
183   sprintf(Nomefile21,"Re_%0.f_We_%0.f/uy_jet_theta_%0.f.mp4",my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*
       sigma_Vec[z]),theta0);
184   sprintf(Nomefile22,"Re_%0.f_We_%0.f/ux_theta_%0.f.mp4",my_H*my_rho_l*
       my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]),
       theta0);
185   sprintf(Nomefile23,"Re_%0.f_We_%0.f/uy_theta_%0.f.mp4",my_H*my_rho_l*
       my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]),
       theta0);
186   sprintf(Nomefile24,"Re_%0.f_We_%0.f/f_theta_%0.f.mp4",my_H*my_rho_l*
       my_U/(mu_l_Vec[k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]),
       theta0);
187   sprintf (Nomefile30, "Re_%0.f_We_%0.f/", my_H*my_rho_l*my_U/(mu_l_Vec
       [k]*2),(my_rho_l*pow(my_U,2)*my_H)/(2*sigma_Vec[z]));
188
189   //We run the solver.
190   run();
191 }
```

```
}

//Event defining the initial conditions, t = 0 is the condition needed
   for the event to happen.
event init (t = 0) {
  if (!restore (file = "restart")) {

  //We use a foreach() to iterate on all the cells in our fields.
  foreach() {
   //We impose that the liquid phase is only present between 0.5*my_H
   and -0.5*my_H.
   f[] = fabs(y) <= 0.5*my_H ? 1. : 0.;
   //We impose parabolic (Poisseille) flow in the liquid phase, along
   the x direction.
   u.x[] = fabs(y) <= 0.5*my_H ? (-my_u_p*sq(y)/sq(0.5*my_H)+my_u_p) :
   0.;
   //We impose y-component equal to zero.
   u.y[] = 0.;
   p[] = 0;
            }

  //We use the boundary function to impose the BCs also on the fields
   defined in the CIs.
  boundary({f,u.x,u.y,p});

  //We then save the level field in our field l[].
  int n1=pow(2,my_maxlev);
  scalar l[];
  foreach()
  l[] = level;
  boundary({l});

  //We then define these new fields, only containing the liquid phase's
    velocities and pressure.
```

```
  scalar u_jet[],v_jet[],p_jet[],y_jet[];
  foreach() {
       u_jet[] = f[]*u.x[];
       v_jet[] = f[]*u.y[];
       p_jet[] = f[]*p[];
       y_jet[] = f[]*y;
            }
 //We apply boundaries on our newly defined fields.
boundary({u_jet,v_jet,p_jet,y_jet});

 //We put them all in a scalar list.
scalar * list = {f,u_jet,v_jet,p_jet,y_jet};

//We the initiate profiling for our solver.
char name[50];
sprintf (name, "1Dt_%g.dat",0*pow(my_tEnd,-1));
profile_equi(list, x,n_cells,fname = name);
profile(list, x,fname = name);
}
}

//This event ends the run.
event end(t = my_tEnd) {
    printf("\nThe run is finished.\n");
}

//This event will impose the contact angle at every timestep.
event contact (i++) {

//Just like in the main, we impose the contact_angle using the
   contact_angle() macro, which in turn modifies the height-function
//field accordingly.
h.t[left] = fabs(y) >= (0.5*my_H) ? contact_angle(theta0*pi/180.) : 0.;

```

```
253 //We print the theta variable, to know at which value we're iterating.
254 printf("\n%f\n",theta0);
255
256 //We then pass the information to the tracers field.
257 f.height = h;
258 }
259
260 //With this event we then introduce gravity, it is done at every
       timestep.
261 event acceleration (i++) {
262
263     //av[] is the vector that the navier-stokes solver will use to
       represent the acceleration. In this case, we just add 9.81
264     //every timestep.
265     face vector av = a;
266     foreach_face(x)
267     av.x[] += my_g;
268 }
269
270 //This event makes a logfile for each run.
271 event my_logfile (i++) {
272
273   //We define the file's name according to the Reynolds and the Webers.
274   FILE * p_file;
275   char Nomefile10[30];
276   sprintf(Nomefile10,"logfile_theta_%0.f_Re_%0.f.txt",theta0,my_H*
       my_rho_l*my_U/(mu_l_Vec[k]*2));
277
278   //We create a pointer to the file.
279   p_file = fopen(Nomefile10, "a+");
280
281   //We write inside the file all of the solver parameters.
282   fprintf(p_file, "%d %g %d %d %g %g %g %g %d %d %g %g \n",
283      i,dt, mgp.i,mgu.i,mgp.resb,mgu.resb,mgp.resa,mgu.resa,mgp.nrelax,
```

```
   mgu.nrelax,TOLERANCE,TOLERANCE/(dt*dt));
284  fclose(p_file);
285
286  //We print the timestep and the time at which we're at.
287 fprintf(stdout, "i = %d , t = %g\n", i, t);
288
289 }
290
291 //We initialize the variables and the fields that will be used inside
      the next event.
292 double du,dun;
293 scalar U_old[],U[];
294
295
296 //This event lets us check if there's convergence. It takes at each
      timestep the biggest variation of u.x[] and then puts it
297 //in a file.
298 event my_checkconv (i=0;i<=my_iter;i++) {
299 //We initialize the max variable and the udiff field.
300 double max = 0.;
301 scalar udiff[];
302
303 //We calculate the velocity's module for each cell.
304 foreach() {
305 U[] = sqrt(sq(u.x[])+sq(u.y[]));
306 }
307
308
309 //What's the reduction operator? It's an operator that lets us operate
      on various cells simultaneously (in parallel). It is needed
310 //because if, for example, we're looking for the max variable it can
      happen that a variable is accessed and written in
311 //simultaneously, invalidating the result of the operation. Reduction
      allows to choose the criteria with which we will choose which
```

```
312 //variable to actually write, if a cell is accessed simultaneously. For
        example, the + means that if two values are being written
313 //at the same time, what will be written inside the cell is the sum. In
        our case, with max, we will only pick the bigger value.
314
315 //The first argument is the criteria, after the : is the variable on
       which the reduction should be done.
316
317 foreach(reduction(max:max)) {
318
319 //We calculate the difference between the module now, and the module at
        the previous timestep.
320 udiff[] = fabs(U[] -U_old[]);
321
322 //We put this value inside ds.
323 double ds = fabs(U[] -U_old[]);
324
325 //We confront the value now obtained with the maximum value ever
       obtained. If it's bigger, then ds will be the new max.
326 if (ds > max && fabs(y) < 0.5*my_H)
327 max = ds;
328 }
329 boundary((scalar*){udiff});
330
331 //Now, the maximum variation was found, and it is put inside du.
332 du = max;
333
334 //We calculate the average variation.
335 dun = normf(udiff).avg;
336
337 //We now put the velocity field inside U_old[], so that at the next
       timestep the value will have changed.
338 foreach(reduction(max:max))
339 U_old[] = U[];
```

```
340
341 //We define the name of the file that will contain the maximum and the
       average variations for each timestep.
342 char Nomefile[30];
343 sprintf(Nomefile, "CheckConv_Theta_%0.f",theta0);
344
345 //We define the name of the file that will contain the parameters for
       each timestep.
346 char Nomefile2[30];
347 sprintf(Nomefile2, "Check_param_Theta_%0.f",theta0);
348
349 //We create pointers to the files.
350 FILE * fp_dvar = fopen(Nomefile, "a+");
351 FILE * fp_dpar = fopen(Nomefile2, "a+");
352
353 //Wr write inside the files.
354 fprintf (fp_dvar, "%d %g %g %g \n",i,t,du*pow(my_U,-1),dun);
355 fclose (fp_dvar);
356 fprintf (fp_dpar, "%g %g %g %g %g %g \n",my_r_rho,my_r_mu,my_RE,my_FR,
       my_WE,my_tEnd);
357 fclose (fp_dpar);
358
359 }
360
361 //This event produces the outputs.
362 event output (i++){
363
364 //It saves the grid in a field.
365  scalar l[];
366   int n1=pow(2,my_maxlev);
367   foreach()
368   l[] = level;
369   boundary({l});
370
```

```
371 //We redefine the fields only containing the liquid phase's velocities.
372 scalar u_jett[], v_jett[];
373 foreach(){
374   u_jett[] = f[]*u.x[];
375   v_jett[] = f[]*u.y[];
376 }
377
378 //We produce an .mp4 containing the grid.
379 output_ppm (l, file = Nomefile11);
380
381 //We produce all the .mp4s containing the fields, but there's a zoom at
       the inlet (to check for contact angle).
382 //The third option, linear = true, allows us to interpolate where the
       level isnt high enough.
383 output_ppm(u_jett, file = Nomefile3, linear = true, box = {{0,-1.3*my_H
       },{my_L/10, 1.3*my_H}});
384 output_ppm(v_jett, file = Nomefile4, linear = true, box = {{0,-1.3*my_H
       },{my_L/10, 1.3*my_H}});
385 output_ppm(u.x, file = Nomefile5, linear = true, box = {{0,-1.3*my_H},{
       my_L/5, 2*my_H}});
386 output_ppm(u.y, file = Nomefile6, linear = true, box = {{0,-2*my_H},{
       my_L/5, 2*my_H}});
387 output_ppm(f, file = Nomefile7, box = {{0,-2*my_H},{my_L/10, 2*my_H}});
388
389 //We produce all the .mp4s containing the fields.
390 output_ppm(u_jett, file = Nomefile20, linear = true);
391 output_ppm(v_jett, file = Nomefile21, linear = true);
392 output_ppm(u.x, file = Nomefile22, linear = true);
393 output_ppm(u.y, file = Nomefile23, linear = true);
394 output_ppm(f, file = Nomefile24, linear = true);
395
396 //If the stationary regime is reached (t > my_t_start) every 6000
       timestep it will take a snapshot.
397 if ((i % 6000) == 0 && t > my_t_start){
```

```
398 //We initialize the strings that will be used to give a name to the
      Snapshots.
399 char Nomefile31[100];
400 char Nomefile32[100];
401 char Nomefile33[100];
402 char Nomefile34[100];
403
404 //We put the name inside the strings.
405 sprintf (Nomefile31, "Snapshot_theta_%0.f/t_%f.dat", theta0, t);
406 strcpy (Nomefile32,Nomefile30);
407 strcat (Nomefile32,Nomefile31);
408
409 sprintf (Nomefile33,"time_theta_%0.f", theta0);
410 strcpy (Nomefile34,Nomefile30);
411 strcat (Nomefile34,Nomefile33);
412
413 //We create a pointer to the file.
414 FILE * fp_snap = fopen (Nomefile32, "w");
415 //We output the field.
416 output_field ({u.x, u.y, f, p}, fp_snap, n=n1, linear = true,box =
      {{0.0,-my_H},{my_L,my_H}});
417 fclose (fp_snap);
418
419 //We create the file containing the times at which the snapshots were
      taken.
420 FILE * fp_time = fopen(Nomefile34, "a+");
421 fprintf (fp_time, "%f\n", t);
422 fclose (fp_time);
423
424 }
425 }
426 //This event is needed to  adapt the grid to the tracer's variation.
      Basically we increase the resolution until the error
427 //is less than 1e-5.
```

```
428 event adapt (i++) {
429 //To do this, we used the adapt_wavelet function.
430 adapt_wavelet ({f}, (double[]){1e-5}, minlevel = my_minlev, maxlevel =
       my_maxlev);
431 adapt_wavelet ({u.x}, (double[]){1e-5}, minlevel = my_minlev, maxlevel
       = my_maxlev);
432 adapt_wavelet ({u.y}, (double[]){1e-5}, minlevel = my_minlev, maxlevel
       = my_maxlev);
433 //We refine our grid.
434 refine(x > -1. && fabs(y) < my_H && level < my_maxlev);
435 }
```

Listing 3.4: Implementation of Contact Angle

Chapter 4

Data-Driven Modal Analysis

4.1 Proper orthogonal decomposition

Proper orthogonal decomposition (POD) is a modal decomposition approach aimed at identifying an optimal collection of eigenfunctions, which form a spatial orthogonal basis. This technique offers an impartial algorithm for breaking down a dataset into the fewest possible basis functions or modes, aiming to capture the maximum energy. Consequently, it is utilized in creating succinct, low-dimensional representations of turbulent fluid flows.

In the context of applying POD to a fluid flow, we initiate the process with a vector field $\mathbf{Q}(\mathbf{x}, t)$ after subtracting its temporal mean $\overline{\mathbf{q}}(\mathbf{x})$. It is assumed that the unsteady component of the vector field can be decomposed as follows:

$$\mathbf{q}(\mathbf{x}, t) = \mathbf{Q}(\mathbf{x}, t) - \overline{\mathbf{q}}(\mathbf{x}) = \sum_{j=1}^{+\infty} a_j(t)\varphi_{\mathbf{j}}(\mathbf{x}) \tag{4.1}$$

where $\varphi_{\mathbf{j}}(\mathbf{x}, t)$ and $a_j(t)$ denote the modes and the time-dependent expansion coefficients, respectively. By removing the mean from the vector field, we observe how the flow field diverges from the average at each moment. Through POD, $\mathbf{q}(\mathbf{x}, t)$ is split into a series of spatially orthogonal modes $\varphi_{\mathbf{j}}$ (i.e., $\langle \varphi_{\mathbf{j}}, \varphi_{\mathbf{k}} \rangle_x = \delta_{jk}$) and temporal coefficients $a_j(t)$. The goal is to find the most effective way to reconstruct the vector field with the minimal number of methods.

First, we arrange the snapshots of the vector field into a column vector $\mathbf{q}(\mathbf{x}, t) = \mathbf{Q}(\mathbf{x}, t) - \overline{\mathbf{q}}(\mathbf{x})$, for $t = t_1, t_2, \ldots, t_m$, where m represents the number of snapshots. Once we've obtained the snapshot vector, it's practical to format our discrete data into a matrix:

$$\mathbf{Q} = \begin{bmatrix} | & | & & | & & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_k & \dots & \mathbf{q}_m \\ | & | & & | & & | \end{bmatrix}, \qquad \mathbf{Q} \in \mathbb{R}^{n \times m} \tag{4.2}$$

In this scenario, each column $\mathbf{q}_k$ represents our state vector for the respective snapshot k, encapsulating all components of the vector field $\mathbf{q}_i$ at each position $\mathbf{x}_j$. Using these snapshots, discrete Proper Orthogonal Decomposition (POD) modes and eigenvalues are derived by examining the covariance matrix $\mathbf{C}$, which is defined as follows:

$$\mathbf{C} = \frac{1}{m-1}\mathbf{Q}\mathbf{Q}^*, \qquad \mathbf{C} \in \mathbb{R}^{n \times n} \tag{4.3}$$

Here, the operator $*$ denotes the complex-conjugate operator. Solving the corresponding eigenvalue problem yields:

$$\mathbf{Q}\mathbf{Q}^* \varphi_k = \lambda_k \varphi_k \tag{4.4}$$

where the factor $\frac{1}{m-1}$ is incorporated into the eigenvalues λ_k. Clearly, the eigenvectors of the covariance matrix yield the optimal directions for representing variations in our vector state. Additionally, the symmetry of the matrix $\mathbf{Q}\mathbf{Q}^*$ ensures the orthogonality of the eigenvectors φ_k. By arranging the eigenfunctions φ_k into a matrix $\mathbf{\Psi}$ with dimensions $n \times n$, we formulate relation (4.1) as:

$$\mathbf{Q} = \mathbf{\Psi}\mathbf{a} \tag{4.5}$$

Since the autofunctions are mutually orthogonal, the $\mathbf{\Psi}$ matrix is both unitary and invertible. As a result, the time-dependent coefficients $\mathbf{a}$ can be calculated as:

$$\mathbf{a} = \mathbf{\Psi}^*\mathbf{Q} \tag{4.6}$$

As the eigenvalues λ_k from Eq. (4.4) are sorted in descending order, the POD modes illustrate their significance in capturing the flow field's kinetic energy. Thus, by selecting only $\mathbf{r}$ crucial modes, fluctuations in the flow field data can be represented using the relation:

$$\sum_{j=1}^{r} \lambda_j / \sum_{j=1}^{+\infty} \lambda_j \approx 1 \tag{4.7}$$

This simplifies Eq. (4.1) to:

$$\mathbf{q}(\mathbf{x}, t) = \sum_{j=1}^{r} a_j(t)\varphi_{\mathbf{j}}(\mathbf{x}) \tag{4.8}$$

In the discrete framework, the matrix $\mathbf{a}$ has dimensions $r \times m$ instead of $n \times m$, achieving dimensionality reduction.

4.1.1 Relationship between SVD and POD

When the spatial dimension of the data n is significantly large, the covariance matrix size $\mathbf{C} = \frac{1}{m-1}\mathbf{Q}\mathbf{Q}^*$ becomes extensive $(n \times n)$. Using the classical spatial Proper Orthogonal Decomposition (POD) method for searching the eigenfunctions becomes computationally inconvenient. In addressing this challenge, since POD decomposition is connected to Singular Value Decomposition (SVD), the issue can be resolved. The data matrix $\mathbf{Q} \in \mathbb{R}^{n \times m}$ can be decomposed as:

$$\mathbf{Q} = \mathbf{\Psi}\mathbf{\Sigma}\mathbf{\Phi}^* \tag{4.9}$$

Where $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$, and $\mathbf{\Phi} \in \mathbb{R}^{m \times m}$. The matrix $\mathbf{\Psi}$ holds the left eigenvectors of $\mathbf{Q}\mathbf{Q}^*$ as columns, matrix $\mathbf{\Phi}$ includes the right eigenvectors of $\mathbf{Q}^*\mathbf{Q}$ as columns, and the matrix $\mathbf{\Sigma}$ contains the singular values $(\sigma_1, \sigma_2, \ldots, \sigma_m)$ along the main diagonal, which correspond to the eigenvalues λ_k of Eq. (4.4) by $\sigma_k^2 = \lambda_k$. This implies that, especially in fluid dynamics applications, $n >> m$, making it computationally more feasible to compute eigenvectors and eigenvalues of $\mathbf{Q}^*\mathbf{Q}$ $(m \times m)$ rather than $\mathbf{Q}\mathbf{Q}^*$ $(n \times n)$. After $\mathbf{\Phi}$ and $\mathbf{\Sigma}$ are calculated for this smaller problem, by inverting the relation (**??**), $\mathbf{\Psi}$ can be easily reconstructed by

$$\mathbf{\Psi} = \mathbf{Q}\mathbf{\Phi}\mathbf{\Sigma}^{-1} \tag{4.10}$$

This method, using the SVD, is called the *method of snapshots*.

4.1.2 Utilization in Thin Liquid Sheet Scenarios

The POD method is used for particular configurations with thin liquid sheets. For analysis, it is essential to gather the disturbance components of velocity, labeled U and V, and changes in volume fraction, marked as C. Specifically, for each snapshot, a column state vector is formed,

covering every point on the numerical grid. The following snapshots are then obtained:

$$u = U - \overline{U}$$
$$v = V - \overline{V}$$
$$c = C - \overline{C}$$

and then they are stacked in the following matrix:

$$\mathbf{Q} = \left[\begin{array}{ccccccc} u_{11}^1 & u_{11}^2 & \cdots & u_{11}^k & \cdots & u_{11}^m \\ u_{21}^1 & u_{21}^2 & \cdots & u_{21}^k & \cdots & u_{21}^m \\ | & | & & | & | & \\ u_{n_y1}^1 & u_{n_y1}^2 & \cdots & u_{n_y1}^k & \cdots & u_{n_y1}^m \\ | & | & & | & | & \\ u_{n_yn_x}^1 & u_{n_yn_x}^2 & \cdots & u_{n_yn_x}^k & \cdots & u_{n_yn_x}^m \\ \hline v_{11}^1 & v_{11}^2 & \cdots & v_{11}^k & \cdots & v_{11}^m \\ v_{21}^1 & v_{21}^2 & \cdots & v_{21}^k & \cdots & v_{21}^m \\ | & | & & | & | & \\ v_{n_y1}^1 & v_{n_y1}^2 & \cdots & v_{n_y1}^k & \cdots & v_{n_y1}^m \\ | & | & & | & | & \\ v_{n_yn_x}^1 & v_{n_yn_x}^2 & \cdots & v_{n_yn_x}^k & \cdots & v_{n_yn_x}^m \\ \hline c_{11}^1 & c_{11}^2 & \cdots & c_{11}^k & \cdots & c_{11}^m \\ c_{21}^1 & c_{21}^2 & \cdots & c_{21}^k & \cdots & c_{21}^m \\ | & | & & | & | & \\ c_{n_y1}^1 & c_{n_y1}^2 & \cdots & c_{n_y1}^k & \cdots & c_{n_y1}^m \\ | & | & & | & | & \\ c_{n_yn_x}^1 & c_{n_yn_x}^2 & \cdots & c_{n_yn_x}^k & \cdots & c_{n_yn_x}^m \end{array}\right], \qquad \mathbf{Q} \in \mathbb{R}^{N \times m} \tag{4.11}$$

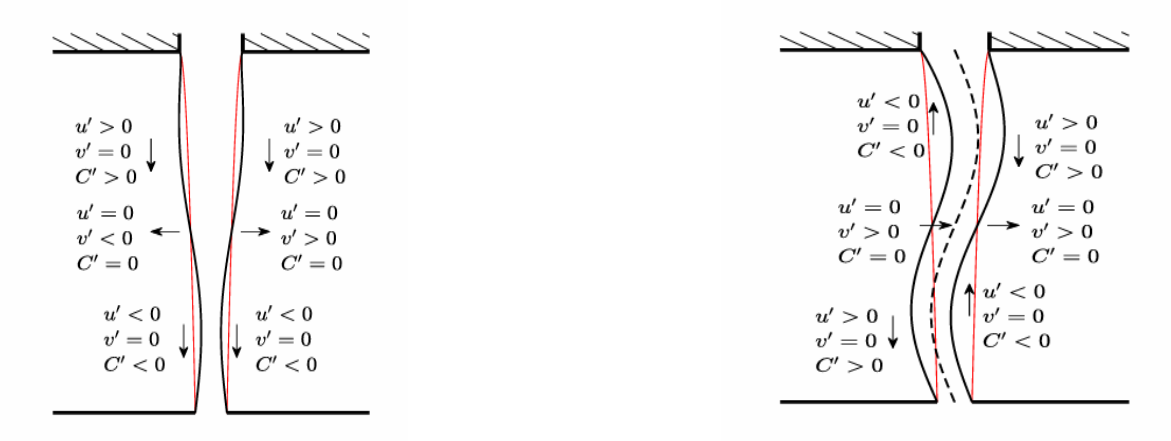
where $N = 3 \times n_x \times n_y$ and m is the number of the snapshots.

Chapter 5

Investigation and Results

The paper intends to analyze a flow field characterized by a Weber number ($We = \frac{\rho_l U^2 H}{2\sigma} = 0.75$) and a Reynolds number ($\mathrm{Re} = \frac{\rho U L}{\mu} = 1635$). The velocity profile's behavior is influenced by the magnitude of the Weber number. When $We > 1$, the flow velocity is considered supercritical; when $We < 1$, as in our study, it is subcritical. In this subcritical domain, the investigation shows a mixed sinuous-varicose motion pattern, affected by changes in the Reynolds number. Notably, it is also the high Reynolds number that contributes to the appearance of varicose modes, as shown in Figure (5.1b).

Utilizing the Proper Orthogonal Decomposition (POD) technique on liquid sheets verifies the dominance of varicose traits in the fluid movement. This method facilitates the detection and reconstruction of critical elements, offering a highly precise approximation of the real flow field.



(a) Sinuous modes behavior

(b) Varicose modes behavior

Figure 5.1: Fluid flow behavior

The tables (5.1) and (5.2) show the numerical data used in MATLAB codes.

Name	Parameter	Supercritical regime
Liquid density	ρ_l	$997\ Kg/(m^3)$
Liquid viscosity	μ_l	$2.25 \times 10^{-4}\ Kg/(ms)$
Gas density	ρ_g	$9.97\ Kg/(m^3)$
Gas viscosity	μ_g	$1.84 \times 10^{-5}\ Kg/(ms)$
Surface tension	σ	$0.241\ N/m$
Gravity acceleration	g	$9.81\ m/(s^2)$
Inlet mean velocity	U_i	$4.92 \times 10^{-1}\ m/s$
Inlet sheet thickness	H_i	$1.50 \times 10^{-3}\ m$
Liquid sheet length	L	$7.50 \times 10^{-2}\ m$

Table 5.1: Supercritical regime data

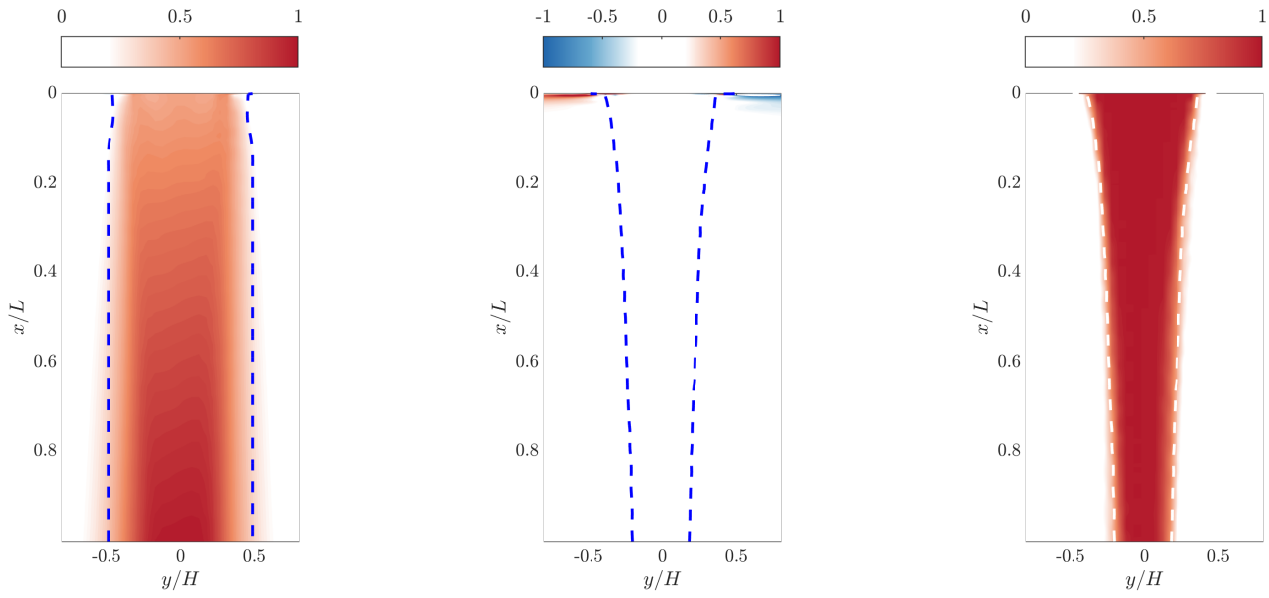
Name	Parameter	Supercritical regime
Density ratio	$r_\rho = \rho_g/\rho_l$	0.01
Viscosity ratio	$r_\mu = \mu_g/\mu_l$	8.17×10^{-2}
Slenderness ratio	$\epsilon = H_i/L$	0.02
Reynolds number	$\mathrm{Re} = \dfrac{\rho_l U_i H_i}{2\mu_l}$	1635
Froude number	$\mathrm{Fr} = \dfrac{U_i^2}{gL}$	3.29×10^{-1}
Weber number	$\mathrm{We} = \dfrac{\rho_l U_i^2 H_i}{2\sigma}$	0.75

Table 5.2: Supercritical regime dimensionless data

5.1 Subcritical Regime

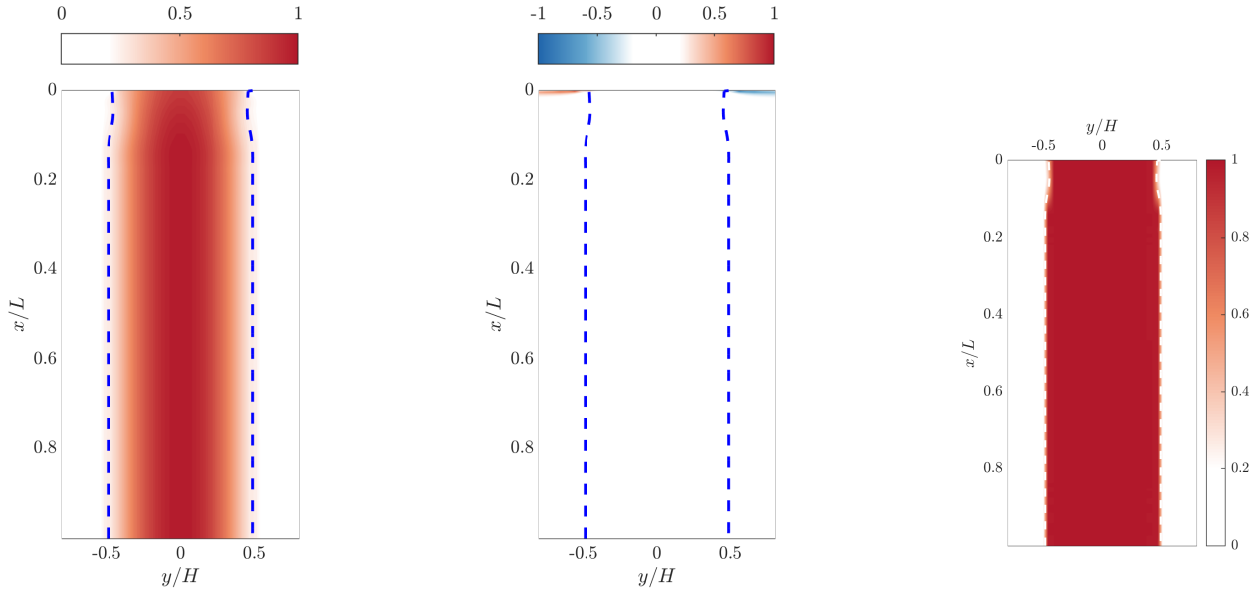
In this section, we explore the **Subcritical Regime** (We $<$ 1) and we further delve into what was previously said. In this setting, we see a combined sinuous-varicose motion, the predominance of which hinges on changes in the Reynolds number. At lower Re values, sinuous motion prevails, whereas at higher Re values, varicose motion is more evident (as observed by Colanera *et al.* [5]). For the purposes of this investigation, we focus solely on the case where Re = 1635.

Prior to implementing the POD decomposition technique, we showcase the mean motion fields and the 100th snapshot of U, V, and C to offer insight into fluid flow patterns, as illustrated in Figures 5.2 and 5.3 respectively:



(a) U mean (b) V mean (c) C mean

Figure 5.2: Mean values of the flow field parameters U,V, and C in subcritical regime for We = 0.75 and Re = 413



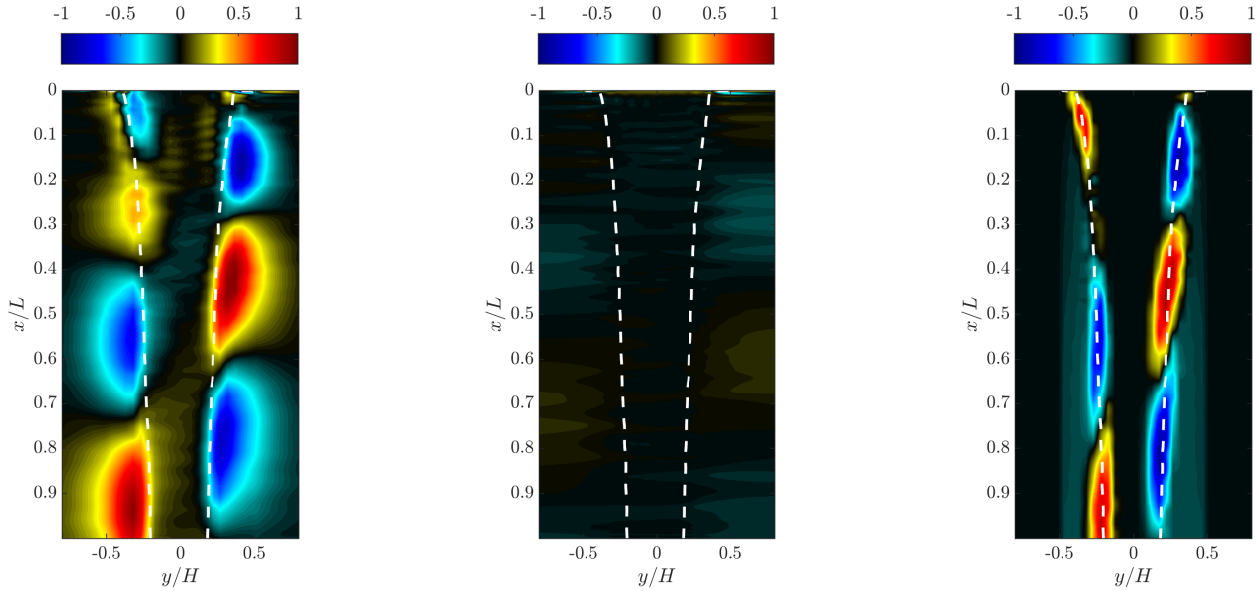
(a) Snapshot U (b) Snapshot V (c) Snapshot C

Figure 5.3: 100th snapshot of U,V, and C in subcritical regime for We $= 0.75$ and Re $= 1635$

Upon increasing the Reynolds number to 1635 (in comparison with Colanera's findings which implements a lower Reynolds [5]), significant changes occur in the fluid flow behavior, with the varicose component becoming more prominent.

After applying the POD decomposition technique, the primary modes of the velocity components (u, v) and of the volume fraction c are shown in Figure (5.4).

From the graphical representation, similarly to the supercritical case, we can observe that in mode 1, u exhibits antisymmetric behavior, v shows symmetry, and c displays antisymmetry.



(a) u mode 1 (b) v mode 1 (c) c mode 1

Figure 5.4: Mode 1 of u,v, and c in subrcritical regime for We = 0.75 and Re = 1635

5.1.1 Symmetric and antisymmetric decomposition

After applying POD in the subcritical case for Re = 1635, it can be observed that the first mode (ϕ_1) of velocity (u, v) and concentration (c) does not display any distinct symmetries. However, it is possible to decompose each component into its symmetric and antisymmetric parts. In general, any mode ϕ_j can be decomposed as follows:

$$\phi_j = \phi_j^s(x) + \phi_j^v(x) \tag{5.1}$$

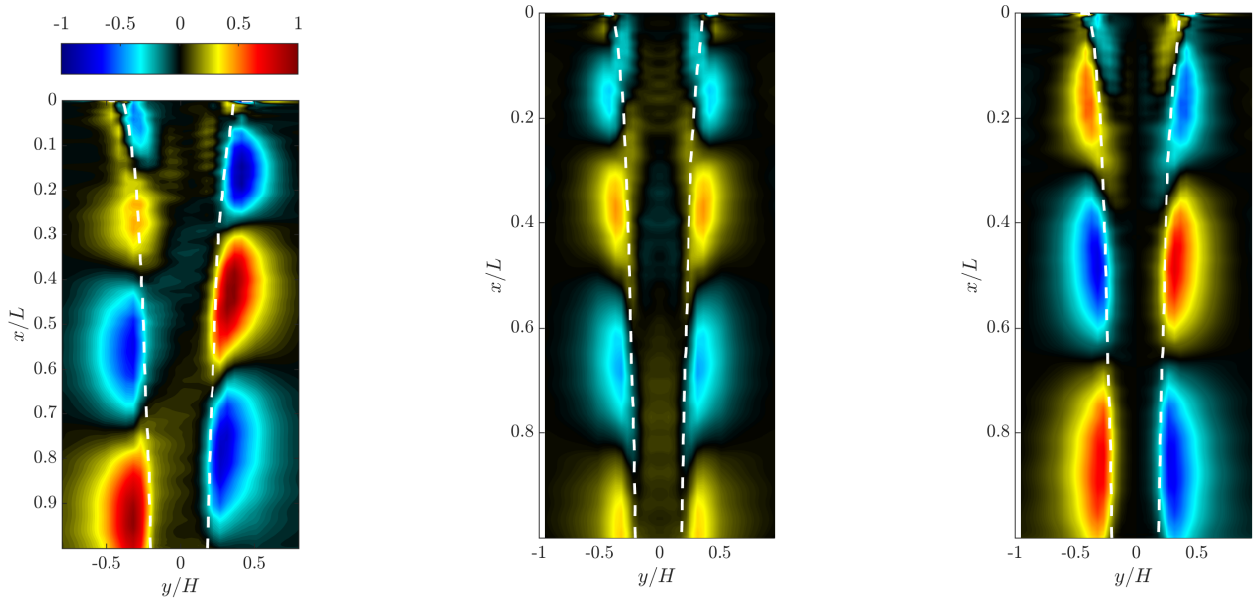
where ϕ_j^s and ϕ_j^v in eq:5.1 are defined as:

$$\phi_j^s(x,y) = [\phi_j(x,y) - \phi_j(x,-y)]/2 \tag{5.2}$$

$$\phi_j^v(x,y) = [\phi_j(x,y) + \phi_j(x,-y)]/2 \tag{5.3}$$

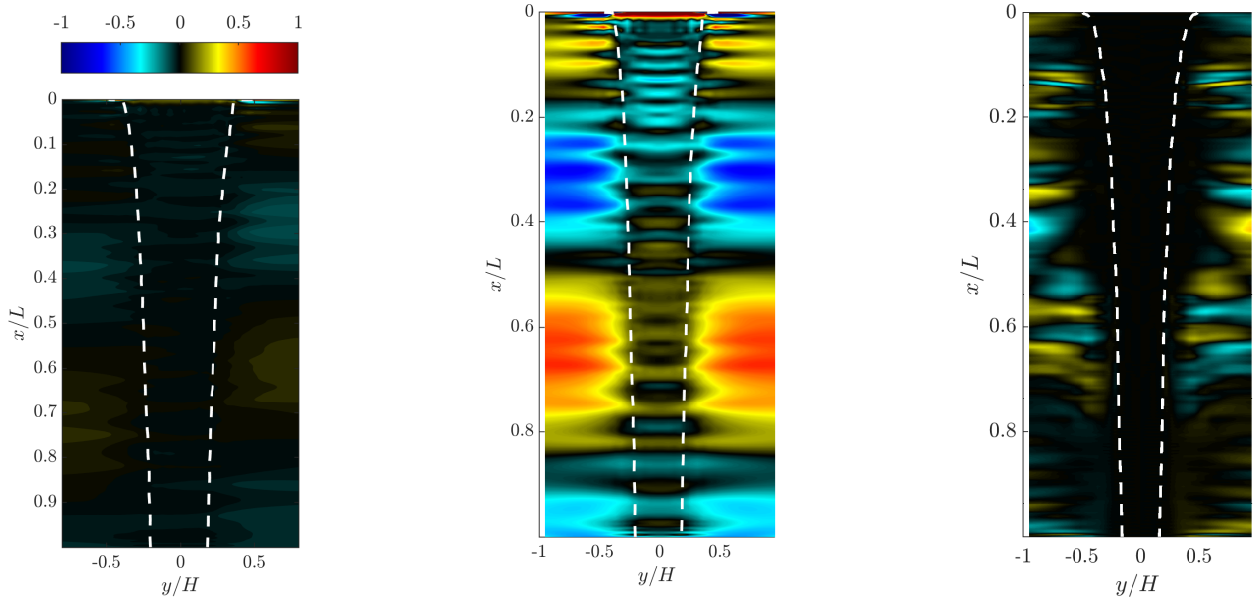
We represent u's decomposition in figure (5.5), v's decomposition in figure (5.6) and c's

decomposition in figure (5.7).



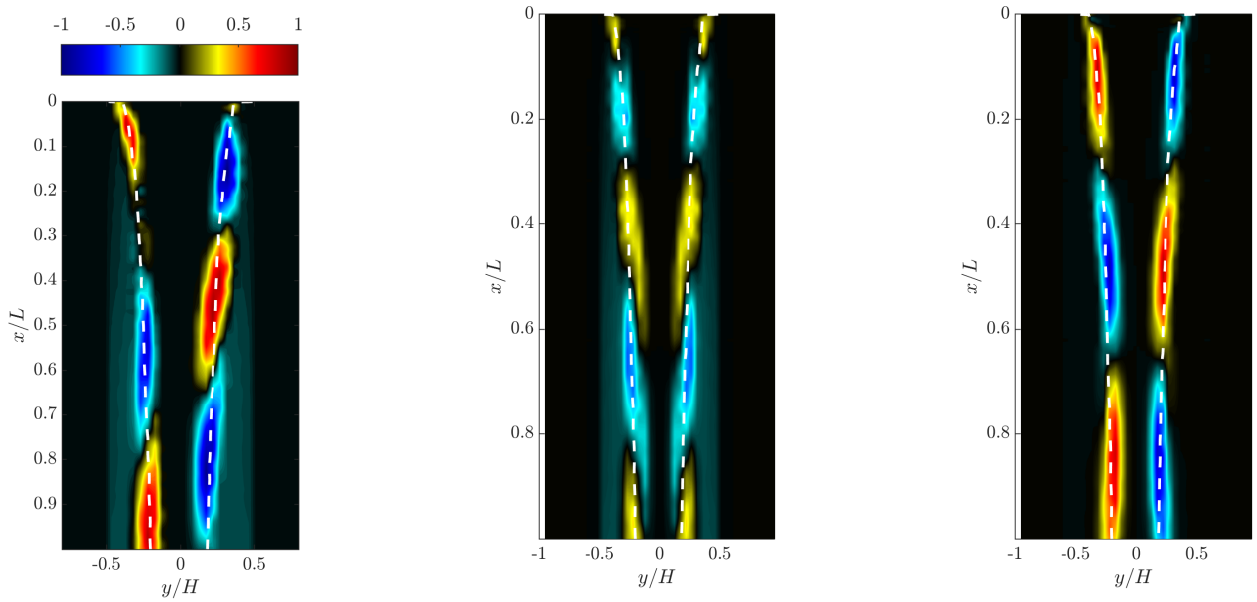
(a) u mode 1 (b) Symmetric u (c) Antisymmetric u

Figure 5.5: Decomposition of mode 1 of u, case subcritical regime for Re = 1635



(a) v mode 1 (b) Symmetric v (c) Antisymmetric v

Figure 5.6: Decomposition of mode 1 of v, case subcritical regime for Re = 1635



(a) c mode 1 (b) Symmetric c (c) Antisymmetric c

Figure 5.7: Decomposition of mode 1 of c, case subcritical regime for Re = 1635

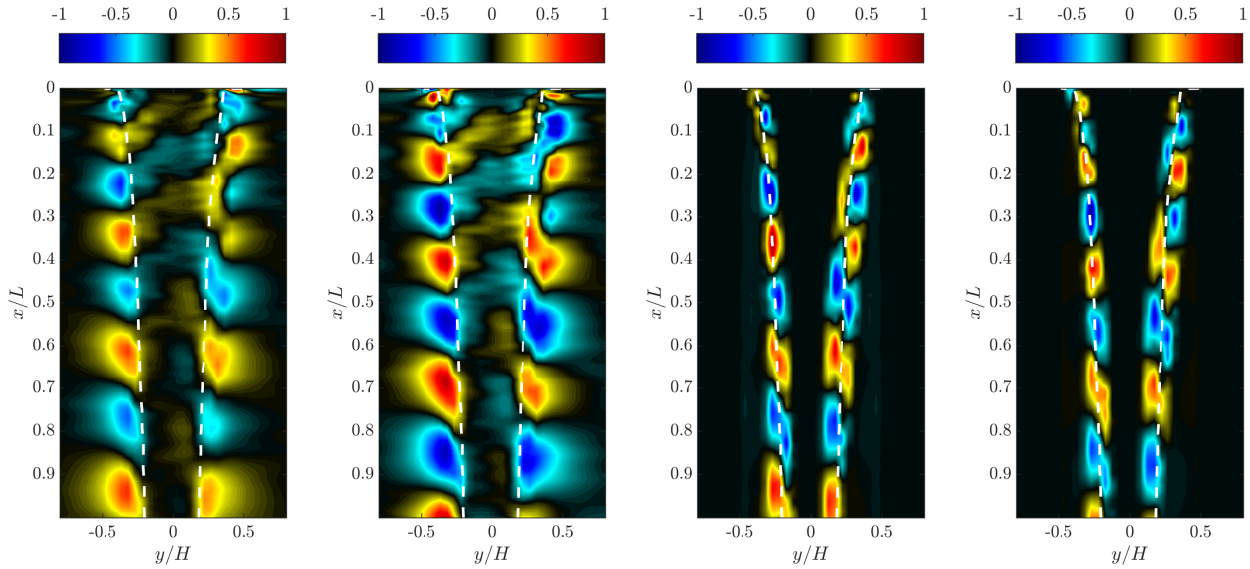
It is evident (from panels 5.5, 5.6 and 5.7) that each component shows both symmetry and asymmetry, indicating a mix of varicose and sinuous behavior.

The emergence of the varicose component results from nonlinear coupling under resonance conditions, where the sheet is driven by a frequency matching its principal natural frequency, aided by low Weber numbers. As noted in [5], at the lowest forcing frequency, the varicose contribution becomes increasingly significant (in terms of energy) as the Weber number (We) decreases, peaking at $We = 0.75$, while the sinuous contribution rises with We.

The excitation of the varicose mode with decreasing Weber numbers, along with a gradual shift from higher harmonics to the principal frequency, can be seen as a precursor to the liquid sheet's rupture. This phenomenon is experimentally observed when We is reduced by progressively lowering the inlet flow rate (e.g., de Luca and Meola [6]). Le Grand-Piteira *et al.* [9] have similarly suggested that the progressive thinning of the sheet observed as the inlet Weber number decreases into the subcritical regime may be one of the mechanisms leading to rupture in a three-dimensional liquid curtain configuration.

Furthermore, by analyzing phase shifts in the modes, a half or full wavelength is evident in modes 4 and 5, as shown in Figure 5.8. This indicates the presence of a traveling wave, with the phase shift showing the wave's propagation direction. This hypothesis can be confirmed by simply analyzing the frequency content of said modes. Indeed, they show a continuous spectrum

of frequencies, as depicted in Figure 5.9, with the frequencies of modes 4 and 5 highlighted in magenta and yellow, respectively.



(a) u mode 4 (b) u mode 5 (c) c mode 4 (d) c mode 5

Figure 5.8: Mode 4 and 5 of u and c in subcritical regime for We $= 0.75$ and Re $= 1635$

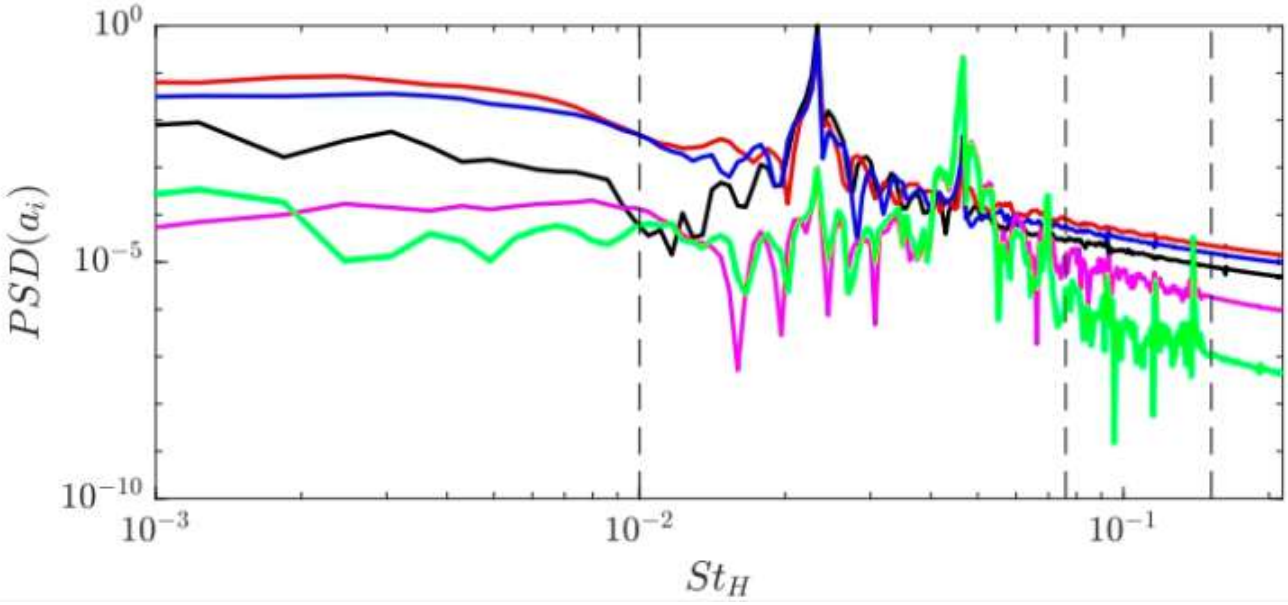
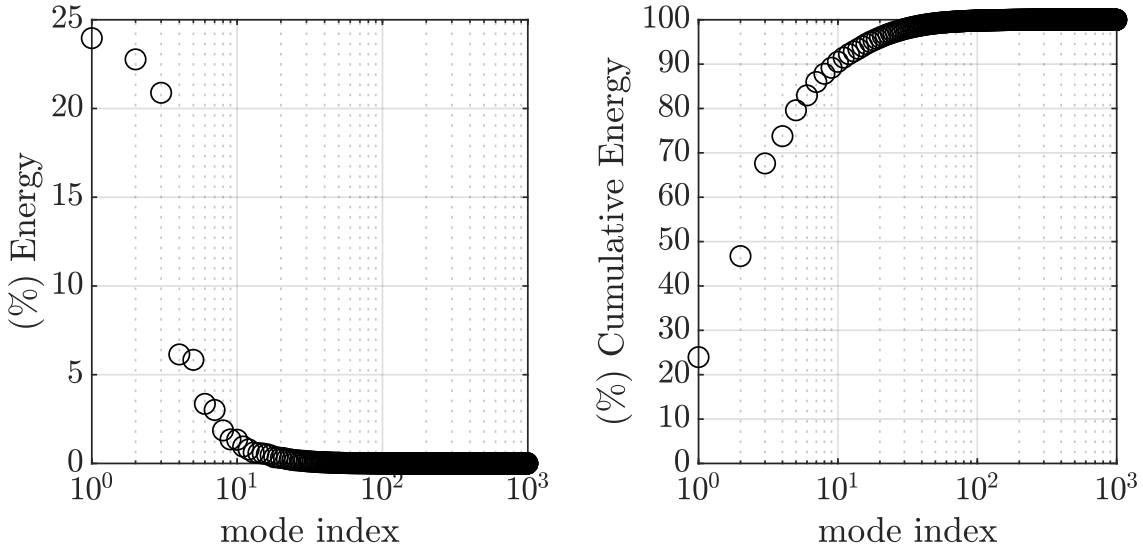


Figure 5.9: Power Spectral Density of some selected a_j for $We = 0.75$ and $Re = 1635$ *(mode's color are black, red, blue, magenta and green respectively for the modes from 1 to 5)*

5.1.2 Mode's energy and frequency analysis

TThe advantage of using Proper Orthogonal Decomposition (POD) lies in its capacity to simplify flow field reconstruction by selecting only a subset of modes. Essentially, this method streamlines the problem by allowing the selection of modes based on their significance in capturing the kinetic energy of the flow field. This flexibility makes the reconstruction process more efficient, enabling a focused analysis. This characteristic is demonstrated in the following figures, where the POD modes are plotted against energy (represented by dimensionless eigenvalues) in Figure 5.11a, and the cumulative energy ratio (ER) in Figure 5.11b, defined as:

$$ER = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{m} \sigma_j^2} \tag{5.4}$$



(a) Dimensionless POD eigenvalues

(b) Cumulative eigenvalues ratio

Figure 5.10: Kinetic Energy in subcritical regime for Re = 1635

Since the eigenvalues are sorted from largest to smallest, using the Proper Orthogonal Decomposition (POD) technique ensures that the first modes capture the most energy, as shown in Figure 5.11a. Remarkably, the initial 15 modes collectively account for approximately 99% of the total kinetic energy, as illustrated in Figure 5.11b. Notably, the kinetic energy associated with the POD modes beyond the initial ones declines rapidly.

Hence, if the goal is to select r modes for motion field reconstruction, choosing the foremost modes while excluding the later ones ensures an accurate approximation without significant in-

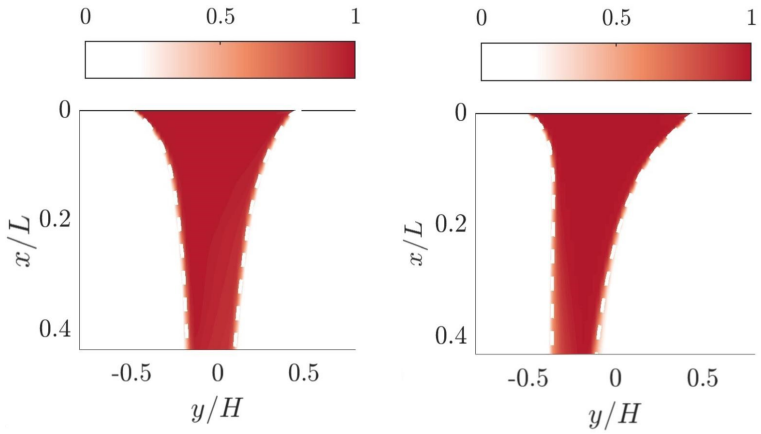
formation loss. Furthermore, when focusing on the initial modes, careful selection is nonetheless essential for reconstructing the motion field. This is evident from Figure 5.9, where although Mode 1 exhibits higher energy than Mode 5, the latter dominates at lower frequencies. Consequently, excluding Mode 5 from the reconstruction leads to significant information loss at lower frequencies. Therefore, the mode selection process is also guided by frequency analysis.

5.2 Result on contact angle

In this paper, we also sought to explore potential differences compared to the standard application by introducing a contact angle, denoted as θ_c, at the inlet. We established two configurations based on the value of θ_c: one set to 40° and the other to 140°. The parameters and configurations remained consistent with those examined in Section 5.1. We expected to observe variances between the 40-degree and 140-degree configurations, especially in the inlet profiles.

For the 40-degree configuration, we anticipated the flow cross-sectional area to initially decrease before stabilizing to the standard configuration observed previously. Conversely, for the 140-degree configuration, we expected an initial expansion of the flow section due to the larger angle. However, these modifications resulted in minimal changes to the liquid sheets, as shown in Figure (insert figure numbers), which presents snapshots of the C field. Remarkably, no significant alterations were observed.

The absence of substantial changes indicates one of two possibilities: either the impact of the contact angle is significantly smaller than the gravitational effects, or the expected variations are highly localized. The latter suggests the need for a finer grid and a more detailed investigation of the inlet region.



(a) 40 degree configuration (b) 140 degree configuration

Figure 5.11: Closer look to C mean snapshot in configuration with contact angle

Chapter 6

Conclusion

The dynamics of a gravitational liquid sheet, perturbed by sinuous disturbances at the inlet section, has been successfully investigated through the careful application of the Proper Orthogonal Decomposition (POD) modal decomposition technique.

The investigation of the flow field at high Reynolds numbers in subcritical conditions has uncovered a sinuous-varicose behavior, representative of a traveling motion phenomenon. Applying the POD technique to the snapshots has enabled a decomposition of the analyzed flows, revealing various spatial structures (modes) each governed by temporal coefficients. Notably, the initial modes contain the majority of the energy, allowing for accurate reconstruction of the flow field.

Furthermore, applying the Fourier transform to the temporal coefficients has allowed a frequency domain analysis, confirming prior energy assessments and highlighting the need to carefully select the number of modes for reconstruction to prevent information loss across various frequency spectrum regions.

Subsequent reconstructions of the flow field with a different number of modes have shown better results with larger mode sets, underscoring the importance of mode selection. Furthermore, quantitative assessment via reconstruction errors has offered additional insights into the effectiveness of different configurations.

This investigation demonstrates the effectiveness of the POD decomposition technique in analyzing liquid sheets across various flow regimes, providing computational efficiency without accuracy losses.

Its applicability extends beyond this study to cover numerous aerospace phenomena, including Von Kármán shedding and Kelvin-Helmholtz instability.

In conclusion, the POD modal decomposition technique emerges as a pivotal tool, capable of delivering robust reconstructions with computational efficiency across a spectrum of complex aerospace phenomena.

Regarding the configuration with the contact angle, it is concluded that the snapshots do not show significant differences from the standard configuration. This may be due to the effects being of a smaller magnitude and not visually detectable. However, further investigation of this setup is needed, possibly through higher grid resolution and a more detailed examination of the inlet region.

Bibliography

[1] S Afkhami and M Bussmann. Height functions for applying contact angles to 2d vof simulations. *International journal for numerical methods in fluids*, 57(4):453–472, 2008.

[2] Ashish Arote, Mukund Bade, and Jyotirmay Banerjee. Numerical investigations on stability of the spatially oscillating planar two-phase liquid jet in a quiescent atmosphere. *Physics of Fluids*, 31(11), 2019.

[3] Ashish Arote, Mukund Bade, and Jyotirmay Banerjee. On coherent structures of spatially oscillating planar liquid jet developing in a quiescent atmosphere. *Physics of Fluids*, 32(8), 2020.

[4] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[5] Antonio Colanera, Alessandro Della Pia, Matteo Chiatto, Luigi de Luca, and Francesco Grasso. Modal decomposition analysis of unsteady viscous liquid sheet flows. *Physics of Fluids*, 33(9), 2021.

[6] Luigi De Luca and Carosena Meola. Surfactant effects on the dynamics of a thin liquid sheet. *Journal of Fluid Mechanics*, 300:71–85, 1995.

[7] Alessandro Della Pia, Matteo Chiatto, and Luigi de Luca. Global eigenmodes of thin liquid sheets by means of volume-of-fluid simulations. *Physics of Fluids*, 32(8), 2020.

[8] Alessandro Della Pia, Matteo Chiatto, and Luigi de Luca. Receptivity to forcing disturbances in subcritical liquid sheet flows. *Physics of Fluids*, 33(3), 2021.

[9] N Le Grand-Piteira, Philippe Brunet, Luc Lebon, and Laurent Limat. Propagating wave pattern on a falling liquid curtain. *Physical Review E*, 74(2):026305, 2006.

[10] Michael Renardy, Yuriko Renardy, and Jie Li. Numerical simulation of moving contact line problems using a volume-of-fluid method. *Journal of Computational Physics*, 171(1):243–263, 2001.

[11] Kunihiko Taira, Steven L Brunton, Scott TM Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. Modal analysis of fluid flows: An overview. *Aiaa Journal*, 55(12):4013–4041, 2017.

[12] J Antoon Van Hooft, Stéphane Popinet, Chiel C Van Heerwaarden, Steven JA Van der Linden, Stephan R De Roode, and Bas JH Van de Wiel. Towards adaptive grids for atmospheric boundary-layer simulations. *Boundary-layer meteorology*, 167:421–443, 2018.

Acknowledgments

I would like to thank my mentors Matteo Chiatto and Antonio Colanera for guiding me through this not at all easy but definitely fun work.