Imperial College London

MEng Aeronautical Engineering

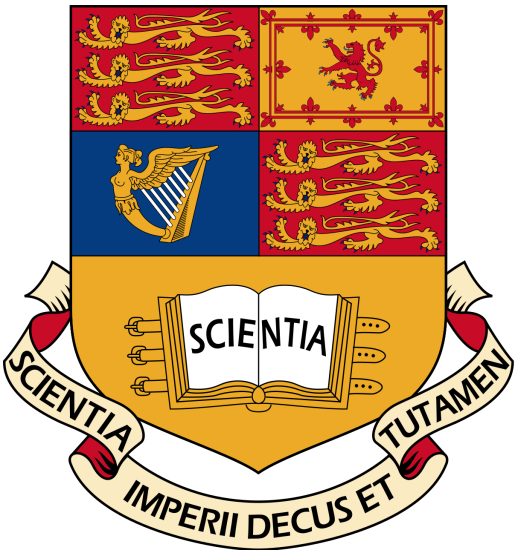
with a Year in Industry H420

Absolute and Convective Instabilities in Stratified Parallel Wakes

Author:
Lloyd Sun-lloyd Fung
(00734032)

Supervisor:
Dr. YongYun Hwang

29th May 2017



Abstract

Recent research by Meunier (2012) has discovered a surprising new mode of instability in the wake flow after a cylinder in a density-stratified medium. Such mode only exhibits its instability at a low Froude number when the cylinder wake (hence the shear) is not aligned with the stratification. It destabilises when stratification strength increases, contradicting conventional believes that stratification stabilises the flow. This project, therefore, set out to explain such phenomenon based on a local linear stability analysis with a parallel wake assumption. The wake profile is taken from Monkewitz (1988), and the linearised governing equations are solved numerically using the Chebyshev Collocation Method for discretisation. A MATLAB solver script was written for such purpose. Although the analysis result shows trends consistent with the experiment when the shear is aligned with the stratification, the new mode was not found under the current wake profile assumption. It is believed that the strong stratification has modified the wake profile significantly such that the new mode is destabilised, but a changing wake profile is outside the scope of this project. Further investigations should be done to confirm the hypothesis that the changing wake profile destabilises the flow at a low Froude number.

Contents

1 Introduction **4**
- 1.1 Potential Application . . . 4
- 1.2 Knowledge Gap . . . 4

2 Literature Review **6**
- 2.1 Theory on Hydrodynamic Stability . . . 6
- 2.2 Theory on Absolute and Convective Instability . . . 6
  - 2.2.1 Mathematical Definition for Absolute and Convective Instability . . . 7
- 2.3 Past work on stability of wake flow . . . 8
- 2.4 Past work on stratified wake flow and stability of stratified flow . . . 8
- 2.5 Recent work on cases when stratification is not align with shear . . . 9

3 Project Aim, Context and Limit **11**
- 3.1 Parallel, linear and 2D assumption . . . 11
- 3.2 Project aim . . . 11

4 Formulation **12**
- 4.1 Coordinate system . . . 12
- 4.2 Non-dimensionalisation and linearisation . . . 12
  - 4.2.1 Non-dimensionalisation . . . 13
  - 4.2.2 Linearisation and parallelisation . . . 13
    - 4.2.2.1 Wake Profile . . . 13
    - 4.2.2.2 Density Profile . . . 14
    - 4.2.2.3 Applying linearisation and setting up modal solution . . . 15
  - 4.2.3 Criteria for Absolute and Convective Instability in terms ω . . . 16
  - 4.2.4 Simplifying the system of equations . . . 16
    - 4.2.4.1 Boussinesq Approximation . . . 17
    - 4.2.4.2 Coupled equations in matrix form with Boussinesq Approximation 18
    - 4.2.4.3 2D Assumption . . . 18
  - 4.2.5 Non-dimensionalisation by Stratification Length and Recovery of Taylor-Goldstein equation . . . 19

5 Numerical Method **20**
- 5.1 Overall architecture . . . 20
- 5.2 Script Component Description . . . 20
  - 5.2.1 Solver and Initialisation . . . 20
    - 5.2.1.1 Discretisation by Chebyshev Collocation Method and transformation 20
    - 5.2.1.2 Boundary Condition . . . 21
    - 5.2.1.3 The Solver and separating the generation of matrices into the Initialisation step . . . 21
  - 5.2.2 Search Method . . . 21
    - 5.2.2.1 Finding Convective Instability . . . 21
    - 5.2.2.2 Finding Absolute Instability . . . 22

5.2.2.3 Finding Neutral Curve . . . 22
5.2.3 Track Visualisation and Database Management . . . 23
5.2.3.1 Tracking Algorithm . . . 23
5.2.3.2 Database Management and Visualisation . . . 23
5.3 Validation . . . 23
5.3.1 Validating basic framework and initialisation component . . . 23
5.3.2 Validating transformation, wake flow profile and search method . . . 24
5.3.3 Validating the solver with Stratification and Viscosity . . . 24

6 Results and Discussion **27**
6.1 Understanding different branches at $\theta = 90°$ at High Froude number . . . 27
6.2 Decreasing Froude number with $\theta = 90°$ . . . 29
6.2.1 Eliminating w modes . . . 30
6.2.2 Branches description . . . 30
6.2.3 General Effect of Re . . . 31
6.2.4 General Effect of Fr . . . 32
6.2.5 Absolutely Unstable Modes . . . 32
6.2.5.1 Comparison of St number . . . 32
6.2.5.2 Eigenfunction . . . 33
6.2.6 Neutral Curve of Convectively Unstable Mode and Howard's 1/4 criterion . 35
6.3 Decreasing Froude number with $\theta = 30°$ . . . 36
6.3.1 Branches description . . . 36
6.3.2 Finding the AU branch at centre . . . 38

7 Conclusion and Future Work **41**

A Script Architecture and Interface **44**

B The MATLAB Solver Script **46**

List of Figures

2.1 Linear response of a disturbance of a) a Linearly Stable flow b) a Convectively Unstable flow c) an Absolutely Unstable flow. (Huerre and Rossi, 1998, P.121). . . 7

2.2 Sketch of the Bickley jet and the definition of the tilting angle θ from Candelier et al. (2011, Fig. 1). The same definition for θ and reference coordinate (X, Y, Z) is used for this project, but a different definition for coordinate in the plane of the wake/jet (x,y,z) would be used. 9

2.3 Stability diagram of the wake of a cylinder. The cylinder (and its wake) is at angle of a) $\theta = 0°$, b) $\theta = 30°$, c) $\theta = 60$, d) $\theta = 90°$ with the vertical stratification. Symbols correspond to ($\bullet$) stable; and ($*$) unstable experiments. The solid line corresponds to numerical results. Grey symbols correspond to experimental results of Boyer et al. (1989). (Meunier, 2012, Fig. 6) 10

4.1 Schematic of the experimental set-up of Meunier (2012) and the coordinate system used in this project. Note that the tilting angle is defined as α in the figure, but in the project it is defined as θ. (Meunier, 2012, Fig. 1) 12

5.1 Figures showing the detection method for cusp forming on complex ω-plane. Points on the left (red) and right (green) switch sides as the line $\omega(\alpha)$ (blue) self-intersect and form cusps when $k \in F_k$, where F_k is a line on the complex α-plane parallel to real axis, and is deforming by moving downward (more negative α_i) from (a) to (d). 22

5.2 Comparison of eigenspectra on the complex ω-plane of a) Couette Flow b) Poiseuille Flow under Re=2000 from the solver this project use and data from Schmid and Henningson (2001) . x-axis represents ω_r. y-axis represents ω_i. 24

5.3 Comparison of Neutral Stability diagram in *Ra*-*Re* plane of Rayleigh-Benard-Poiseuille flow for $Pr = 1$ between data from a) this solver (after conversion to Ra), and data from b) Jerome et al. (2012). Key modes: Transverse Rolls (TR), Tollmien-Schlichting waves (WS). 26

6.1 Breakdown of Eigenspectra by variables, at $Fr = 50$, $Re = 7.9$, $\alpha = 0.7713 - 0.48i$. $\theta = 90°$ and a) $Sc = 700$; b) $Sc = \infty$. 28

6.2 Comparison of Eigenspectra Branches at a) high and b) medium Froude number with $Re = 26.4031$, $\alpha = 0.98945 - 0.552255i$, $\theta = 90°$. $Sc = 700$. 29

6.3 Destabilising effect of increasing Re on the OS mode in the Orr-Sommerfeld equation (almost no stratification, $Fr = 50$). $Sc = 700$, $\theta = 90°$. 30

6.4 Destabilising effect of increasing Re on the OS mode at medium stratification ($Fr = 2$). $Sc = 700$, $\theta = 90°$. 31

6.5 Stabilising effect of decreasing Fr on the OS mode at medium $Re = 20$. $Sc = 700$, $\theta = 90°$. 31

6.6 Neutral Reynolds number at different Froude number when the shear is aligned with stratification. $Sc = 700$. 32

6.7 Comparison of St against Fr from a) the local stability analysis ($Sc = 700$, $\theta = 90°$) and b) the experiments by Meunier. (Meunier, 2012, Fig. 10). $Sc = 700$, $\theta = 90°$. . 33

6.8 Streamline and Quiver plot of Velocity field after perturbation at a) almost no stratification, and b) medium stratification. 34

6.9 Vorticity Contour of Velocity field after perturbation at a) almost no stratification, and b) medium stratification. 34

6.10 Density Perturbation Contour after perturbation at medium stratification. 35

6.11 Neutral Reynolds number (for Convective Instability) at different Froude number when shear is aligned with stratification. $Sc = 700$, $\theta = 90°$. Note that the neutral Re tends to infinity before Fr decrease to the stability boundary defined by the Howard's criterion. 36

6.12 Eigenspectra Branches comparison at a) high and b) medium Froude number with $Re = 7.9$, $\alpha = 0.77 - 0.48i$,$\theta = 30°$. $Sc = 700$. 37

6.13 Strouhal number as a function of the Froude number at the onset of the von Karman vortex street when cylinder is tilted at $\theta = 30°$. Results are obtained experimentally (large symbols) and numerically (small dots fitted by solid lines). (Fig. 6b, Meunier, 2012). 38

6.14 Neutral Reynolds number (for Absolute Instability) at different Froude number when shear is not aligned with stratification. $Sc = 700$, $\theta = 30°$. 39

6.15 $\omega_j(\alpha)$ tracks when $\alpha \in [0.1, 3]$, at low Froude number when shear is not aligned with stratification. The grey lines are the tracks of each $\omega_j(\alpha)$ on the complex ω-plane. The blue crosses are points of the Eigenspectra at $\alpha = 0.4$ on the complex ω-plane. The track highlighted in red is the Convectively Unstable Mode at the Central Branch. $Sc = 700$, $\theta = 30$. 39

A.1 Interface of the visualisation of results from the tracking algorithm, saved in a database. 44

A.2 The Data flow and Architecture of the Solver Script 45

List of Tables

2.1 Summary of different kinds of linear stability and their criterion in an open flow . 8

5.1 Comparison of data from Monkewitz (1988) and the numerical scheme developed in this project 24

6.1 A few example of parameter sets that would generate Absolute Instability at low Froude number 40

Chapter 1

Introduction

The wake behind a bluff body has long been a fascinating topic of study for both engineers and applied mathematicians. While the application of its fundamental behaviour have direct significance to all kinds of engineering scenario, from offshore facilities to civil construction, its physics is also as puzzling for all the researchers attempting to propose theories to explain the phenomenon, from the d'Alembert's paradox, to the discovery of Reynolds number and Von Karman Vortex Street. Only in the recent 30 years, with the advancement of numerical methods, and the understanding on Absolute and Convective instability, did a more thorough theory on the stability of the wake flow emerge.

On the other hand, the density-stratified flow has been studied for over half a century, mostly for explaining geophysical flow and application in meteorology and oceanography. However, the mathematics of many types of stratified flow, like the stratified wake this project is investigating, is far from solved.

This project is the first step towards a more thorough understanding of stratified wake, a subject that links the topic of wake flow and the subject of stratified flow. Besides the potential engineering application, the project is also largely motivated by the discovery of an unexplained new branch of stability mode at a low Froude number (strong stratification) by Meunier (2012).

1.1 Potential Application

While typical aeronautical applications do not often encounter stratified fluid, flow structure in a stratified fluid is of high interest in some other applications, namely, oceanography and meteorology. For the ocean, due to gravity, the salt content at the bottom of the ocean is higher than the one on the surface, creating a stratified density field of fluid. For the atmosphere, it is known that the atmosphere is also density-stratified under earth's gravity. Such stratification modifies flow behaviour significantly that it requires new theories for an explanation.

In the review by Lin and Pao (1979), they have listed some applications of stratified wake in explaining meteorological phenomenon, like the blocking phenomenon of the stratified atmosphere over a large mountain range, and the strong turbulence generated by such wake that affects aviation safety. A similar phenomenon can also found in the ocean bed and affects offshore structures. A better understanding of the fundamental physics of stratified wake helps offshore engineers engineer safer but leaner structures for marine and other offshore applications.

1.2 Knowledge Gap

While much experimental work has been done on the turbulence generated by the wake in a stratified medium, as summarised by Lin and Pao (1979), little has been investigated on the stability of such wake at low Reynolds number, and how the wake transition from laminar to the von Karman vortex street. Boyer et al. (1989) were the first to have done an experimental study on the stability of the linearly stratified wake after a circular cylinder (under linear stratification) at a broad range of Froude number and Reynolds number. In the experiment, the cylinder was perpendicular to the stratification direction. It showed that stratification tends to stabilise wake from transitioning into von Karman vortex street.

Such stabilising trend is consistent with previous experience and theories on wake flow and stratification. Stratification tends to inhibit flow in stratification direction (vertical usually), hence

inhibit the formation of a vortex. However, the experiment done by Meunier (2012) has shown a new phenomenon that is yet to be explained by either theory.

The experiment by Meunier (2012) shows that, by tilting the cylinder, the wake, in the cylinder's coordinate, is destabilised by increasing stratification. The discontinued trend of neutral Re at a low Froude number from that observed at high Froude number (see Figure 2.3), as well as the discontinued Strouhal number 'jump' at the low Froude number (see Figure 6.13), very possibly suggest that a new branch, or a new mode, is governing stability of the wake. Interestingly, such destabilising effect is only observed when the cylinder is tilted, but disappears when the cylinder is perpendicular to flow.

Such observation from experiments motivated this project to look for this mode under parallelised and linearised assumptions, and hopefully, formulate a theory that explains the unexpected unstable behaviour of the flow under strong stratification.

Chapter 2

Literature Review

This first half of this chapter aims to provide a brief introduction to the linear stability theory for parallel flow. For a more detailed explanation of the theory which this project is largely based upon, please visit the Interim Report or Huerre and Rossi (1998).

The second half of this chapter reviews the stability analysis performed in the past by other researchers. These analysis includes the stability of the wake flow and different classes of stratified flow, including the Poiseuille-Couette flow. It also explores different governing equations that can model stratified flow, like the Taylor-Goldstein equation. These literature are important in validating the numerical script of this project. Lastly, we would review the experiment by Meunier (2012).

2.1 Theory on Hydrodynamic Stability

Hydrodynamic Stability is a branch of fluid mechanics that tackles the governing equation of fluid flow from a stability standpoint. The study of Hydrodynamic Stability is important mainly because it provides an explanation as to why many analytical solutions that satisfy the governing equation is not naturally occurring or observable at all in experiments. As Landau and Lifshitz (1959) puts it:

> 'Yet not every solution of the equations of motion, even if it is exact, can actually occur in Nature. The flows that occur in Nature must not only obey the equations of fluid dynamics, but also be stable.'

The study of Hydrodynamic Stability, therefore, helps explain the flow behaviour, like the transition from laminar to turbulence, or in our project, the transition to von Karman Vortex Street from circulation bubbles.

The essence of hydrodynamic stability was formulated back in the nineteenth century, most notably by Helmholtz, Kelvin, Rayleigh and Reynolds. At the time, the method of normal modes to study dynamic systems was already in place (Drazin and Reid, 2004).

The theory suggests that the flow can be modelled as a "system", with a known solution (space-time field) referred to as the "basic flow". When the system is perturbed from the basic flow, the stability of the system under such basic flow arrangement determines if the system will go back the basic flow eventually. A more formal way to define stability is in the Lyapunov sense, but we will restrict our study in the linear sense, in which the perturbation is assumed to be infinitesimal, or sufficiently low. Although nonlinear and secondary instability may also destabilise a flow, literature has shown that primary linear stability firstly governs the stability of a wake flow before any global instability or nonlinear instability. Therefore in this project, we will be focusing only on linear stability.

In short, linear stability suggests that, when the system is perturbed from the equilibrium (basic flow) by an infinitesimal amount, if the system goes back to the equilibrium, it is stable. Otherwise, it is unstable. Naturally occurring solution to the governing equation should always exhibit stability since the unstable solutions will not stand perturbations like natural noises. Note that the basic flow need not be steady (e.g. von Karman Vortex Street).

2.2 Theory on Absolute and Convective Instability

Before the concept of Absolute and Convective Instability was introduced in the 1980s into the Hydrodynamic Stability community, only the linear instability of an open flow was considered.

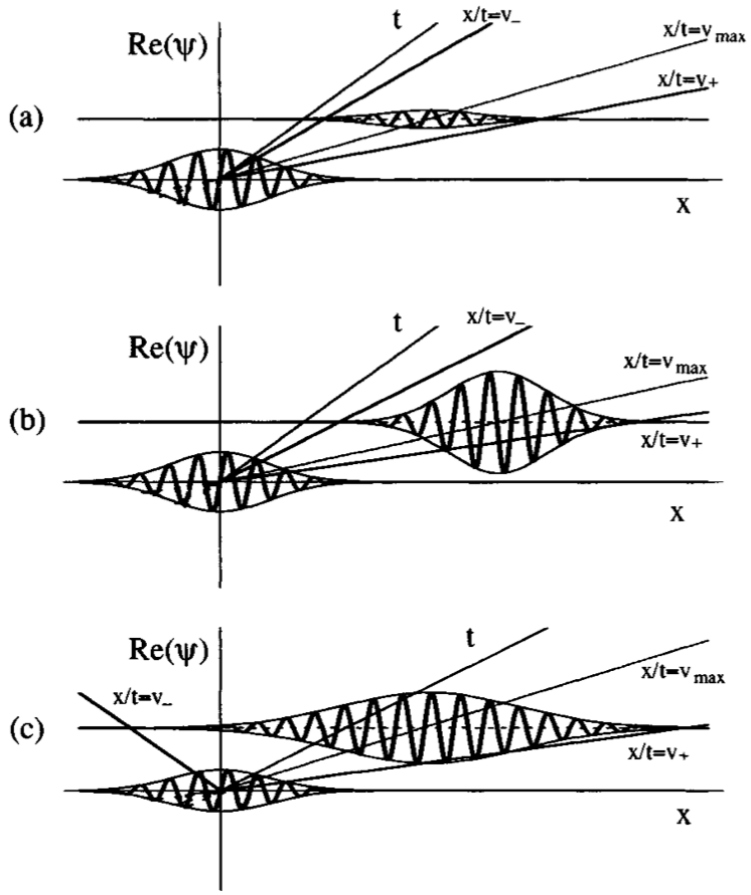


Figure 2.1: Linear response of a disturbance of a) a Linearly Stable flow b) a Convectively Unstable flow c) an Absolutely Unstable flow. (Huerre and Rossi, 1998, P.121).

The concept follows that, in a linearised stability problem, if all infinitesimal disturbance of any sort would decay over the concerned domain in time and space, it is linearly unstable.

However, many flows, including the one this project is focusing on, are open flow — flow that allows the disturbance to be washed away downstream and never circulated back — meaning that even if the flow is linearly unstable, without a continuous input of perturbation, the open flow will not grow unstable. Such concept led to the definition of Convective and Absolute Instability, which distinguish the stability of the perturbations that would be convected away, and the stability of the perturbations that would stay and grow. The formal definition for Convective and Absolute Instabilities are:

1. (Linearly) Convectively Unstable: Some disturbance would grow in time and space, but ultimately advected away from the spatial domain considered by the free stream (basic flow).
2. (Linearly) Absolutely Unstable: Some disturbance would grow in time and space and stayed at the source location, which would NOT be advected away from the spatial domain considered by the free stream (basic flow).

Their definitions are also presented in Figure 2.1 (Huerre and Rossi, 1998, P.121).

Note that both Convective and Absolute stability discussed in this project is implicitly referred as the linear sense.

2.2.1 Mathematical Definition for Absolute and Convective Instability

Formally, Convective and Absolute Instability are defined by looking at the asymptotic impulse response (i.e. Green's function of the system) over time ($t \to \infty$) at disturbed location ($x = 0$),

Linear Stability	Definition ($G(x,t)$ is the Green function)
Linearly Stable	$\lim_{t\to\infty} G(x,t) = 0$ along all rays $\frac{x}{t} = const.$
Linearly Unstable	$\lim_{t\to\infty} G(x,t) = \infty$ along all rays $\frac{x}{t} = const.$
Convectively Unstable	$\lim_{t\to\infty} G(x,t) = 0$ along the ray $\frac{x}{t} = 0.$
Absolutely Unstable	$\lim_{t\to\infty} G(x,t) = \infty$ along the ray $\frac{x}{t} = 0.$

Table 2.1: Summary of different kinds of linear stability and their criterion in an open flow

assuming the impulse amplitude is infinitesimal under the linear assumption (no finite amplitude effect). Given the system is Linearly Unstable, if at $t \to \infty$, $x = 0$ the response decays, then it is Convectively Unstable. If not, it is Absolutely Unstable. The mathematical definition is also summarised in Table 2.1.

2.3 Past work on stability of wake flow

The formulation of the theory for the transition of wake flow from steady and stable recirculation bubble to von Karman vortex street started to gain momentum in the 1980s, with the introduction of Absolute and Convective Instability. In the research monologue Huerre and Rossi (1998), they recognised Koch (1985) as the first to realise the Absolute/Convective Instability in the wake flow. After a formal introduction of the concept in the research community, Monkewitz (1988) performed a local analysis utilising this concept. By applying WKBJ approximation (given the wake is weakly non-linear), Monkewitz performed the local analysis with a two parameter parallel wake profile, which is established based on experimental data. He showed that at the beginning of the transition, there is a pocket of absolute instability. Chomaz et al. (1988) proposed that this pocket, when grew large enough, would further develop into the global mode of self-sustained oscillation. It is therefore via this process of local absolute instability growing into the global mode that steady recirculation bubble transition into von Karman vortex street. The review paper by Huerre and Monkewitz (1990) and the research monologue Huerre and Rossi (1998) both provided a nice summary of this process.

It is therefore believed that, by analysing the local instability under the linear parallel assumption, similar to what Monkewitz (1988) has done with the non-stratified wake, one can develop a qualitative explanation to the stratified wake too. This local stability analysis will be the main focus of this project.

However, it is also important to recognise that it is not possible to precisely predict the transition Reynolds number without a proper global instability analysis. As will be discussed in Chapter 3, the global analysis will be outside the limit of this project under the tight project timeframe.

2.4 Past work on stratified wake flow and stability of stratified flow

Much of the foundational theory of stratification, including internal gravity waves, blocking effect and baroclinic instability is reviewed in the book by Drazin and Reid (2004). As these behaviour are not the main focus of this project, they are not reviewed here.

There were also a considerable amount of work in particular on the topic of wake collapse. They are reviewed by Lin and Pao (1979). Although the turbulence in a stratified wake is not the main focus of the project, one of the main observation drawn from the review is that stratification tends to inhibit vertical perturbation or vertical fluid motion.

Regarding stability, there has been a few work on the stability of different classes of flow under stratified fluid. Due to limited project timeframe, only those utilising numerical method was reviewed. Research on stratified flow is especially active in the early 1970s, surrounding the Taylor-Goldstein equation under the inviscid assumption. Limited by the computational resource and mathematical tools available, inviscid assumption allows them to combine the equation governing density ρ with vertical velocity v, which simplifies the linearised equations to the Taylor-Goldstein equation.

The earliest attempt at using a numerical method on the Taylor-Goldstein equation was by Gage and Reid (1968). With limited computational resource, it only performed the numerical stability analysis of Poiseuille flow in a thermally linearly stratified fluid after quite a considerable amount of analytical or asymptotical steps. Note that thermal stratification is the same as density stratification after Boussinesq approximation that links temperature directly to density.

A more straightforward numerical approach on the Taylor-Goldstein equation was performed by Hazel (1972) using Runge-Kutta shooting method when analysing the mixing layer. In particular, Hazel (1972) has shown precise numerical consistency with the celebrated theory by Howard (1961) that if everywhere in the flow the local Richardson number

$$Ri > \frac{1}{4}, \tag{2.1}$$

then the flow is unconditionally stable.

There are other research on the stability of stratified flow that include viscosity and diffusivity. These works include, but not limited to, Maslowe and Thompson (1971) on free shear layer using Runge-Kutta shooting method, and Fujimura and Kelly (1988) on Poiseuille-Couette flow using Chebyshev collocation method. Jerome et al. (2012) also performed a similar study on Poiseuille-Couette flow, but had expanded the analysis to 3D. Although these research are not focusing on wake flow, they provided two aspirations for this project:

1. They provided readily available data that can be used to validate the solver script written for this project (see Section 5.3).
2. They all show that stratification stabilise the flow. This implies an increase in neutral Reynolds number as Froude number decrease, as long as shear is aligned with stratification.

2.5 Recent work on cases when stratification is not align with shear

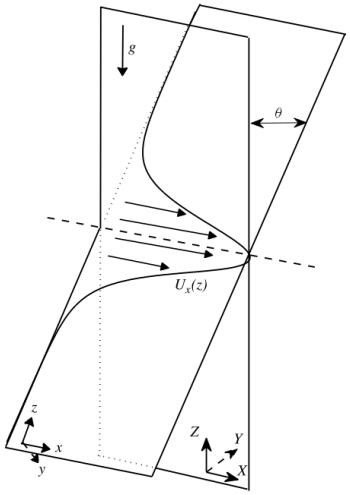


Figure 2.2: Sketch of the Bickley jet and the definition of the tilting angle θ from Candelier et al. (2011, Fig. 1). The same definition for θ and reference coordinate (X, Y, Z) is used for this project, but a different definition for coordinate in the plane of the wake/jet (x,y,z) would be used.

Among all of the research work mentioned in the previous section, they have all shown a stabilising trend as stratification strength increase. It is indeed the case when stratification is aligned with the shear force on the 2D plane of the wake (Boyer et al., 1989). However, recent research by Meunier (2012) and Candelier et al. (2011) has shown that when the shear and stratification is not aligned, but at an inclination angle, after a certain threshold of stratification, increasing stratification destabilise the system. Such angle is defined as θ in this report (α in Meunier (2012), but α is

the wavenumber in stream-wise direction in this report), as shown in Figure 2.2 (Candelier et al., 2011, Fig. 1).

This is first shown by Candelier et al. (2011) on a Bickley jet under the inviscid and non-diffusive assumption (i.e. using Taylor-Goldstein equation). When the plane of the jet is inclined with an angle, the most unstable mode is the sinuous Kelvin-Helmholtz mode at Froude number larger than the equivalent of $Ri = 1/4$, but as Froude number decrease (stronger stratification) below such threshold, there is a 'jump' from the KH mode to another mode.

A similar trend is observed experimentally by Meunier (2012). When the plane of the wake is at angle of 30° and 60° with the stratification, as Froude number decrease, the neutral Reynolds number increase. The trend continues up to a threshold of Froude number. After that, the neutral Reynolds number decrease as Froude number decrease further, as shown in Figure 2.3 (Meunier, 2012, Fig. 6).

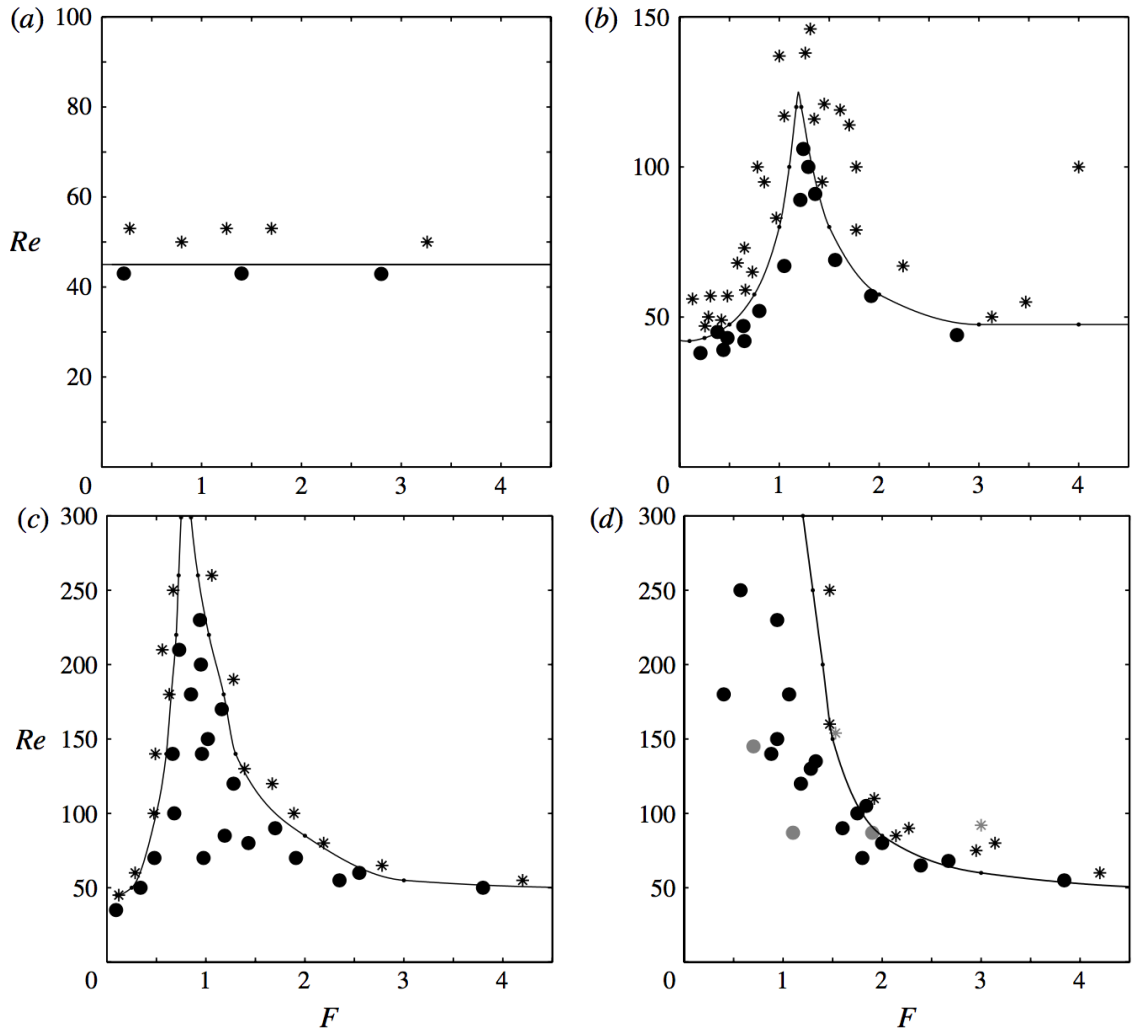


Figure 2.3: Stability diagram of the wake of a cylinder. The cylinder (and its wake) is at angle of a) $\theta = 0°$, b) $\theta = 30°$, c) $\theta = 60$, d) $\theta = 90°$ with the vertical stratification. Symbols correspond to (•) stable; and (∗) unstable experiments. The solid line corresponds to numerical results. Grey symbols correspond to experimental results of Boyer et al. (1989). (Meunier, 2012, Fig. 6)

It should also be noted that experimental result from Meunier (2012) shows that the wake remains 2D in the direction of the cylinder tilting. Meunier has also performed 2D DNS to validate the experimental result. This implies the phenomenon observed is a 2D feature in the cylinder's coordinate.

Chapter 3

Project Aim, Context and Limit

As mentioned in the beginning, this project is in a larger context in working towards a complete understanding of stratified wake. This project by no means attempts to give a full picture to the physics of the stratified wake, but it is the first step towards a thorough understanding. Much work will continue based on the outcome of this project to gain the complete picture, including but not limited to a global analysis of the wake, a sensitivity study or an adjoint study. However, in this project, we will confine ourselves to linear analysis of the flow under a parallel assumption.

3.1 Parallel, linear and 2D assumption

As mentioned before, this project will be based on linear assumption because linear stability first governs the local Absolute Instability before transitioning to a von Karman Vortex Street. By calculating the onset of the local Absolute Instability, one should be able to recreate trends similar to the observation from the experiments.

Keeping in line with the analysis done by Monkewitz (1988), we will further assume a parallel model for the basic flow. Under WKBJ approximation, this parallel assumption is valid at each stream-wise location when wake flow is weakly non-parallel. Further steps are required, however, to generalise the local analysis to the global sense. This step, however, would be outside of the boundary of this particular project.

We will also assume the wake itself is 2D and aligned with the cylinder. This assumption is validated by experimental and numerical analysis by Meunier (2012).

Furthermore, the usual assumptions of open flow and incompressibility are also implicitly implied in the project.

3.2 Project aim

This project aims to provide a more quantitative and theoretical understanding as to the reason behind the behaviour of a stratified wake, via linear stability analysis under the parallel assumption. In particular, based on experimental work by Meunier (2012), we are interested in explaining the stabilising trend when stratification strength increase, and the destabilising mode that happens at a very high stratification only when shear is not aligned with stratification. The project also aims to provide data for further analysis for predicting the onset of global stability and frequency of von Karman Vortex Street.

Chapter 4

Formulation

4.1 Coordinate system

Since there are multiple coordinate systems involved, it is better to define them clearly. In this project, we will be using the coordinate systems defined by Meunier (2012), that is:

- (x, y, z) would be main coordinate system the equation would be resolved on, which follows the cylinder tilting. This is because Meunier (2012) has shown that wake is 2D and follows the cylinder.
- (X, Y, Z) would be the reference coordinate system. It follows the gravity and stratification direction.
- x and X are the same direction pointing at the stream-wise direction.
- z is the spanwise direction of the cylinder, while $+y$ is the $-Z$ direction when $\theta = 90^{\circ}$
- $+Z$ is the upward vertical direction, opposite to gravity direction.

To better show the coordinate systems, Figure 4.1 shows the schematic of the experimental set up of Meunier (2012).

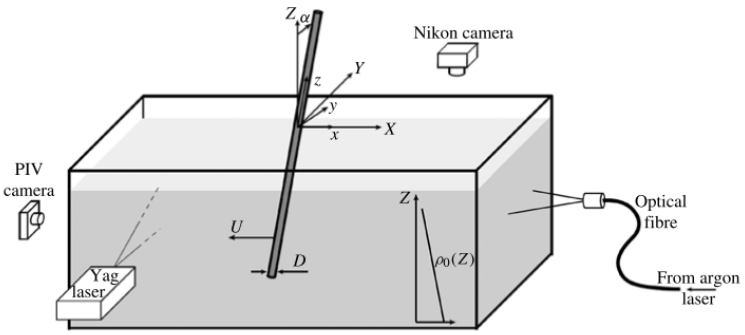


Figure 4.1: Schematic of the experimental set-up of Meunier (2012) and the coordinate system used in this project. Note that the tilting angle is defined as α in the figure, but in the project it is defined as θ. (Meunier, 2012, Fig. 1)

4.2 Non-dimensionalisation and linearisation

There are quite a few ways to non-dimensionalise the equation, as mentioned in Chen and Spedding (2017). While time scale is commonly defined by the free stream velocity, there are a few ways to set the length scale and density scale based on either the geometry or the wake and density profile. Different non-dimensionalisation provides convenience for different kinds of applications, but only one would be used in this project.

Here, variables with $*$ represent dimensional quantity.

4.2.1 Non-dimensionalisation

The most straightforward way to non-dimensionalise the governing equations is to use free stream velocity U_o^*, cylinder diameter D^* and the average density ρ_o^*.

However, to allow a more convenient definition for the wake profile $U(y)$ (to be defined in Section 4.2.2.1), in this report, we will define U_o^* as the average mean velocity of the profile and D^* as the half-width of the wake profile. They would be defined in Section 4.2.2.1.

Using D^*/U_o^*, D^* and $\rho_o^* {D^*}^3$ to scale time, length and mass, we obtain the following governing equations in non-dimensional form.

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u}) = -\boldsymbol{\nabla} p + \frac{1}{Re} \boldsymbol{\Delta} \boldsymbol{u} + \frac{1}{Fr_g^2} \rho \hat{\boldsymbol{g}}, \tag{4.1a}$$

$$\frac{\partial \rho}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \rho = \frac{1}{ReSc} \nabla \rho. \tag{4.1b}$$

where $Re = \rho_0^* U_o^* D^* / \mu^*$ (μ^* is the dynamic viscosity) and $Sc = \kappa^* / \mu^* \rho_0^*$ (κ^* is the mass diffusivity) and $Fr_g = U_o^* / \sqrt{g^* D^*}$. $\hat{\boldsymbol{g}}$ is the unit vector representing the direction of gravity, while g^* is the dimensional value of gravity. $\mathbf{u}, \rho, p$ represents velocity, density and pressure as usual. This non-dimensionalisation scheme is also suitable even when the density stratification profile is non-linear.

4.2.2 Linearisation and parallelisation

As mentioned in Chapter 3, this project will explore the transition of the wake flow based on the parallel and linear assumptions. To linearise the governing equations, we can split the variables $(\mathbf{u}, p, \rho)$ into the basic flow and the perturbation $(\mathbf{u}', p', \rho')$:

$$\rho = \bar{\rho}(y, z) + \rho', \tag{4.2a}$$

$$\boldsymbol{u} = \boldsymbol{U}(y) + \boldsymbol{u}', \tag{4.2b}$$

$$p = p_{0,y=0} + \int_0^y \frac{\bar{\rho}(y)}{Fr_g^2} dy + p'. \tag{4.2c}$$

Note that the basic flow (wake profile) $\boldsymbol{U}(y)$ is a function of y only (in the cylinder's coordinate), but the density profile $\bar{\rho}(y, z)$ is a function of both y and z to allow stratification profile to be at a tilting angle with the cylinder. p is split into the pressure three components because the pressure variation due to density stratification needs to be taken account of by the $\int_0^y \frac{\bar{\rho}(y)}{Fr_g^2} dy$ term, where as $p_{0,y=0}$ is the reference pressure at a certain height. These component are not important once the equation is linearised, because they would be cancelled out in the linearisation process.

4.2.2.1 Wake Profile

The wake profile to be used is the same two-parameter 2D profile adopted in Monkewitz (1988). It was proven to be consistent with experimental data by Monkewitz. The profile is defined with two parameters, R and a, as shown below:

$$U(y; R, a) = 1 - R + 2RF(y), \tag{4.3a}$$

$$F(y) = \{1 + \sinh^{2a}(y \operatorname{arcsinh} 1)\}^{-1}. \tag{4.3b}$$

R is a non-dimensional parameter describing velocity ratio of centreline velocity and free-stream velocity, while a is a shape parameter of the profile, often described as the "stiffness" of the profile. By having $R = 1$, $a = 1$, one can recover the $\operatorname{sech}^2 y$ jet profile common for analysing plane jet flow.

Note that the dimensioning parameters are defined by the profile. The mean velocity U_0^* is equivalent to the average of centreline velocity and free-stream velocity $U_0^* = [U^*(\infty) + U^*(0)]/2$, or in non-dimensional form, $U_0 = [U(\infty) + U(0)]/2 = [(1 - R) + (1 + R)]/2 = 1$. The half width D^* is chosen such that $U^*(D^*) = U_0^*$, or in non-dimensional form, $U(1) = U_0 = 1$.

Due to the limit on the project timeframe, this project will only explore the one profile, $R = -1.105$, $a = 1.34$, instead of exploring the whole spectrum as Monkewitz (1988) did. The hypothesis is that at high Froude number (low stratification), the profile will not be so significantly changed by stratification, such that $R = -1.105$, $a = 1.34$ would remain a good approximation. This set of R and a was found to be the most relevant to the experimental data (Monkewitz, 1988). There would be further discussion on this hypothesis in Section 6.3.

However, it should be noted that in the development of the solver script, these two parameters R and a are not hardwired into the code, but remains as an input parameter that can be easily changed. This allows future re-use of the code in other projects.

4.2.2.2 Density Profile

Only the profile of a linear stable stratification is considered in this project. The average density profile (density at system's equilibrium) can be represented by

$$\bar{\rho}(y) = 1 - \frac{\partial \bar{\rho}}{\partial Z} y \sin\theta. \tag{4.4}$$

Note that for a stably stratified fluid, $\frac{\partial \bar{\rho}}{\partial Z} < 0$. Also, when $\theta = 90°$, y is pointing downward (negative Z-direction), and $\bar{\rho}$ should be increasing linearly with y for a stable stratification.

With the linear density profile defined, we can also define a few parameters in the non-dimensional form. The dimensional Brunt-Väisälä frequency (of a stably stratified density field) is defined as:

$$N^{*^2}(Z^*) = -g^* \frac{d\bar{\rho}^*/dZ^*}{\bar{\rho}^*}(Z^*), \tag{4.5}$$

which in non-dimensional form, is given by:

$$N^2(Z) = -\frac{1}{Fr_g^2} \frac{d\bar{\rho}/dZ}{\bar{\rho}}(Z) = N^{*^2}(Z^*) \, (\frac{U_o^*}{D^*})^2. \tag{4.6}$$

Note that Brunt-Väisälä frequency varies along the vertical direction (Z) for a general density profile, because of changing average density $\bar{\rho}$. However, in a linearly stratified profile, we can approximate Brunt-Väisälä frequency to be roughly constant under Boussinesq approximation. Therefore, we can define the global Brunt-Väisälä frequency as the $N_o^* = N^*(0)$ and use it as a homogeneous constant along the vertical direction.

Since the definition of Froude number used by most researchers are actually the inverse of non-dimensional global Brunt-Väisälä frequency, we can simply define the Froude number Fr as :

$$Fr = 1/N_0 = -Fr_g \sqrt{\frac{\bar{\rho}}{d\bar{\rho}/dZ}(0)} = \frac{1}{N^*(0)} \, (\frac{D^*}{U_0^*}).$$

To distinguish the Froude number representing the stratification strength, used by most researchers, and the Froude number used to represent just the gravity strength, we use the subscript g to represents gravitational Froude number (i.e. $Fr_g = U_o^*/\sqrt{g^* D^*}$).

We can also define the non-dimensional stratification length, which is the inverse of $\frac{d\bar{\rho}/dZ}{\bar{\rho}}(Z)$ taken at mid point ($Z = 0$) of the density field (assuming $\bar{\rho}(0) = 1$):

$$L = \frac{-1}{D^*} \frac{\bar{\rho}^*}{d\bar{\rho}^*/dZ}(0) = -\frac{\bar{\rho}}{d\bar{\rho}/dZ}(0). \tag{4.7}$$

4.2.2.3 Applying linearisation and setting up modal solution

So by substituting (4.2a),(4.2b),(4.2c) into the governing equation and eliminating higher order terms, the governing equations become:

$$\rho(\frac{\partial u'}{\partial t} + U\frac{\partial u'}{\partial x} + v'\frac{\partial U}{\partial y}) = -\frac{\partial p'}{\partial x} + \frac{1}{Re}(\frac{\partial^2 u'}{\partial x^2} + \frac{\partial^2 u'}{\partial y^2} + \frac{\partial^2 u'}{\partial z^2}), \tag{4.8a}$$

$$\rho(\frac{\partial v'}{\partial t} + U\frac{\partial v'}{\partial x}) = -\frac{\partial p'}{\partial y} + \frac{1}{Re}(\frac{\partial^2 v'}{\partial x^2} + \frac{\partial^2 v'}{\partial y^2} + \frac{\partial^2 v'}{\partial z^2}) + \frac{\rho' \sin\theta}{Fr_g^2}, \tag{4.8b}$$

$$\rho(\frac{\partial w'}{\partial t} + U\frac{\partial w'}{\partial x}) = -\frac{\partial p'}{\partial z} + \frac{1}{Re}(\frac{\partial^2 w'}{\partial x^2} + \frac{\partial^2 w'}{\partial y^2} + \frac{\partial^2 w'}{\partial z^2}) - \frac{\rho' \cos\theta}{Fr_g^2}, \tag{4.8c}$$

and the density transport equation becomes:

$$\frac{\partial \rho}{\partial t} + U\frac{\partial \rho'}{\partial x} + v'\frac{\partial \bar{\rho}}{\partial y} + w'\frac{\partial \bar{\rho}}{\partial z} = \frac{1}{ReSc}(\frac{\partial^2 \rho'}{\partial x^2} + \frac{\partial^2 \rho'}{\partial y^2} + \frac{\partial^2 \rho'}{\partial z^2}), \tag{4.9}$$

and incompressibility becomes:

$$\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z} = 0. \tag{4.10}$$

After that, one can first formulate the modal solution. The parallel assumption allows us to assume the flow is homogeneous in the x-direction (under WKBJ approximation for the weakly non-linear flow), whereas the cylinder is assumed to be infinitely long, allowing us to make a homogeneous assumption in the z-direction. Therefore, the perturbation variation in the x- and z-direction can be represented in wavenumber α and β. Furthermore, the homogeneity assumed in time allows us to use frequency to represent perturbation in time. Therefore, we will assume the perturbation of all the parameters can be written as

$$\begin{bmatrix} u' \\ v' \\ w' \\ p' \\ \rho' \end{bmatrix} = \begin{bmatrix} \tilde{u}(y) \\ \tilde{v}(y) \\ \tilde{w}(y) \\ \tilde{p}(y) \\ \tilde{\rho}(y) \end{bmatrix} \exp\{i(\alpha x + \beta z - \omega t)\}, \tag{4.11}$$

which is also set to be compatible with the basic flow profile (wake profile and linear stratification profile). $[\tilde{u}(y), \tilde{v}(y), \tilde{w}(y), p(y), \tilde{\rho}(y)]^T$ represents the shape of the mode at variable, whereas α and β represents wavenumber in the x and z direction. ω represents frequency in time. Note that up to this point, the full 2D assumption is not yet applied, as $\beta \neq 0$ is still allowed. This is to allow the derivation of a more general set of equations for the problem before applying the assumption for this project.

Substituting above equations, we can get:

$$i\bar{\rho}(U\alpha - \omega)\tilde{u} + \bar{\rho}DU\tilde{v} = -i\alpha\tilde{p} + \frac{1}{Re}(D^2 - k^2)\tilde{u}, \tag{4.12a}$$

$$i\bar{\rho}(U\alpha - \omega)\tilde{v} = -D\tilde{p} + \frac{1}{Re}(D^2 - k^2)\tilde{v} + \frac{\tilde{\rho}\sin\theta}{Fr^2}, \tag{4.12b}$$

$$i\bar{\rho}(U\alpha - \omega)\tilde{w} = -i\beta\tilde{p} + \frac{1}{Re}(D^2 - k^2)\tilde{w} - \frac{\tilde{\rho}\cos\theta}{Fr^2}, \tag{4.12c}$$

$$i(U\alpha - \omega)\tilde{\rho} + D\bar{\rho}\tilde{v} + D_z\bar{\rho}\tilde{w} = \frac{1}{ReSc}(D^2 - k^2)\tilde{\rho}, \tag{4.12d}$$

$$i\alpha\tilde{u} + D\tilde{v} + i\beta\tilde{w} = 0, \tag{4.12e}$$

where $D = \frac{d}{dy}$,$D_z = \frac{d}{dz}$ and $k^2 = \alpha^2 + \beta^2$. Also, $D\bar{\rho} = -\frac{d\bar{\rho}}{dZ}\sin\theta$ and $D\bar{\rho} = \frac{d\bar{\rho}}{dZ}\cos\theta$.

This is now a linear system of ODE equations to be solved, in which the solution set would be in terms of (α, β, ω). In other words, this system of equations can be regarded as an eigensystem with input (α, β) and output ω. Furthermore, this system is orthogonal, meaning with a suitable scheme of discretisation, the unknown eigenfunction can be discretised into eigenvector, and the system can be resolved numerically in matrix form.

4.2.3 Criteria for Absolute and Convective Instability in terms ω

Previously, the linear instability was introduced in Section 2.1 and definitions of Absolute and Convective Instability were given in Section 2.2, but their criterion in the practical form of the analysis was yet to be defined because the modal solution was not yet defined. Now with the modal solution and the system of equations defined, we can define the criterion for linear instability. This following explanation is largely based on Huerre and Rossi (1998).

Linear instability, in short, is equivalent to having a ω whose imaginary part, which is the temporal growth rate, is larger than zero under real input α and β, such that over time, the perturbation will grow. Since there will be a spectrum of $\omega_j(\alpha, \beta)$ under different input (α, β), the system's (or the flow's) stability at equilibrium is the defined by the ω_j with the maximum temporal growth rate in the whole spectrum. In other words, the ω that defines linear stability is

$$\omega_{i,max} = \max\{\omega_{j,i}(\alpha, \beta)\} \text{ for all real } (\alpha, \beta) \text{ and indices } j$$

where j is the index for each mode in the spectrum of ω and the subscript i represents the imaginary part of ω.

(Note that in this report, subscript i represents imaginary part, whereas subscript r represents real part.)

Linear stability is determined by:

$$\text{If } \omega_{i,max} > 0, \text{ the system is linearly unstable.}$$

$$\text{If } \omega_{i,max} < 0, \text{ the system is linearly stable.}$$

It should be noted that in an eigensystem, the amplitude of the eigenfunction/eigenvector is not defined. Physically it is because linear stability assumes infinitesimal perturbation. Finite amplitude effect or any nonlinear effect is therefore not considered in this analysis, and one should be caution of these limits when performing this linear analysis.

Under the umbrella of linear instability, stability is further categorised as Absolute and Convective. For Absolute Instability, its criterion involve the pinching process in the complex α/β-plane and cusp forming on the ω-plane. The pinching process will not be discussed here. One should refer to the Interim Report for a more detailed explanation. Here, only the criterion is listed:

$$\text{If } \omega_{i,0} > 0 \text{ and } \omega_{i,max} > 0, \text{ the system is Absolutely Unstable.}$$

$$\text{If } \omega_{i,0} < 0 \text{ and } \omega_{i,max} > 0, \text{ the system is Absolutely Stable and Convectively Unstable.}$$

$$\text{If } \omega_{i,0} < 0 \text{ and } \omega_{i,max} < 0, \text{ the system is Absolutely Unstable.}$$

The absolute frequency $\omega_0(\alpha_0, \beta_0)$ is defined as the peak point of the cusp formed in ω-plane when α(and β) is equal to the pinching point α_0 (and β_0). The pinching point is proved to be a saddle point of the function $\omega(\alpha)$ (and $\omega(\beta)$), and the function is analytic on the complex α/β plane. Therefore a Secant Search method can be implemented to search for the pinching point α_0 and the absolute frequency ω_0. Such property would be utilised later in the script (see Section 5.2.2.2).

4.2.4 Simplifying the system of equations

To simplify the system of equations so as to reduce the computational cost, one can first eliminate some variables. By putting (4.12a), (4.12b), (4.12c) into (4.12e), and eliminating $\tilde{u}$ and $\tilde{w}$, one can get:

$$iD[\bar{\rho}(U\alpha-\omega)]\tilde{v}+i\alpha\bar{\rho}DU\tilde{v}=-(D^2-k^2)\tilde{p}+\frac{\sin\theta-\cos\theta}{Fr^2}D\tilde{\rho}. \tag{4.13}$$

If we sum $-D\times$(4.13) and $(D^2-k^2)\times$ (4.12b), one will get:

$$i\bar{\rho}[(U\alpha-\omega)(D^2-k^2)-\alpha D^2U]\tilde{v}+i(D\bar{\rho})[(U\alpha-\omega)D-\alpha(DU)]\tilde{v} \\ =\frac{1}{Re}(D^2-k^2)^2\tilde{v}-\frac{k^2}{Fr^2}\tilde{\rho}\sin\theta+\frac{D^2\tilde{\rho}}{Fr^2}\cos\theta. \tag{4.14}$$

This equation is equivalent to the Orr-Sommerfeld equation with the additional buoyancy term.

Now, to couple this equation with (4.12d), one also need an extra equation to solve for $\tilde{w}$, but (4.12c) has an extra $\tilde{p}$ we would not like to solve for. Instead, $\tilde{p}$ can be eliminated by summing $-i\beta\times$ (4.12b) and $D\times$ (4.12c). This gives:

$$i(D\bar{\rho})(U\alpha-\omega)\tilde{w}+i\alpha\bar{\rho}(DU)\tilde{w}+i\bar{\rho}(U\alpha-\omega)(D\tilde{w}-i\beta\tilde{v}) \\ =\frac{1}{Re}(D^2-k^2)(D\tilde{w}-i\beta\tilde{v})-\frac{1}{Fr^2}(i\beta\tilde{\rho}\sin\theta+D\tilde{\rho}\cos\theta). \tag{4.15}$$

Physically, this equation represents the transport equation for the flow-normal vorticity (the term $(D\tilde{w}-i\beta\tilde{v})$).

Now there are three equations, (4.14), (4.15), and (4.12d), to solve for $\tilde{v}$,$\tilde{w}$,$\tilde{\rho}$.

In addition, we can also add the Squire equation as the forth equation. Squire equation can formed by summing $i\beta\times$(4.12a) and $-i\alpha\times$(4.12c):

$$i\bar{\rho}(U\alpha-\omega)\tilde{\eta}+i\beta\bar{\rho}(DU)\bar{v}=\frac{D^2-k^2}{Re}\tilde{\eta}+\frac{1}{Fr^2}i\alpha\tilde{\rho}\cos\theta, \tag{4.16}$$

where $\tilde{\eta}=i\beta\tilde{u}-i\alpha\tilde{w}$ is the wall-normal vorticity. However, it is only driven by the other three variables ($\tilde{v}$,$\tilde{w}$,$\tilde{\rho}$) and does not feedback into the three governing equation. Physically it also does not contribute to instability.

4.2.4.1 Boussinesq Approximation

The system of equations: (4.14), (4.15) and (4.12d) can be solved directly in numerical matrix form to get $\tilde{v}$,$\tilde{w}$,$\tilde{\rho}$. However, to follow the convention of most research in the past, Boussinesq Approximation is applied to the system of equations before solving. Such approximation eliminates any term with $1/L\approx 0$ in the equations, and approximate non-dimensional $\bar{\rho}\approx 1$, thus eliminating the inertial effect of the changing density. However, the buoyancy force of the density stratification is kept by $\tilde{\rho}$. Therefore, the final equations to be solved are:

$$i[(U\alpha-\omega)(D^2-k^2)-\alpha D^2U]\tilde{v}=\frac{1}{Re}(D^2-k^2)^2\tilde{v}-\frac{k^2}{Fr_g^2}\tilde{\rho}\sin\theta+\frac{D^2\tilde{\rho}}{Fr_g^2}\cos\theta, \tag{4.17}$$

$$i\alpha(DU)\tilde{w}+i(U\alpha-\omega)(D\tilde{w}-i\beta\tilde{v})=\frac{1}{Re}(D^2-k^2)(D\tilde{w}-i\beta\tilde{v})-\frac{1}{Fr_g^2}(i\beta\tilde{\rho}\sin\theta+D\tilde{\rho}\cos\theta), \tag{4.18}$$

$$i(U\alpha-\omega)\tilde{\rho}+D\bar{\rho}\tilde{v}+D_z\bar{\rho}\tilde{w}=\frac{1}{ReSc}(D^2-k^2)\tilde{\rho}. \tag{4.19}$$

4.2.4.2 Coupled equations in matrix form with Boussinesq Approximation

Therefore, the three equations (4.17), (4.18) and (4.19) can be re-written in matrix form as below:

$$-i\omega \begin{pmatrix} (D^2-k^2) & 0 & 0 \\ -i\beta & D & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \tilde{v} \\ \tilde{w} \\ \tilde{\rho} \end{pmatrix} + \begin{pmatrix} \mathcal{L}_v & 0 & (k^2\sin\theta - \cos\theta D^2)/Fr_g^2 \\ \alpha\beta U + i\beta(D^2-k^2)/Re & \mathcal{L}_{wD} & (i\beta\sin\theta + \cos\theta D)/Fr_g^2 \\ D\bar{\rho} & D_z\bar{\rho} & \mathcal{L}_\rho \end{pmatrix} \begin{pmatrix} \tilde{v} \\ \tilde{w} \\ \tilde{\rho} \end{pmatrix} = 0, \quad (4.20)$$

where

$$\mathcal{L}_v = i\alpha[U(D^2-k^2) - (D^2U)] - \frac{1}{Re}(D^2-k^2)^2, \quad (4.21a)$$

$$\mathcal{L}_{wD} = i\alpha(UD + (DU)) - \frac{1}{Re}(D^2-k^2)D, \quad (4.21b)$$

$$\mathcal{L}_\rho = iU\alpha - \frac{1}{ReSc}(D^2-k^2). \quad (4.21c)$$

The system should fulfil the following boundary condition:

$$v'(-\infty) = v'(\infty) = Dv'(-\infty) = Dv'(\infty) = w'(-\infty) = w'(\infty) = \rho'(\infty) = \rho'(-\infty) = 0 \quad (4.22)$$

Note that this system of equations is orthogonal. They can then be discretised and solved as an eigenvalue problem.

4.2.4.3 2D Assumption

According to Meunier (2012), the wake is 2D in the x-y plane of the cylinder before the transition to 3D turbulence. It is further confirmed by the 2D DNS that shows a consistent result with the experiment. Based on this assumption, we can further narrow down our research limit by setting $\beta = 0$. Numerically, the same solver will be used by simply defining $\beta = 0$.

Also note that, as Meunier (2012, P. 178) puts it:

> '...since this study focuses on the 2D wake, the flow is assumed to be invariant along the axis of the cylinder, such that the z derivatives can be suppressed. It should be noted that it does not mean that the axial velocity w is supped equal to zero. It simply means that there is no 3D instability with an axial wavelength ($\beta = 0$)...'

w can still exist under the 2D wake assumption.

Under $\beta = 0$ assumption, one can directly use (4.12c) instead of (4.15) in the system of equations, since $\tilde{p}$ in (4.12c) is already eliminated with $\beta = 0$. Under $\beta = 0$, (4.15) is effectively a version of (4.12c) after differentiation, but using (4.12c) instead of (4.15) is advantageous numerically because an order less in differentiation implies less noise is imposed into the system.

Therefore, under 2D assumption the system of equation in matrix form is:

$$-i\omega \begin{pmatrix} (D^2-k^2) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \tilde{v} \\ \tilde{w} \\ \tilde{\rho} \end{pmatrix} + \begin{pmatrix} \mathcal{L}_v & 0 & (k^2\sin\theta - \cos\theta D^2)/Fr_g^2 \\ 0 & \mathcal{L}_w & (\cos\theta)/Fr_g^2 \\ D\bar{\rho} & D_z\bar{\rho} & \mathcal{L}_\rho \end{pmatrix} \begin{pmatrix} \tilde{v} \\ \tilde{w} \\ \tilde{\rho} \end{pmatrix} = 0, \quad (4.23)$$

where

$$\mathcal{L}_w = i\alpha U - \frac{1}{Re}(D^2-k^2). \quad (4.24)$$

The same boundary condition in (4.22) applies.

4.2.5 Non-dimensionalisation by Stratification Length and Recovery of Taylor-Goldstein equation

Section 4.2.1 gives a non-dimensionalisation scheme which is suitable for linear and non-linear stratification. To increase reusability of the script, a further non-dimensionalisation that is specific to the linear stratification is not performed in this project. However, with a different non-dimensionalisation scheme, one can recover the Taylor-Goldstein equation, which is commonly used by other researchers.

When the flow is linearly stratified, we are more interested in Fr that represents stratification strength than gravity strength Fr_g, as it is the stratification strength Fr that effectively modifies the system of equations. In other words, Fr_g only arises in the equation as a product with $\sqrt{L}$ (i.e. $Fr = \sqrt{L}Fr_g$).

This may not be obvious in the non-dimensionalisation scheme used by this project, but by further dimensionalising the $\tilde{\rho}$ term by L, giving $\tilde{\rho}_\Delta = \tilde{\rho}L$, one can rewrite (4.17) in the following form:

$$i[(U\alpha - \omega)(D^2 - k^2) - i\alpha D^2 U]\tilde{v} = \frac{1}{Re}(D^2 - k^2)^2\tilde{v} - \frac{(k^2 \sin\theta - \cos\theta D^2)}{Fr^2}\tilde{\rho}_\Delta. \tag{4.25}$$

With Boussinesq Approximation, we can also rewrite (4.12d) into:

$$i(U\alpha - \omega)\tilde{\rho}_\Delta + \sin\theta\tilde{v} - \cos\theta\tilde{w} = \frac{1}{ReSc}(D^2 - k^2)\tilde{\rho}_\Delta. \tag{4.26}$$

After further non-dimensionalisation, we can see that indeed Fr modifies the equation.

Furthermore, if we then put the inviscid assumption in (4.25), one can get:

$$[i(U\alpha - \omega)(D^2 - k^2) - i\alpha D^2 U]\tilde{v} = -\frac{(k^2 \sin\theta - \cos\theta D^2)}{Fr^2}\tilde{\rho}_\Delta. \tag{4.27}$$

After that, by substituting a non-diffusive (zero RHS) version of (4.26) and assuming the cylinder is horizontal ($\theta = 90°$), that is:

$$\tilde{\rho}_\Delta = \frac{i\tilde{v}}{U\alpha - \omega}, \tag{4.28}$$

into (4.27), one can recover the Taylor-Goldstein equation:

$$[(U\alpha - \omega)(D^2 - k^2) - \alpha D^2 U]\tilde{v} + \frac{k^2\tilde{v}}{Fr^2(U\alpha - \omega)} = 0. \tag{4.29}$$

Chapter 5

Numerical Method

(4.23) gives the eigensystem needed to be solved in this project. To allow the use of a computer to solve such a system numerically, one need to discretise the equations from an ODE into a matrix operation by a numerical scheme. After that, such solver would be embedded into an algorithm that allows us to search for Convective/Absolute Instability point for a given set of parameter. Building on top of the search algorithm one can generate data like a neutral curve, a visualisation of eigenspectra, etc.

In this section, we will introduce the MATLAB script that implements this numerical solver for the system. We will first introduce the script's overall architecture. Since this project is in the context of a greater investigation on the stratified wake, the script is written in a way to maximise reusability and flexibility. Therefore, a clear overview of the script's architecture is essential. After that, we will discuss the script component one-by-one, and lastly, validate the script by comparing the result it generates with various research in the past.

5.1 Overall architecture

In general, the workflow of all the scripts follow the same "input-initialise-solving-post-processing-visualise" routine. Input parameters are first set in a "caller script", then passed down to the solver with other parameters from the initialisation process. The solver then solves for the eigenspectra, and output the eigenspectra ω into the post-processing units. Finally, the post-processed results are visualised and outputted. Figure A.2 in Appendix A shows the data flow and the architecture of the script.

There are a few caller scripts that can be called. The "visualisation" scripts are simplest, which take in the full set of input parameters (control parameters and wavenumber) and output the eigenspectra in the complex ω-plane, with the option to plot the eigenfunction of a specific ω_j. The "search" scripts take in a set of control parameters (Re,Fr,Sc,R,a) and perform a Secant Search to find α_0 and ω_0 (setting $\beta = 0$). A further search can also be performed to search for the neutral Re at a given set of (Fr,Sc,R,a). It outputs a file containing the $Re_{neutral}$ and respective α_0 at each Fr. The "track" scripts are designed to allow visualisation of how the eigenspectra changes as α_r changes at a fixed α_i under a set of control parameters. An algorithm is in place to calculate the trace of each eigenvalue on the complex ω-plane.

5.2 Script Component Description

5.2.1 Solver and Initialisation

The solver takes in the set of control parameters (Re,Fr,Sc) and the input wavenumber (α,β) and outputs a set of ω. Since it is a one-dimensional model, to allow numerical computation, we need to discretise in the y-direction. There are numerous discretisation schemes, like the Runge-Kutta shooting method, which was popular in the 60s to 70s. In this project, we will use the Chebyshev Collocation Method, which gives good accuracy for a relatively low mesh density, as suggested by Orszag (1971).

5.2.1.1 Discretisation by Chebyshev Collocation Method and transformation

The main convenience of the Chebyshev Collocation Method is that it allows differentiation (in the y-direction in this project) to be represented by a matrix multiplication. This implies the

whole system of equations can be represented entirely by matrix operations and solved as an eigenvalue problem. Although Chebyshev polynomials only work in the $[-1, 1]$ domain, a simple transformation,

$$y = \tan(b\xi), \tag{5.1}$$

using tangent can span the domain into $[-\infty, \infty]$. In the formula, ξ is the coordinate in $[-1, 1]$ for the Chebyshev polynomials. y is the non-dimensional coordinate we are discretising, perpendicular to the flow. b controls the domain size. If $b = \pi/2$, the domain will be transformed into $[-\infty, \infty]$. However, it is impossible to represent ∞ numerically. Moreover, to ensure the algorithm is robustly resolving the system and to eliminate modes that generated from numerical issues (e.g. delta waves), it is often needed to change the domain size to ensure the eigenmodes are resolved correctly. It is because the boundary conditions of the system are clamped conditions at infinity, as shown in (4.22). Numerically, a true eigenvalue will not vary if the domain size changes, given it is sufficiently large. Therefore, in this project $b = 0.99 \times \pi/2$ and $b = \arctan 10$ will be interchangeably used. They give domain size of $[-62, 62]$ and $[-10, 10]$.

5.2.1.2 Boundary Condition

As mentioned, clamped conditions are the boundary conditions imposed at infinity. Dirichlet conditions can be easily imposed by removing first and last rows and columns of the resulting matrices in the solver, but it would create spurious modes. A more elegant method was proposed by embedding the clamped condition right within the fourth order Chebyshev differential matrices. Weideman and Reddy (2000) provides the MATLAB script `cheb4c.m` which generates the fourth order Chebyshev differential matrices with the clamped condition embed, while the script `chebdif.m` generates the Chebyshev differential matrices of the first three order.

5.2.1.3 The Solver and separating the generation of matrices into the Initialisation step

With the discretisation scheme in place, the collocation points can also be put into the wake profile (4.3a) and the density profile (4.4) to generate diagonal matrices. With the transformation, matrices representing differentiation of order one to four are also in place. These matrices can now be input into the solver and assemble into a larger matrix, according to the equations in (4.23). The larger matrix is then solved for its eigenvalue, which would be equal to $i\omega$.

However, from the perspective of the overall architecture of the script, the profiles are mostly static within the iterations by "search" or "track" caller scripts. They do not need to be recalculated at every iteration, and therefore, have been put in the initialisation step. This saves a considerable amount of computational resource.

5.2.2 Search Method

Now that a solver is implemented to find ω for a given set of control parameter (Re,Fr,Sc) from input (α,β), we shall find $\omega_{i,0}$ and $\omega_{i,max}$ for the onset of Absolute and Convective Instability.

5.2.2.1 Finding Convective Instability

Finding ω_{max} is simply a matter of finding the maximum imagery part along real α, given $\beta = 0$. First, we shall find the ω_j among all the output of that will give the maximum imagery part. This will be done by the sorting algorithm `eigen_sort.m`.

Next, we can search along the real axis on ω_i-α plane to find the ω_{max}. Since we know that $\partial\omega_i/\partial\alpha = 0$ at $\omega = \omega_{max}$, we can implement a simple Secant Search algorithm to find the point where $\partial\omega_i/\partial\alpha = 0$, in which $\partial\omega/\partial\alpha$ can be approximated with a simple Central Difference scheme with an arbitrary small real $\Delta\alpha$.

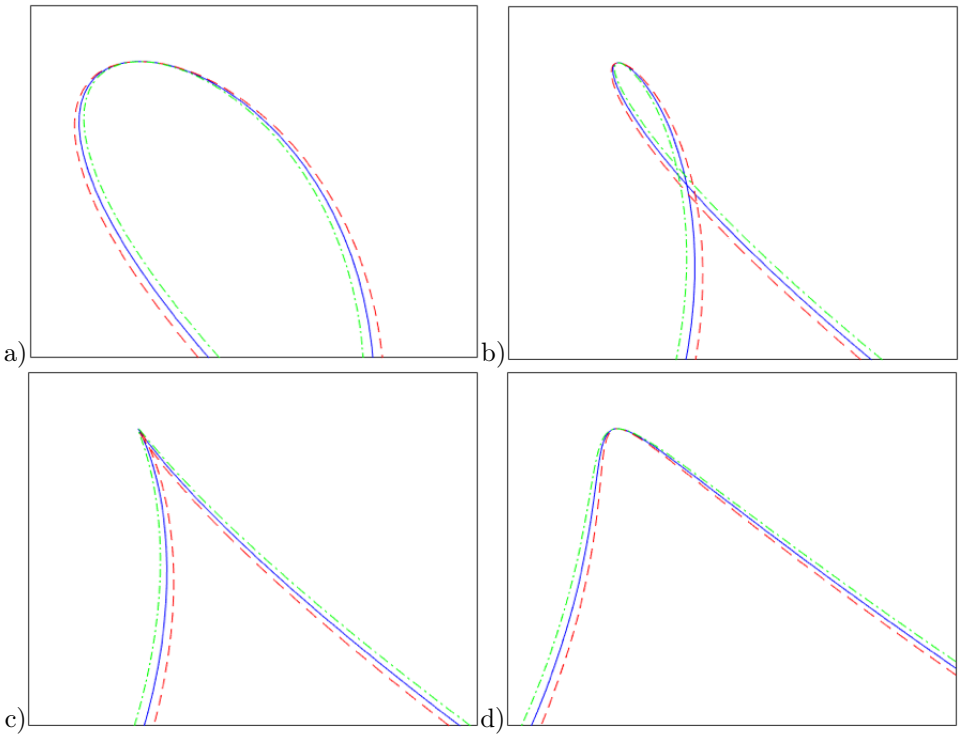


Figure 5.1: Figures showing the detection method for cusp forming on complex ω-plane. Points on the left (red) and right (green) switch sides as the line $\omega(\alpha)$ (blue) self-intersect and form cusps when $k \in F_k$, where F_k is a line on the complex α-plane parallel to real axis, and is deforming by moving downward (more negative α_i) from (a) to (d).

5.2.2.2 Finding Absolute Instability

For ω_0, a similar Secant Search method can be used, since we know that $\partial\omega/\partial\alpha(\alpha_0) = 0$, which corresponds to a saddle point if we plot ω on the complex α-plane. Unless branch switching is at the vicinity, it is known that $\omega(\alpha)$ is analytic, meaning $\partial\omega/\partial\alpha$ is the same no matter which direction in the complex α-plane the differentiation is taken.

By utilising the analytical property, the central difference scheme mentioned before can be used here to find $\partial\omega/\partial\alpha$ in any direction. To avoid potential numerical error, $\partial\omega/\partial\alpha$ is taken as the average of four directions (1+1i, 1-1i, 1, 1i) of complex α in arbitrary small magnitude. A similar Secant Search algorithm mentioned in the previous section can then used to find ω_0.

It should be noted that a saddle point is only a necessary but not sufficient condition for the point to be the pinching point that determines Absolute Instability. One should always look at its cusps forming behaviour in the ω-plane. `cusps.m` was written for such purpose.

Cusps forming is detected by the switching of parallel lines inside and outside the $\omega(\alpha)$ line, when α varies along F_k, a horizontal line on complex α-plane. As F_k is shifted downward (lowering α_i), cusps will be formed by the $\omega(\alpha)$ line. Near the $\omega(\alpha)$ line, points are set at the left and right near the line to represents the parallel lines. It is known that, as cusps form, the points on the inside of the line will become on the outside, and vice versa, as shown in Figure 5.1. Singularity at points enclosed by (inside) the $\omega(\alpha)$ line, closed by connecting beginning and end of the $\omega(\alpha)$ line, can be detected by a complex integral contour. By detecting the moment when all points are switched from one side to another, one can find the imagery part of α_0, and hence α_0 and ω_0.

Alternatively, one can use the tracking algorithm in Section 5.2.3.1 to visualise the cusp forming behaviour manually.

5.2.2.3 Finding Neutral Curve

The neutral curve on the Re-Fr plane is one of the best ways to visualises and summarises the system's character. To generate this graph, one can implement another layer of Secant Search

method to the algorithm for searching Absolute Instability. So for each given (Fr,Sc), one can search of Re that gives $\omega_{i,0} = 0$ using the same Secant Search method described earlier.

5.2.3 Track Visualisation and Database Management

5.2.3.1 Tracking Algorithm

To find new branches and have better visualisation of potential branch switching, one would like to visualise $\omega(\alpha)$ on the complex ω-plane when α moves along a horizontal line on the complex α-plane. This motivated the creation of the tracking algorithm, as the solver only output eigenvalues ω at a random order. A tracking algorithm is needed to recognise the corresponding ω_j when αchanges.

This project utilises the Hungarian Algorithm to match two sets of eigenvalues from two input α, by viewing the matching of the two sets of ω as a linear assignment problem. Although it may, from time to time, fail to link the right tracks if two tracks are getting too close to each other, the performance has been acceptable to visualise the trace of each eigenvalue in the eigenspectra on the complex ω-plane. After all, it is merely a non-critical visualisation component, and therefore is not validated.

There are multiple open source MATLAB scripts available online to perform this algorithm, but Jean-Yves Tinevez (2016) was chosen for its efficiency and clarity. It was slightly modified to add in parallelisation capability to speed up the program.

5.2.3.2 Database Management and Visualisation

Computing tracks involve iterating the core solver script hundreds of time. It can be computationally intensive to run. To avoid data being deleted and to avoid re-running the same set of input multiple times (which has wasted a considerable amount of time in the middle of the project), several scripts were written to ensure each time the tracking algorithm is run, the data is saved and can be accessed and visualised afterwards.

With the database also comes the visualisation component which helps the visualisation of the data in the database. Figure A.1 in Appendix A shows and explains the interface of the visualisation.

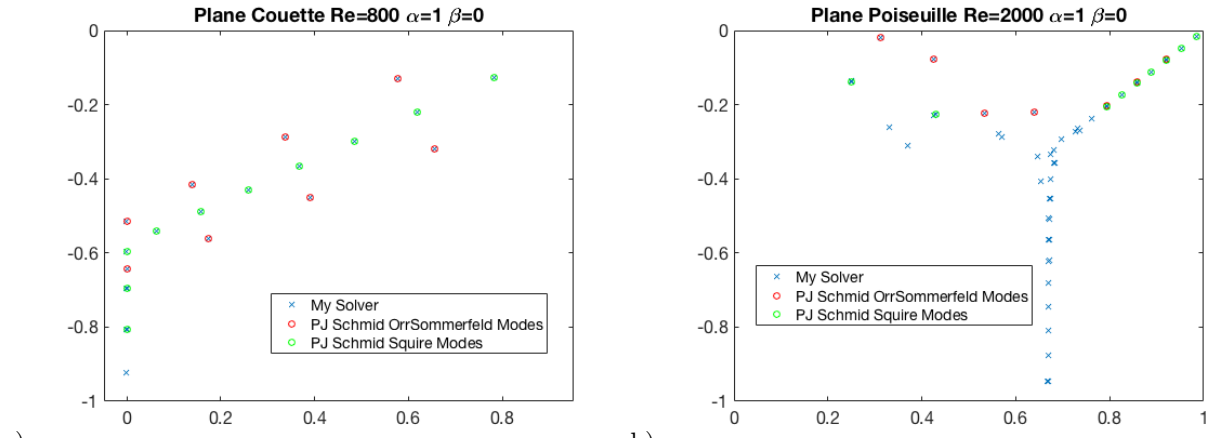
5.3 Validation

The script was designed to maximise reusability for projects later on. It is, therefore, important to ensure the script not only works in this particular project but in a broad range of flows and systems. Much effort was put into the project in validating the script with different past research.

We will use data readily available from multiple sources (Schmid and Henningson, 2001; Hazel, 1972; Jerome et al., 2012; Monkewitz, 1988; Hwang and Choi, 2006; Fujimura and Kelly, 1988). Different data will be used to validate different aspect of the script. One may also notice that some of the paper are not investigating the same class of flow (e.g. some may be investigating Poiseuille flow), while some may have different governing equation (e.g. Orr-Sommerfeld equation or Taylor-Goldstein equation). Such act only proves the flexibility of the script to be easily changed to other types of flow and system, as the script was designed to be modular. Changing governing equations or flow profile should be an easy swap.

5.3.1 Validating basic framework and initialisation component

The book by Schmid and Henningson (2001) contains data for the eigenspectra of Plane Poiseuille flow and Plane Couette flow under the Orr-Sommerfeld equation. These data are compared against data generated from the script and shown in Figure 5.2. Note the successful validation only proves that basic infrastructure, like the Chebyshev differential matrices, the boundary conditions, and the initialisation component of the script are working. The solver for the full governing equation of this project was not used. Instead, a separate solver was written for the Orr-Sommerfeld (which is a simplified version of the project's solver by assuming $Fr = Sc = \infty$.) It also uses a untransformed domain $[-1, 1]$ instead of $[-\infty, \infty]$.



a) b)

Figure 5.2: Comparison of eigenspectra on the complex ω-plane of a) Couette Flow b) Poiseuille Flow under Re=2000 from the solver this project use and data from Schmid and Henningson (2001) . x-axis represents ω_r. y-axis represents ω_i.

Method/Source	Input	ω_0	k_0
Monkewitz (1988)	$R = -1\ a = 2\ Re = 11.3$	$1.008 + 0i$	no numerical value available
Secant Method		$1.008 + 0i$	$0.804 - 0.562i$
Monkewitz (1988)	$R = -1.105\ a = 1.34\ Re = 12.5$	$0.990 + 0.061i$	$0.832 - 0.504i$
Secant Method		$0.992 + 0.061i$	$0.832 - 0.504i$

Table 5.1: Comparison of data from Monkewitz (1988) and the numerical scheme developed in this project

5.3.2 Validating transformation, wake flow profile and search method

Since the project is largely following the steps of Monkewitz (1988), naturally its data will be one of the past research data to compare with. Monkewitz (1988) did not show the plot of the eigenspectra on the complex ω-plane, but Hwang and Choi (2006), a paper that built upon Monkewitz (1988), did show an eigenspectra plot. Comparison with Monkewitz (1988) and Hwang and Choi (2006) can validate the wake profile to be used in the project, as well as the use of tangent transformation, where mistakes can be easily made. Note that the Orr-Sommerfeld solver is still being used.

Further more, Monkewitz (1988) searched for ω_0 at some parameter set, which provides validation for the saddle point Secant Search algorithm for this script. Table 5.1 summarised the comparison.

Note the slight difference between the result from this script and those from Monkewitz (1988). It is very possibly due to the different numerical method used. Monkewitz (1988) used shooting method with Runge-Kutta method, while this project used Chebyshev Collocation Method.

5.3.3 Validating the solver with Stratification and Viscosity

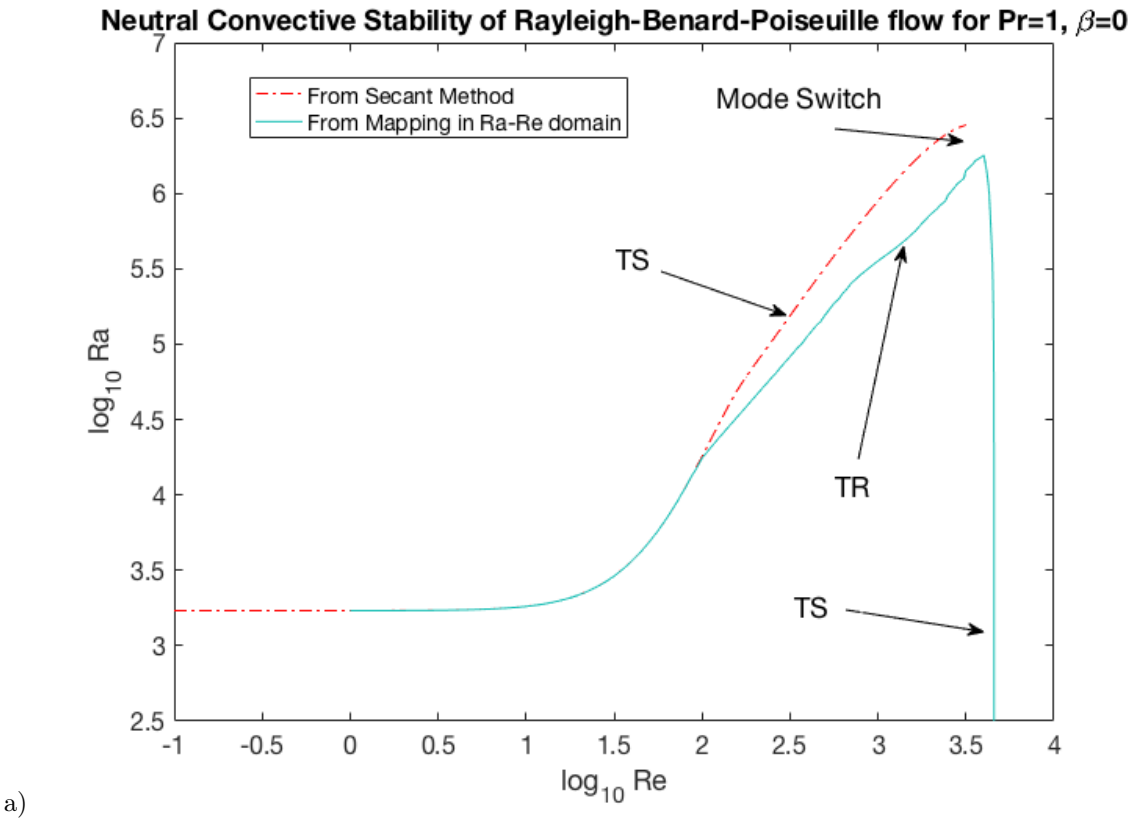
Up to this point, we have only validated the whole script architecture with an Orr-Sommerfeld solver. The Orr-Sommerfeld solver was first developed to assist the development of the entire script architecture. With all of them validated, we can start developing the solver to be used in this project.

Fujimura and Kelly (1988) solves a similar set of equations as this project for a Poiseuille-Couette flow in a stratified fluid. Although it is thermally stratified, it has made a Boussinesq Approximation, which makes ρ proportional to temperature T. After non-dimensionalisation, ρ is essentially the same as T. Fujimura and Kelly (1988) also have similar non-dimensional control parameters, in which Ra can be converted to Fr once the basic flow profile is known, and Pr is equivalent to Sc.

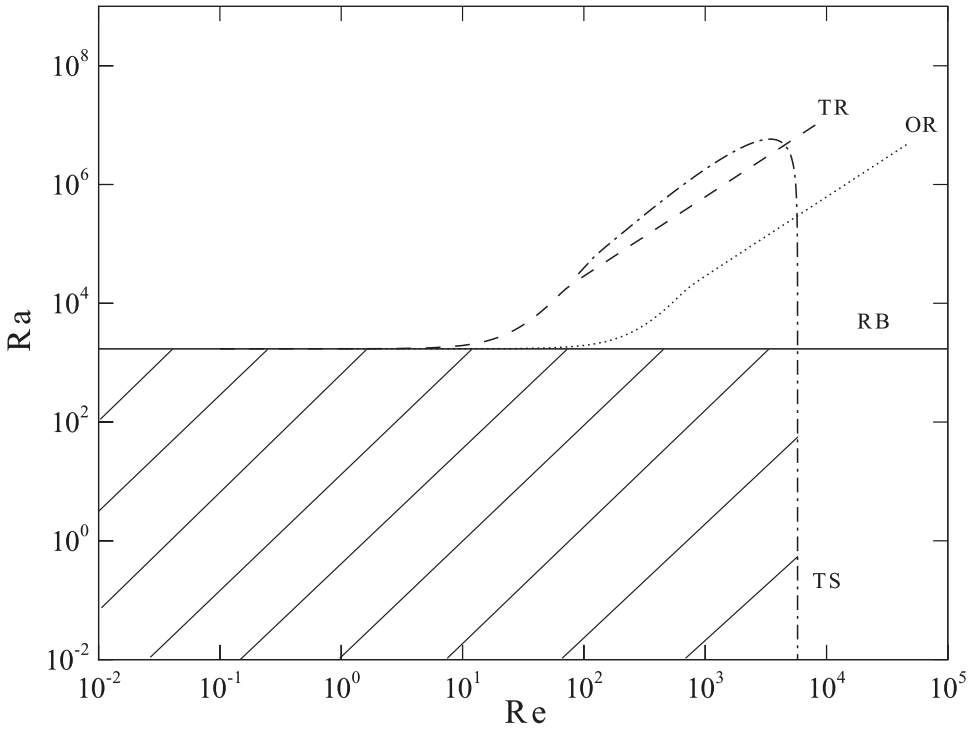
Fujimura and Kelly (1988) provided a large library of numerical data. After the conversion to the equivalent parameters, it is found that the solver developed for this project produce the exact same results as Fujimura and Kelly (1988), proving that the solver is written correctly.

Furthermore, the solver, combined with the search algorithm, is also able to reproduce the neutral curve for Transverse Rolls mode and Tollmien-Schlichting mode in Jerome et al. (2012, Fig. 2), which is shown in Figure 5.3. Since the Secant Search method is sensitive to the initial condition (which determines which saddle point it will converge to), one branch (blue) is found by the Secant Search method, while the other branch is found by mapping out the maximum imaginary part of all ω. Despite different methods, the consistency between results from this script and that of Jerome et al. (2012) has validated the solver.

On a side note, in the development process, results from Hazel (1972), which is solving the Taylor-Goldstein equation under a mixing layer profile, is also compared against with another Taylor-Goldstein equation solver written for this script architecture. Since it would only validate what was already validated in previous sections, it is not presented here. Such work, however, does highlight the robustness and flexibility of the script this project has developed.



a)



b)

Figure 5.3: Comparison of Neutral Stability diagram in *Ra-Re* plane of Rayleigh-Benard-Poiseuille flow for $Pr = 1$ between data from a) this solver (after conversion to Ra), and data from b) Jerome et al. (2012). Key modes: Transverse Rolls (TR), Tollmien-Schlichting waves (WS).

Chapter 6

Results and Discussion

This chapter reviews results generated from the solver. Keeping in line with the analysis by Meunier (2012), the parameters study fixed $Sc = 700$, while allowing Fr, Re to vary. α is varied to find Absolute and Convective instability of the most unstable $\omega_{i,0}$ and $\omega_{i.max}$, while 2D assumption allows us to put $\beta = 0$ in our analysis. Under limited project timeframe, only the $\theta = 30°$ and $90°$ cases are studied, which represents the cases for shear aligning with stratification and not aligning with stratification respectively.

The first two section of the chapter will first cover the case when $\theta = 90°$, when the wake experiences the full effect of high Froude number. As mentioned in previous sections, there should be no new unstable mode, and that stratification is expected to stabilise the flow.

Then we will look into the case when $\theta = 30°$. According to Meunier (2012), there should be a new unstable mode at a low Froude number. This would be discussed in Section 6.3.

Note that, unless specified otherwise, we will assume $R = -1.105$ and $a = 1.34$ for the wake profile. We will also assume $L = 300$ in the script, to stay consistent with the experimental set up of Meunier (2012). However, in reality, L have no effect on the system because of the Boussinesq Approximation,. Its effect would be cancelled out by using Fr instead of Fr_g in the calculation.

Also note that, unless specified otherwise, 'Froude Number' will be referred as the Fr that represents stratification strength instead of Fr_g. This is to keep consistency with the terminology of Meunier (2012).

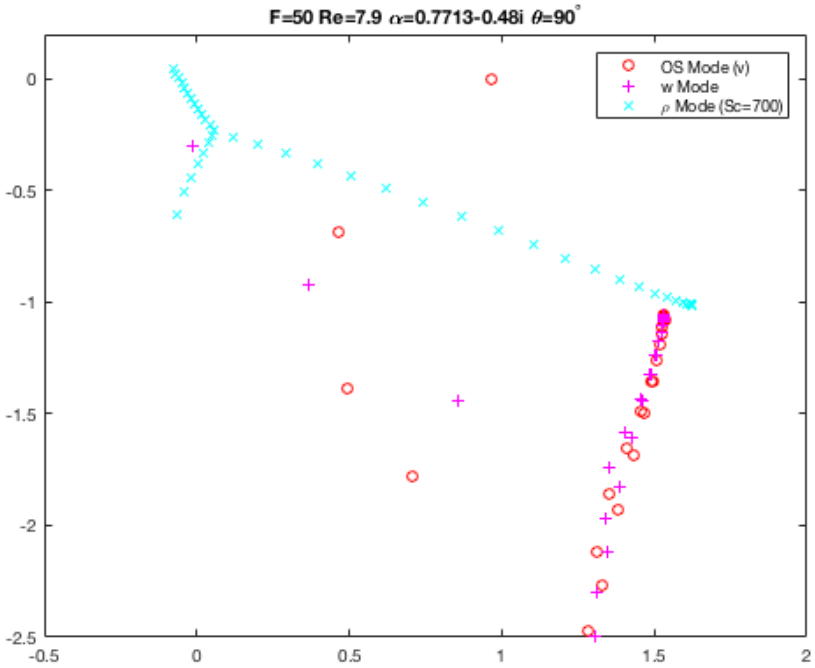
6.1 Understanding different branches at $\theta = 90°$ at High Froude number

The Froude number is inversely proportional to stratification strength, so the higher the Froude number, the weaker the stratification. In the equations, Fr couples the density perturbation ρ with vertical velocity perturbation v. Also, at $\theta = 90°$, the spanwise velocity perturbation w is uncoupled from ρ. This implies that, at $\theta = 90°$ and $Fr \to \infty$, the system of equations (4.20) are all uncoupled, and v should show the same eigenspectra as the Orr-Sommerfeld equation.

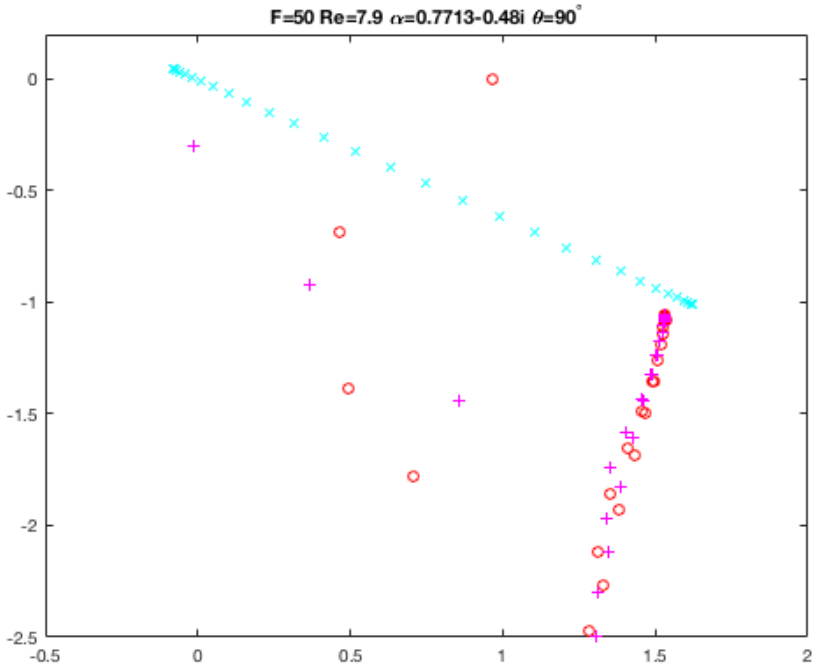
While physically this is the same as solving for Orr-Sommerfeld equation, analysing this case is useful for validating the script and giving a better understanding of the eigenspectra itself. Most importantly, it gives an indication to which perturbation variable lead to which branch of eigenvalues in the eigenspectra plot, before coupling of the variables begins. Figure 6.1 shows how each perturbation variables contribute to the eigenspectra, at the parameter set which is equivalent to when the Orr-Sommerfeld equation is neutrally Absolutely Stable. $Fr \to \infty$ is taken as $Fr = 50$ numerically.

As shown in Figure 6.1, ρ, v, w each contributes to the eigenspectra separately. While those contributed by v are completely identical to what would be found from purely solving the Orr-Sommerfeld equation, it is also important to note that the most unstable ω_j (red circle at the top of the figure) remains to be the most unstable when compared to ω_j from the other two perturbation variable.

As for the other two new branches, the S and P branch of w mode behaves similarly to that of v, because the uncoupled equation of v and w are very similar (only differ by the D^2U term in the Orr-Sommerfeld equation, which governs the A-Branch). The uncoupled density equation gives modes that are very close to $c_i = (\omega_j/\alpha)_i = 0$, where c is the phase speed. Note that if $Sc \to \infty$, i.e. turning off the density diffusive terms, the ρ modes would become a straight-line continuous branch with $c_i = 0$. This is because at $Fr \to \infty$ and $Sc \to \infty$, the transport equation for ρ becomes $\omega = \alpha U$. Also note that this branch is dependent on the mesh density N when Sc is finite.



a)

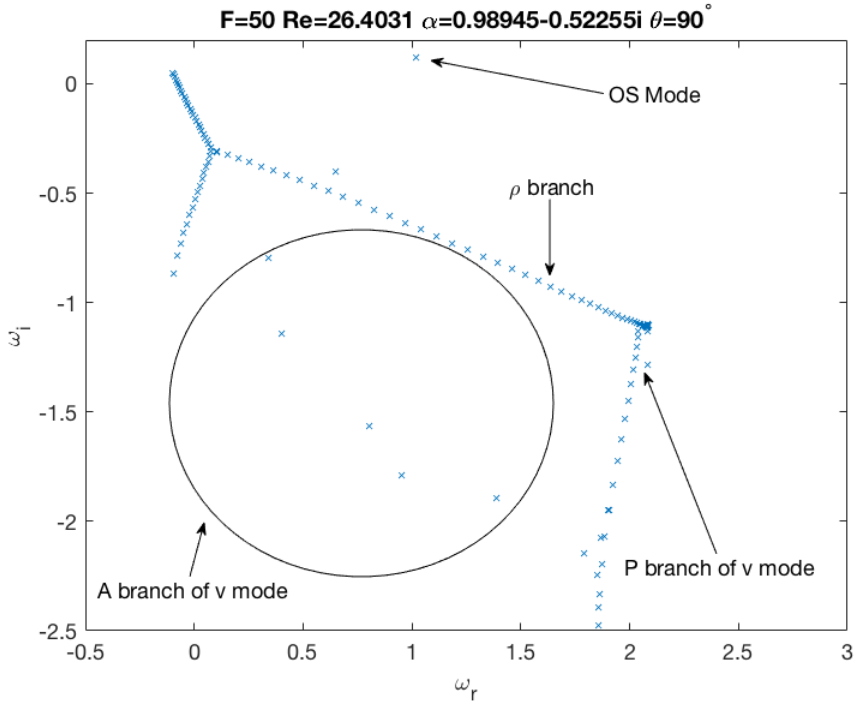


b)

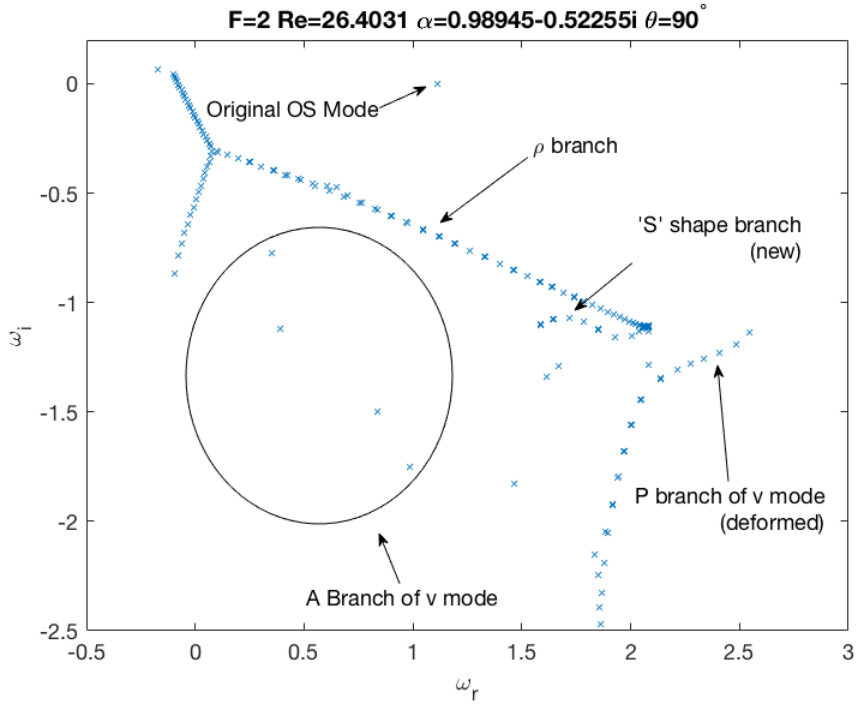
Figure 6.1: Breakdown of Eigenspectra by variables, at $Fr = 50$, $Re = 7.9$, $\alpha = 0.7713 - 0.48i$. $\theta = 90°$ and a) $Sc = 700$; b) $Sc = \infty$.

6.2 Decreasing Froude number with $\theta = 90°$

In this section, we decrease the Froude number to increase the stratification. Multiple parameter studies are performed in the process by using different criteria to set Re and α, including but not limited to, finding neutral Re for absolute instability (with α_0) at various Fr, or setting Re and α_0 at the original neutral OS mode. The following section will only discuss the studies that show significant physical implication.



a)



b)

Figure 6.2: Comparison of Eigenspectra Branches at a) high and b) medium Froude number with $Re = 26.4031$, $\alpha = 0.98945 - 0.552255i$, $\theta = 90°$. $Sc = 700$.

6.2.1 Eliminating w modes

Since at $\theta = 90°$ and assuming $\beta = 0$, w is essentially decoupled from the other two equations, we can eliminate the w mode in the study of the stratified wake when shear is aligned with stratification. This elimination speeds up computation significantly, and allow us to view the eigenspectra clearer.

6.2.2 Branches description

To introduce the new branches due to the coupling of ρ and v, a plot with the set of parameters that gives neutral Re at $Fr = 2$ is shown in Figure 6.2. For comparison, the same set of parameter but with $Fr = 50$ (very high Froude number) is also plotted for comparison.

First of all, one can observe the formation of a new branch that looks like a tilted 'S' shape. This branch begins at the right corner of the original ρ branch and spread horizontally to the negative ω_r direction. A further trial of different α_r and Fr shows that the right end of the 'S' shape always positions at around the right end of the ρ branch, which always has a value of about $2\alpha_r$. The branch always spans in length approximately equal to $1/Fr$.

Since Fr physically is also the non-dimensional Brunt Visalia frequency, it is believed these two branches are related to the internal gravity waves.

Notably, the original ρ branch stays the same when Fr changes, but only change with the product $ReSc$ (controlling diffusivity) and α. This implies that while Fr controls the coupling of ρ and v, the coupling does not affect the continuous branch of the ρ mode. Rather, the coupling affects ρ by creating a new branch.

Another very noticeable change is to the top of the P branch of v mode. It is deformed to the positive ω_r direction. A further trial of different α_r and Fr shows that horizontal length of such deformation, as compared to uncoupled OS mode, is again approximately equal to $1/Fr$. Such change is possibly due to the forcing from the new 'S' shape branch of ρ mode, or at least related to the 'S' shape branch.

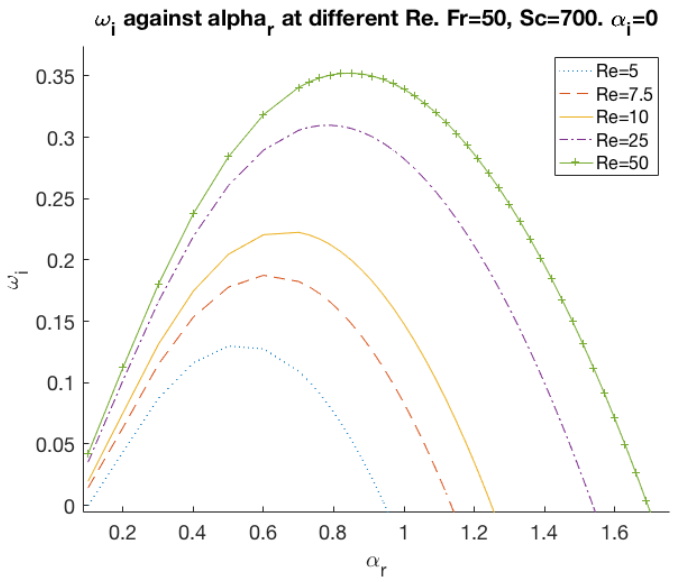


Figure 6.3: Destabilising effect of increasing Re on the OS mode in the Orr-Sommerfeld equation (almost no stratification, $Fr = 50$). $Sc = 700$, $\theta = 90°$.

6.2.3 General Effect of Re

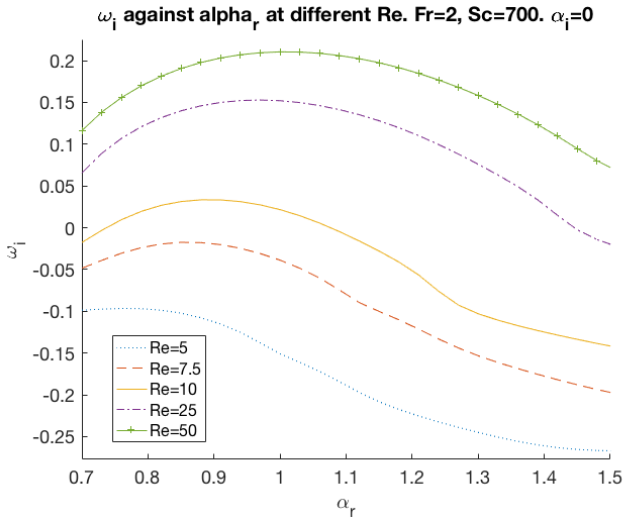


Figure 6.4: Destabilising effect of increasing Re on the OS mode at medium stratification ($Fr = 2$). $Sc = 700$, $\theta = 90°$.

In the discussion of the following two Subsections, we will focus on the sinuous mode originate from the Orr-Sommerfeld equation (hereby called the OS mode). It is most unstable mode, and remains as the most unstable one in various Re and Fr before it becomes stable.

It is known that the most unstable mode is becoming more unstable as Re increase in the Orr-Sommerfeld equation. This is shown in Figure 6.3. The question of this project is if the stratification changes such trend.

As shown in Figure 6.4, the stratification does not change this trend. Re remains a factor that controls the stability of the OS mode. We can also see that as the OS mode becomes more unstable, its peak shifts to the right on the ω_i-α_r plane. The increase in wavenumber of the peak instability is related to the Strouhal number of the vortex shedding experimentally. This would be discussed in Section 6.2.5.1.

Note that Figure 6.4 is plotted to show the stability of the OS mode, not the most unstable mode at each particular α_r. The OS mode is singled out to give clarity to the plot. When OS mode is Convectively Stable, it is possible that there are other modes that is more unstable than the OS mode (e.g. the top of the ρ branch). The OS mode, however, remains to be the most unstable when it is Convectively Unstable in our parameter study when $\theta = 90°$.

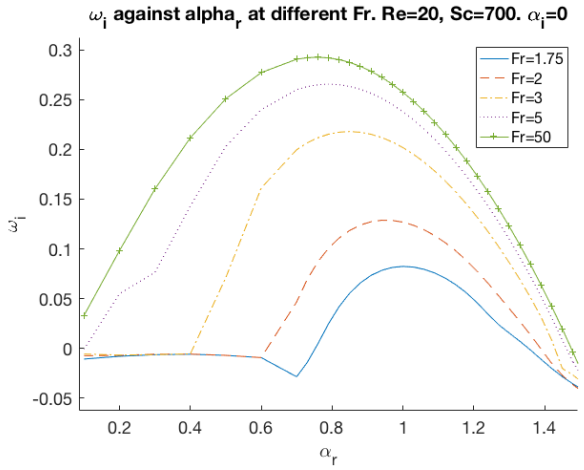


Figure 6.5: Stabilising effect of decreasing Fr on the OS mode at medium $Re = 20$. $Sc = 700$, $\theta = 90°$.

6.2.4 General Effect of Fr

As seen in Figure 6.4, Fr may also have an effect on the stability of the OS mode. It is indeed the case in our parameter study, that decreasing Fr (increasing stratification) can stables the OS mode. Figure 6.5 shows an example at $Re = 20$. The stabilising effect of stratification is consistent with past research, as mentioned in Section 2.4, as long as the shear is aligned with the stratification.

Note that the horizontal line at lower left corner is from another branch, but not from the OS mode. They are Convectively Stable and should be ignored.

6.2.5 Absolutely Unstable Modes

Despite having new branches, these branches remain very stable. There is no new unstable mode found throughout the parameter study.

As for the the OS mode, which remains to be the most unstable mode, it is stabilised by the decreasing Froude number. Such trend is observed in the finding the neutral Re at each Froude number by the Secant Search method described in Section 5.2.2.3. The result is shown in Figure 6.6.

Comparing with Figure 2.3a, Figure 6.6 shows a consistent trend to the experimental ones observed by Meunier (2012, Fig. 6) and Boyer et al. (1989), confirming that the local linear parallel analysis is able to recreate and explain the experimental result. However, this local analysis cannot predict the actual Reynolds number for the transition to the oscillation regime, as the transition would be determined by the global analysis (see Section 2.3). Instead, it would only show a trend similar to the experimental data (a graph with a similar shape).

It is also interesting note that, beyond $Fr < 1.41$, the Secant Search Method fails to find a neutral Reynolds number. This would be explained in Section 6.2.6.

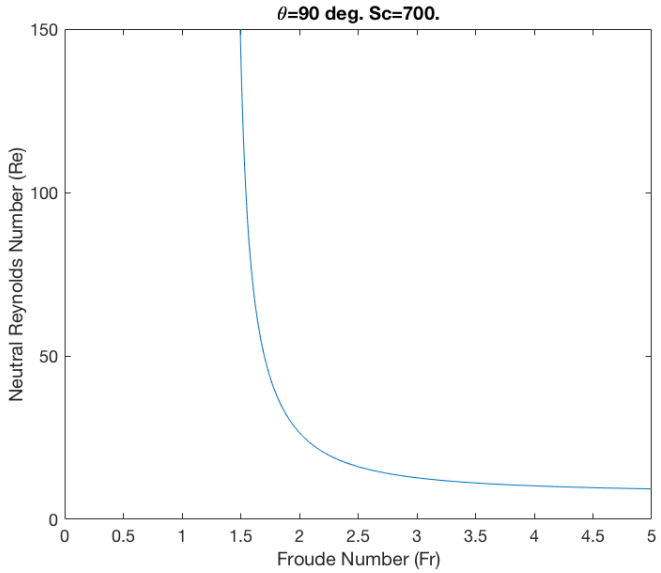


Figure 6.6: Neutral Reynolds number at different Froude number when the shear is aligned with stratification. $Sc = 700$.

6.2.5.1 Comparison of St number

From the neutral curve data generated from the Secant Search Method, it is also possible to predict the Strouhal number of the vortex shedding frequency of the wake. This is because, based on the theories described in Section 2.3, the singularity frequency from global linear stability analysis will determine the frequency of the vortex shedding, which is approximately close to the neutral absolute frequency ($\omega_{r,0}$ when $Re = Re_{neutral}$) from the local linear stability analysis. In other words, Strouhal number of vortex shedding will follow the same trend as $\omega_{r,0}$ when $Re = Re_{neutral}$.

This allows us to define a Strouhal number approximate:

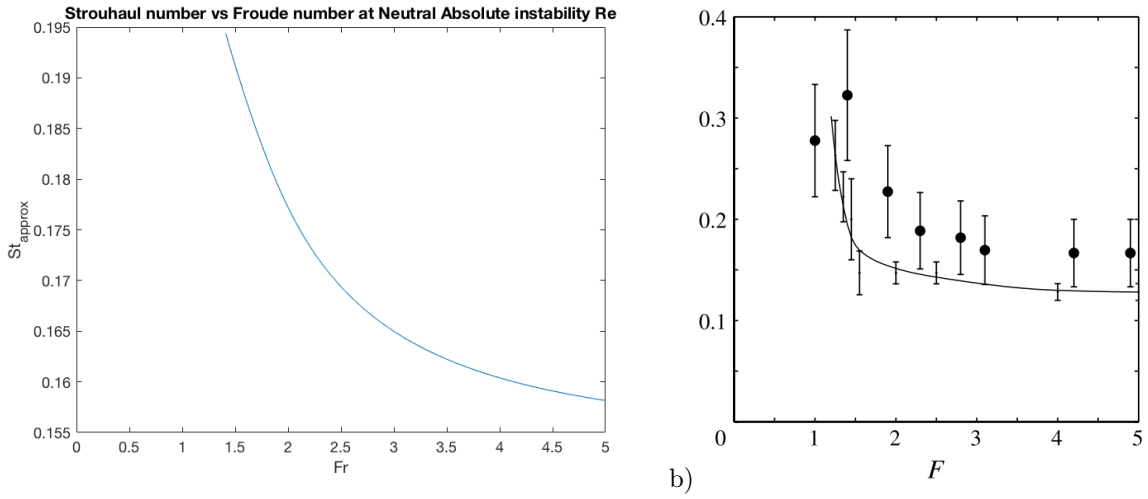


Figure 6.7: Comparison of St against Fr from a) the local stability analysis ($Sc = 700$, $\theta = 90°$) and b) the experiments by Meunier. (Meunier, 2012, Fig. 10). $Sc = 700$, $\theta = 90°$.

$$St \approx St_{approx} = \omega_{r,0}/2\pi. \tag{6.1}$$

Under such definition, we can plot St_{approx} against Fr. This is shown in Figure 6.7. It closely resembles the trend in Meunier (Fig. 10, 2012), but the magnitude is different. Such difference is partly due to the different way of non-dimensionalisation, in which Meunier (2012) uses free stream flow velocity and cylinder diameter as dimensional references, while this project uses the ones defined by the wake profile. Another reason is that the profile itself we have chosen for this analysis is possibly different the experimental one. It is expected that the profile is going to be increasingly different from $R = -1.105$, $a = 1.34$ (chosen for the non-stratified wake) as Froude number decreases. On the issue of determining R and a. please refer to Section 6.3.2.

6.2.5.2 Eigenfunction

After solving for the eigenvalue ω, the eigenfunction of a corresponding eigenvalue can also be output in the script. Although the eigenfunction is mainly used to ensure that there is no mode switching when tracing the track of a certain mode (by ensuring no sudden change in the eigenfunction), the eigenfunction can also be visualised to get a better understanding of the shape of the perturbation.

The eigenfunction is only output as a complex function of y, as the perturbation is written in the Fourier form in (4.11). In this project, only the shape of the most unstable Absolute Instability mode is of interest. To visualise the mode shape, one can recover the mode shape (time independent) with the following equation:

$$v'(x,y) = \mathfrak{Re}\{\tilde{v}(y)e^{i\alpha x}\}, \tag{6.2}$$

$$\rho'(x,y) = \mathfrak{Re}\{\tilde{\rho}(y)e^{i\alpha x}\}. \tag{6.3}$$

Notice that the time dependency is dropped and hence ω is dropped. This method does not recreate the perturbation in reality, but is only applied to visualise the flow.

To visualise the flow, which is fully 2D and has no w' component at $\theta = 90°$, the compressibility equation can be used to recover u'. It can be calculated in the following equation:

$$u'(x,y) = \mathfrak{Re}\{\frac{\partial\tilde{v}(y)}{\partial y}\frac{e^{i\alpha x}}{i\alpha}\}. \tag{6.4}$$

To visualise the velocity field, one can take the eigenfunction from the Absolutely Unstable mode, put $\alpha = \alpha_{r,0}$ in (6.4) and (6.2), and sums it up with the basic flow profile. Since linear instability

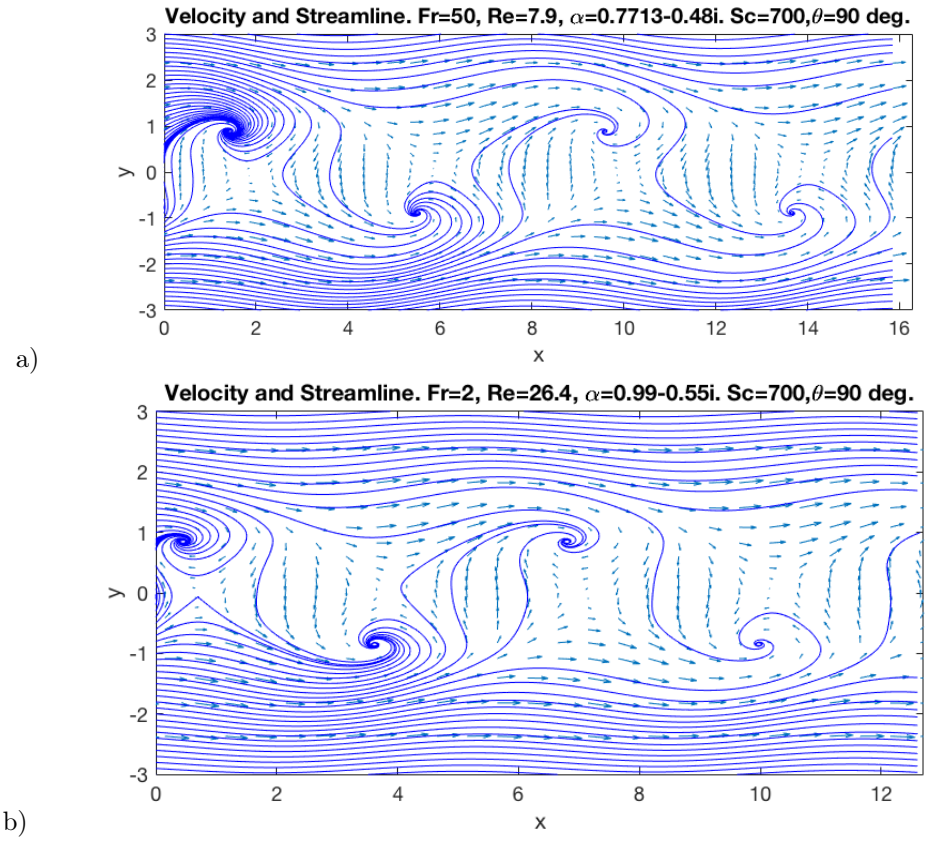


Figure 6.8: Streamline and Quiver plot of Velocity field after perturbation at a) almost no stratification, and b) medium stratification.

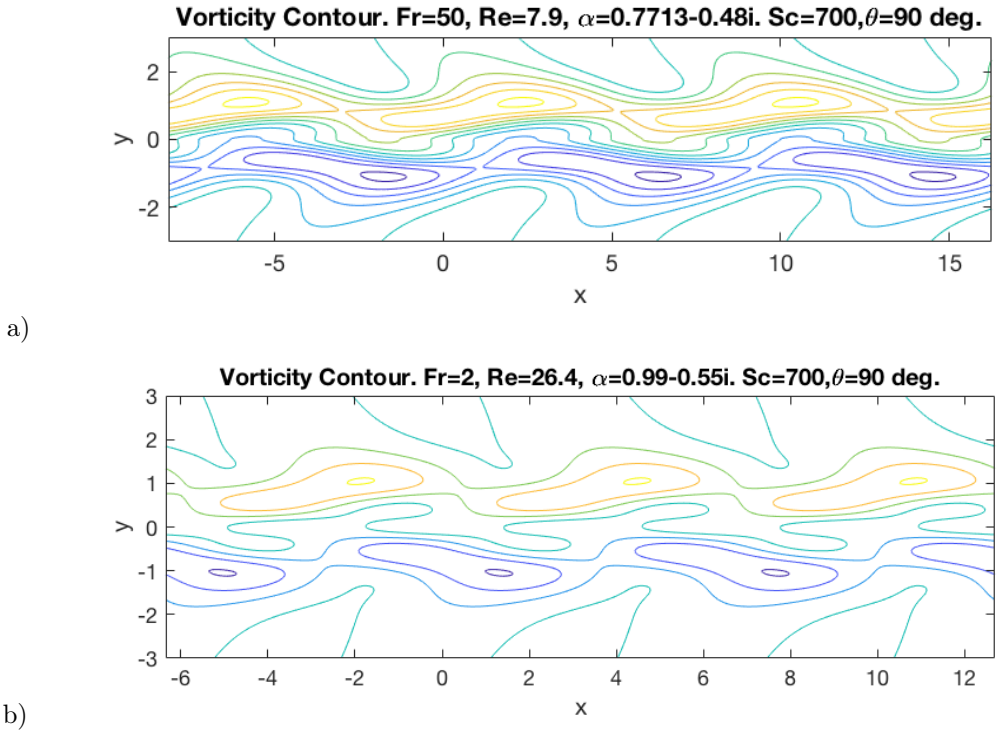


Figure 6.9: Vorticity Contour of Velocity field after perturbation at a) almost no stratification, and b) medium stratification.

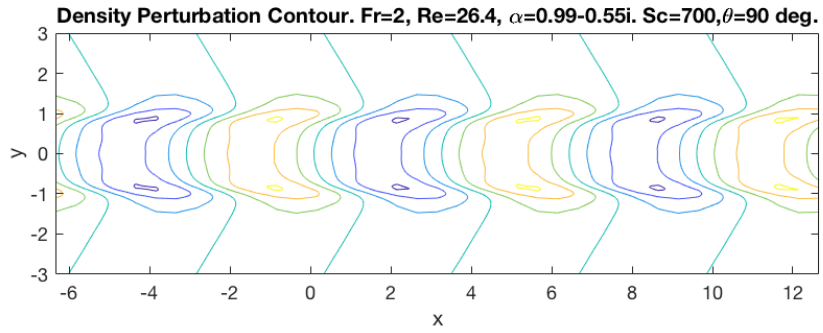


Figure 6.10: Density Perturbation Contour after perturbation at medium stratification.

assumes perturbation is infinitesimal, we need to define a finite amplitude ϵ to the perturbation. Such amplitude is $\epsilon = 1.6$ for the following visualisation.

As shown in Figure 6.8, the streamline and the vector field of the velocity field is shown. From the comparison, we can see that stratification is not significantly changing the shape of the eigenfunction. They remain very similar to that of the original OS mode.

We can also contour plot the vorticity, which is shown in Figure 6.9. Again, we can see little change the shape of the eigenfunction, confirming we are tracing the same mode as we increase the stratification.

ρ' can also contour plotted. By Boussinesq Approximation, there is no need to include the original stratification profile. Director contour of ρ' is shown in Figure 6.10. Note that at uncoupled state (very high Fr), ρ' mode is unrelated to the OS mode and has no physical meaning, and hence its mode shape is not plotted below.

We can see that while the vorticity and velocity have an alternating pattern, the density perturbation has a symmetric one. The density perturbation is likely to be driven by the sinuous perturbation v', which is an even function, while the u' is an odd function.

6.2.6 Neutral Curve of Convectively Unstable Mode and Howard's 1/4 criterion

From Figure 6.6, one can observe that neutral Reynolds number increase towards infinity, as Froude number decrease to a certain threshold value. Not only is this trend consistent with experimental result, but it can also be explained by the hallmark theory by Howard (1961). Generalising the work by Miles, Howard (1961) stated that

> 'there is [convective] stability if the local Richardson number Ri is everywhere greater than or equal to 1/4. '

This is the Howard's 1/4 criterion. Local Richardson number is defines as the ratio of stratification over shear:

$$Ri = N^{*2}/(\frac{dU^*}{dZ^*})^2 \approx \frac{1}{Fr^2}\frac{1}{(\frac{dU}{dZ})^2}, \tag{6.5}$$

which is fully defined in this analysis, as the profile is already defined. With $R = -1.105$, $a = 1.34$, we can get $(\frac{dU}{dZ})_{max} = 1.918$ from the profile definition in (4.3a).

Therefore, if the smallest local Richardson number in the whole flow field (largest shear, since under Boussinesq Approximation, Fr is globally constant) is larger 1/4, the flow is Convectively Stable. In other words, there exist a Froude number threshold, where below it the flow is unconditionally stable at all finite Reynolds number. For the profile of this project, this implies the system is linearly stable at $Fr < 1.087$.

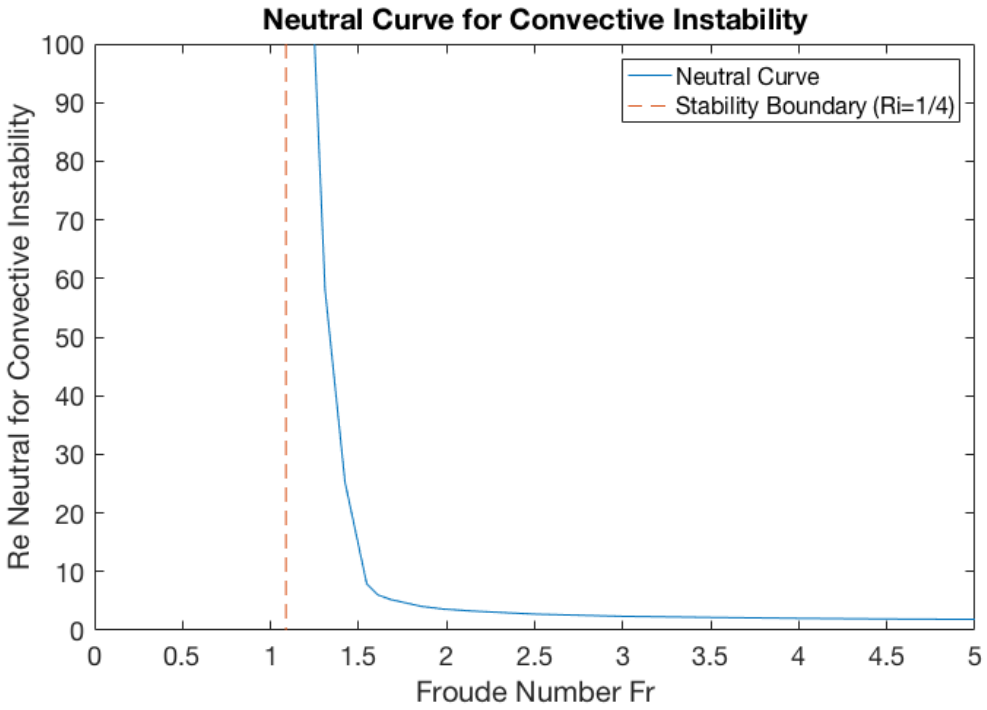


Figure 6.11: Neutral Reynolds number (for Convective Instability) at different Froude number when shear is aligned with stratification. $Sc = 700$, $\theta = 90°$. Note that the neutral Re tends to infinity before Fr decrease to the stability boundary defined by the Howard's criterion.

Figure 6.11 shows the neutral Re curve for Convective Instability on the $Re - Fr$ plane. It shows that, the theory indeed applies to the system, and correctly predicted the stability boundary. Beyond $Fr < 1.087$, the algorithm searches for Re that is tending to infinity and does not converge.

Moreover, it is supposed that neutral Reynolds number for Convective Instability is closely related to that of Absolute Instability. Therefore, a similar threshold exists for Absolute Instability. It explains why Secant Search Method fails to find neutral Reynolds number at Froude number lower than a certain threshold, as mentioned in Section 6.2.5.1.

6.3 Decreasing Froude number with $\theta = 30°$

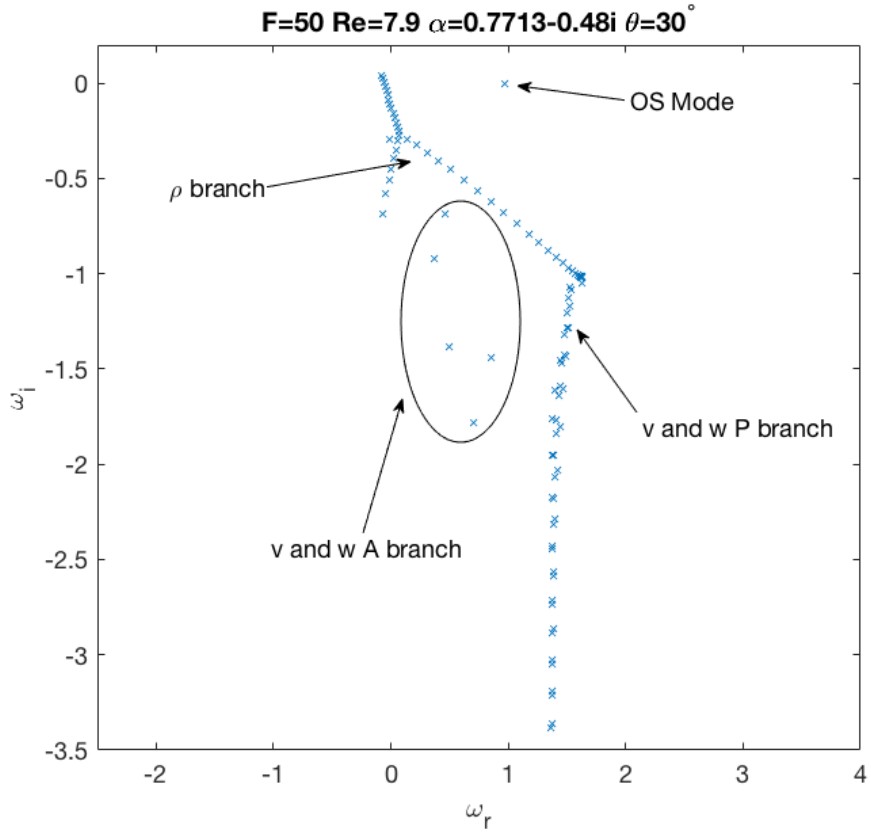
6.3.1 Branches description

First of all, it should be noted that at $\theta = 30°$, w cannot be eliminated, since it is coupled with ρ. This means there is always a w A-branch, as well as, w P- and S-branches in the eigenspectra.

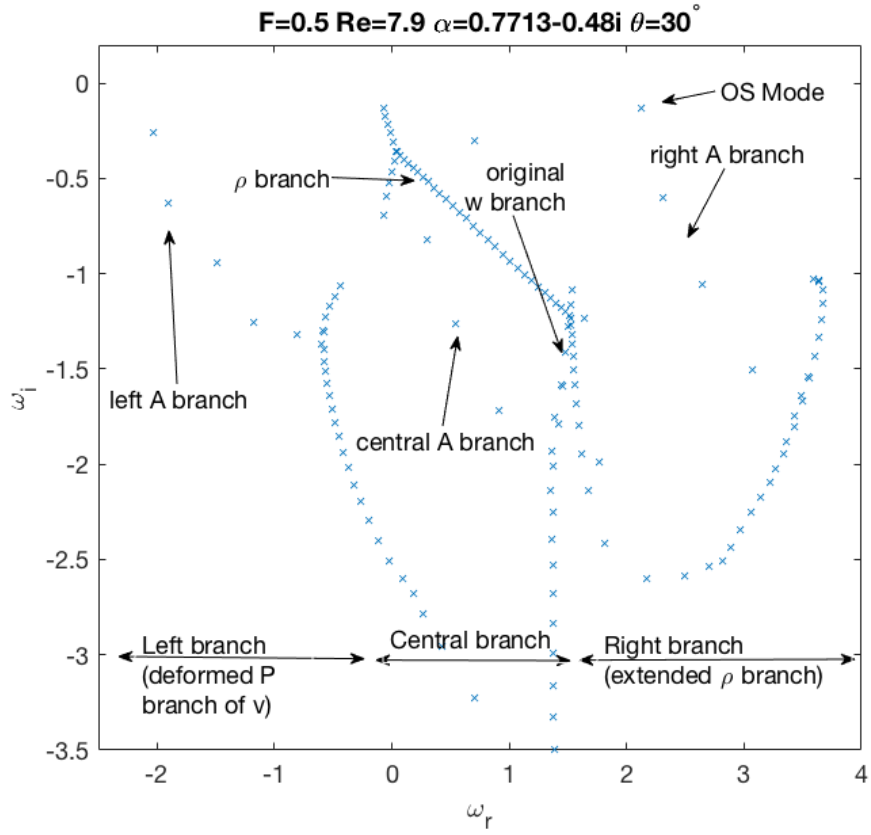
Similar to what have been done in Section 6.2.2, we can vary only Fr with a set of parameters to observe the new branches forming. This time the parameter set for the OS mode at neutral Re is used for observing the new branch formation. A plot with a low Froude number is also shown in Figure 6.12 as a comparison.

Two new branches are formed as result of coupling, but before looking into them, it should be noted that similar to the case in $\theta = 90°$, the ρ branch and the w P-branch changes little as Fr changes. Unlike the case of $\theta = 90°$, the ρ branch does move, but only when Fr is small enough, possibly because the coupling has an effect on it only after v is significantly changed by the forcing from the gravity term (i.e. $1/Fr$ large enough) that the effect of such forcing is feedback into ρ. Notably, w P-branch is also not affected by the change of Fr.

The two new branches initiated from the top of the P-branch of v, similar to the case in $\theta = 90°$, but their shape are significantly different. The 'S' shape branch found in $\theta = 90°$ is now the deformed P-branch of v, whereas the original deformed P-branch of v is now an extended part of



a)



b)

Figure 6.12: Eigenspectra Branches comparison at a) high and b) medium Froude number with $Re = 7.9$, $\alpha = 0.77 - 0.48i$,$\theta = 30°$. $Sc = 700$.

the ρ branch, forming an 'U' shape. The end of both branches follows the same rule as in $\theta = 90°$, that is, the real part of the end of the two branches are approximately $2\alpha_r \pm 1/Fr$.

For convenience, we will call the new branch on the left as the 'left branch, and the new branch on the right the 'right branch' in the following discussion. The original branch in the centre, which includes the original A-branch of w as well as the ρ branch and the w branch, will be called the 'central branch'.

Interestingly, there are also two more sub-branches emerging from the left and right branches. They are sparsely located, similar to an A-branch, but also follows their respective main continuous branches as Fr changes. Most importantly, the original OS-mode joined the A-branch of the right branch, meaning as Fr decrease, it will follow the right branch and be washed away to the right. This perhaps best explains the increasing Strouhal number trend with decreasing Froude number in Meunier (Fig. 6b, 2012), which is also shown in Figure 6.13, as the real part of the eigenvalue of right branch follows the approximation rule $\omega_r \approx 2\alpha_r + 1/Fr$.

It should also be noted that the A-branch of the central branch remains rather undisturbed by the Froude number change. In the next section, we will discuss the role of this particular A-branch in the stability of the flow.

6.3.2 Finding the AU branch at centre

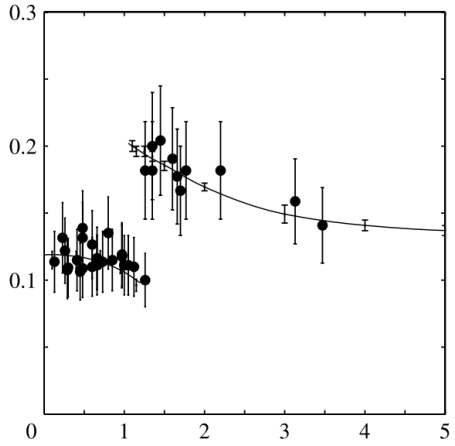


Figure 6.13: Strouhal number as a function of the Froude number at the onset of the von Karman vortex street when cylinder is tilted at $\theta = 30°$. Results are obtained experimentally (large symbols) and numerically (small dots fitted by solid lines). (Fig. 6b, Meunier, 2012).

According to Meunier (2012), in the case when shear is not aligned with stratification, and when the Froude number is low enough, there is a branch switch. Further decrease of Fr will destabilise instead of stabilising the flow, as shown in Figure 2.3. Furthermore, the Strouhal number of the vortex shedding in the regime of low Froude number remains approximately at the level similar to that of a non-stratified wake, as shown in Figure 6.13. The Strouhal number hinted that there might be an Absolutely Unstable mode at the central branch (right and left branch should give rise to Strouhal number roughly inversely proportional Froude number).

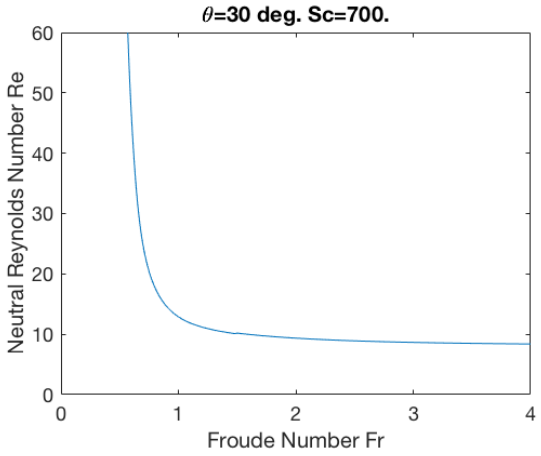


Figure 6.14: Neutral Reynolds number (for Absolute Instability) at different Froude number when shear is not aligned with stratification. $Sc = 700$, $\theta = 30°$.

In order to find this mode, a parameter study is performed to map out the most unstable mode at various combinations of Re and Fr and search for Absolutely Unstable modes. However, despite a lengthy and thorough search, no Absolutely Unstable modes were found. The neutral curve is plotted on the Re-Fr plane, as shown in Figure 6.14, but only the right half of the curve of Figure 2.3b is found. Further decrease in Fr the Secant Search method will not converge, similar to the $\theta = 90°$ case. Furthermore, by only varying Fr with fixed α and Re, the study suggested that the central branch is either stabilising at low Fr (ρ branches), or unaffected by the changing Fr at low Fr (w branches and A-branch of the central branch), which are all stable. All evidence suggested that ω_j at the central branch are Absolutely Stable at low Fr, contradicting results from Meunier (2012).

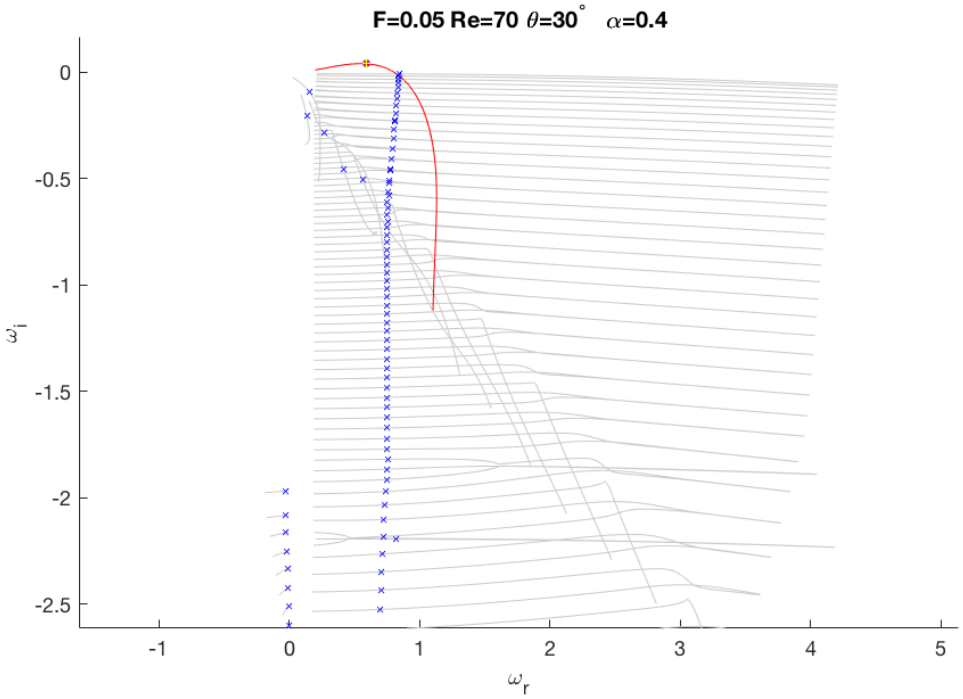


Figure 6.15: $\omega_j(\alpha)$ tracks when $\alpha \in [0.1, 3]$, at low Froude number when shear is not aligned with stratification. The grey lines are the tracks of each $\omega_j(\alpha)$ on the complex ω-plane. The blue crosses are points of the Eigenspectra at $\alpha = 0.4$ on the complex ω-plane. The track highlighted in red is the Convectively Unstable Mode at the Central Branch. $Sc = 700$, $\theta = 30$.

However, during the parameter study, there is one mode in the A-branch of the central branch that is found to be Convectively Unstable, which is a necessary condition for Absolute Instability. This mode is shown in Figure 6.15. To expand the search, a few trial was attempted to change also the parameters not currently studied in this project, including the wake profile parameter R and a, as well as Sc.

It was found that, while changing Sc basely affect the mode, R and a are very effective in destabilising the mode. Furthermore, it is found that with the right combination of R and a, such mode **can** be Absolutely Unstable. A few combination of the parameter used are listed in Table 6.1.

R	a	Re	Fr	$\alpha_{i,0}$
-2.5	2.5	70	0.05	-0.3
-1.105	8	20	0.05	-0.4

Table 6.1: A few example of parameter sets that would generate Absolute Instability at low Froude number

Note that Table 6.1 is only listed to suggest the possibility of an unstable mode at the central branch that might be responsible to the destabilising trend observed in Meunier (2012). These values might be unphysical, and remain a subject for further investigation.

However, results as such allow us to conclude that, **the strong stratification can destabilise the linearised system, not by the changing control parameter like Fr, but by changing the wake profile (the basic flow) itself**. Judging from the parameter tested, stratification probably need to result in a larger reverse flow (more negative R) and a stiffer profile (higher a) to destabilise the equations, which in turn destabilise the flow with a profile of stronger shear.

Chapter 7

Conclusion and Future Work

In this report, we have briefly reviewed the theory of Hydrodynamic stability and Absolute and Convective Instability. These theories formed the basis for the local stability analysis performed in this project. We have also reviewed some past research on both the wake flow and different classes of stratified flow. We have understood that stratification tends to stabilise the flow when the shear is aligned with stratification, and have understood the celebrated Howard's criterion for stability.

Using the same method from the local stability analysis for unstratified wake by Monkewitz (1988), we attempted to do the same analysis for a stratified wake. A solver script was written in MATLAB to perform such analysis with a Chebyshev Collocation Method for discretisation. We have also compared the results from the solver script with multiple past research to validate the solver script. With the rich amount of validation, the script is proven to be both flexible to adopt different governing equations, and robust enough to solve for different linearised systems accurately.

With the script ready, a local stability analysis is performed for the stratified wake based on the linear and parallel assumption. In the case when stratification is aligned with shear, the results from the solver script shows high consistency with past research and experimental results. The OS mode remains to be the dominating mode for Absolute Instability of the stratified wake, and the mode shape is similar even under medium stratification. Generally, the OS mode destabilise in high Re, and stabilise when Fr decreases. We have also confirmed that the Howard's Criterion is valid for the OS mode. We have found no new unstable mode in low Fr.

In terms of the branches in the eigenspectra, we can see the coupling of ρ and v creates two new branches. They are both related to $1/Fr$, implying they might have strong correlation with the internal gravity waves. However, the internal gravity waves are not the focused of this project. Knowing that the new branches are stable, no further research was performed in this direction.

However, in the case when stratification is not aligned with stratification, where new mode were expected in the low Fr regime according to Meunier (2012), no Absolutely Unstable mode was found despite the best effort. According the St number of the vortex shedding from the experiments, new Absolutely Unstable should be found in the central branch, but only a Convectively Unstable mode was found under the current set up of the project. This contradicts the experimental and numerical results by Meunier (2012).

To resolve this seemingly contradictory result, the search for the new mode was expanded beyond the scope of this project. In the hope that the Convectively Unstable mode might become Absolutely Unstable too, the wake profile parameters R and a are tested to try destabilise the mode. It was found that the two parameters are significantly changing the stability of such mode, suggesting a strong possibility that such mode might be the Absolutely Unstable mode responsible for the mode switch found in the experiments by Meunier (2012). If such possibility is indeed true, we can not only resolve the seemingly contradictory results between experiments and local stability analysis, but also be able to establish one important point in the theory: the strong stratification can destabilise the linearised system, not by the changing control parameter like Fr in the governing equation, but by changing the wake profile itself.

Therefore, the most immediate future work to be done after this project is to establish the relationship between stability (ω_i) and the wake profile parameters (R,a). Once the relationship is established, it is highly recommended to run a full numerical simulation (DNS) to extract the true profile at each stream-wise location and confirm the stability of the profile. After that, a global analysis can also be performed to allow us to predict quantitatively the stability boundary (neutral curve on Re-Fr plane) of a stratified wake.

Bibliography

D L Boyer, P A Davies, H J S Fernando, and Xiuzhang Zhang. Linearly Stratified Flow Past a Horizontal Circular Cylinder. *Philos. Trans. R. Soc. London. Ser. A, Math. Phys. Sci.*, 328 (1601):501–528, 1989. ISSN 00804614.

Julien Candelier, Stéphane Le Dizès, and Christophe Millet. Shear instability in a stratified fluid when shear and stratification are not aligned. *J. Fluid Mech.*, 685(May):191–201, 2011. ISSN 0022-1120. doi: 10.1017/jfm.2011.306.

Kevin K. Chen and Geoffrey R. Spedding. Boussinesq global modes and stability sensitivity, with applications to stratified wakes. *J. Fluid Mech.*, 812:1146–1188, 2017. ISSN 0022-1120. doi: 10.1017/jfm.2016.847.

J. M. Chomaz, P. Huerre, and L. G. Redekopp. Bifurcations to local and global modes in spatially developing flows. *Phys. Rev. Lett.*, 60(1):25–28, 1988. ISSN 00319007. doi: 10.1103/PhysRevLett.60.25.

P. G. Drazin and W. H. Reid. *Hydrodynamic Stability*. Cambridge University Press, Cambridge, 2 edition, 2004. ISBN 9780511616938. doi: 10.1017/CBO9780511616938.

Kaoru Fujimura and Robert E Kelly. Stability of unstably stratified shear flow between parallel plates. *Fluid Dyn. Res.*, 2(4):281–292, 1988. ISSN 01695983. doi: 10.1016/0169-5983(88)90006-8.

K. S. Gage and W. H. Reid. The stability of thermally stratified plane {Poiseuille} flow. *J. Fluid Mech.*, 33(1):21–32, 1968. doi: 10.1017/S0022112068002326.

Philip Hazel. Numerical studies of the stability of inviscid stratified shear flows. *J. Fluid Mech.*, 51(1):39–61, 1972. doi: 10.1017/S0022112072001065.

L. N. Howard. Note on a paper of John W. Miles. *J. Fluid Mech.*, 10(March):509–512, 1961. ISSN 0022-1120. doi: 10.1017/S0022112061000317.

Patrick Huerre and Peter A Monkewitz. Local and Global Instabilities in Spatially Developing Flows. *Annu. Rev. Fluid Mech.*, 22(1):473–537, jan 1990. ISSN 0066-4189. doi: 10.1146/annurev.fl.22.010190.002353.

Patrick Huerre and M. Rossi. Hydrodynamic instabilities in open flows. In Claude Godreche and Paul Manneville, editors, *Hydrodyn. Nonlinear Instab.*, pages 81–294. Cambridge University Press, Cambridge, 1998. doi: 10.1017/CBO9780511524608.004.

Yongyun Hwang and Haecheon Choi. Control of absolute instability by basic-flow modification in a parallel wake at low Reynolds number. *J. Fluid Mech.*, 560:465, 2006. ISSN 0022-1120. doi: 10.1017/S0022112006000140.

Jean-Yves Tinevez. Simple Tracker, mar 2016. URL `https://uk.mathworks.com/matlabcentral/fileexchange/34040-simple-tracker`.

J. John Soundar Jerome, Jean-Marc Chomaz, and Patrick Huerre. Transient growth in Rayleigh-Benard-Poiseuille/Couette convection. *Phys. Fluids*, 24(4), 2012. ISSN 10706631. doi: 10.1063/1.4704642.

W. Koch. Local instability characteristics and frequency determination of self-excited wake flows. *J. Sound Vib.*, 99(1):53–83, 1985. ISSN 10958568. doi: 10.1016/0022-460X(85)90445-6.

L D Landau and E M Lifshitz. *Fluid Mechanics*, volume 6 of *Course of theoretical physics*. Pergamon Press, London, 1959.

J T Lin and Y H Pao. Wakes in Stratified Fluids. *Annu. Rev. Fluid Mech.*, 11(1):317–338, 1979. ISSN 0066-4189. doi: 10.1146/annurev.fl.11.010179.001533.

S A Maslowe and J M Thompson. Stability of a Stratified Free Shear Layer. *Phys. Fluids*, 14(453), 1971. doi: 10.1063/1.1693456.

Patrice Meunier. Stratified wake of a tilted cylinder. Part 1. Suppression of a von Kármán vortex street. *J. Fluid Mech.*, 699(April):174–197, 2012. ISSN 0022-1120. doi: 10.1017/jfm.2012.92.

Peter A Monkewitz. The absolute and convective nature of instability in two dimensional wakes at low Reynolds numbers. *Phys. Fluids*, 31(May):999–1006, 1988. ISSN 00319171. doi: 10.1063/1.866720.

Steven a. Orszag. Accurate Solution of the Orr-Sommerfeld stability equation. *J. Fluid Mech.*, 50 (04):689–703, 1971. ISSN 0022-1120. doi: 10.1017/S0022112071002842.

Peter J. Schmid and Dan S. Henningson. *Stability and Transition in Shear Flows*, volume 142 of *Applied Mathematical Sciences*. Springer New York, New York, NY, 2001. ISBN 978-1-4612-6564-1. doi: 10.1007/978-1-4613-0185-1.

J A C Weideman and S C Reddy. A MATLAB Differentiation Matrix Suite. *ACM Trans. Math. Softw.*, 26(4):465–519, 2000.

Appendix A

Script Architecture and Interface

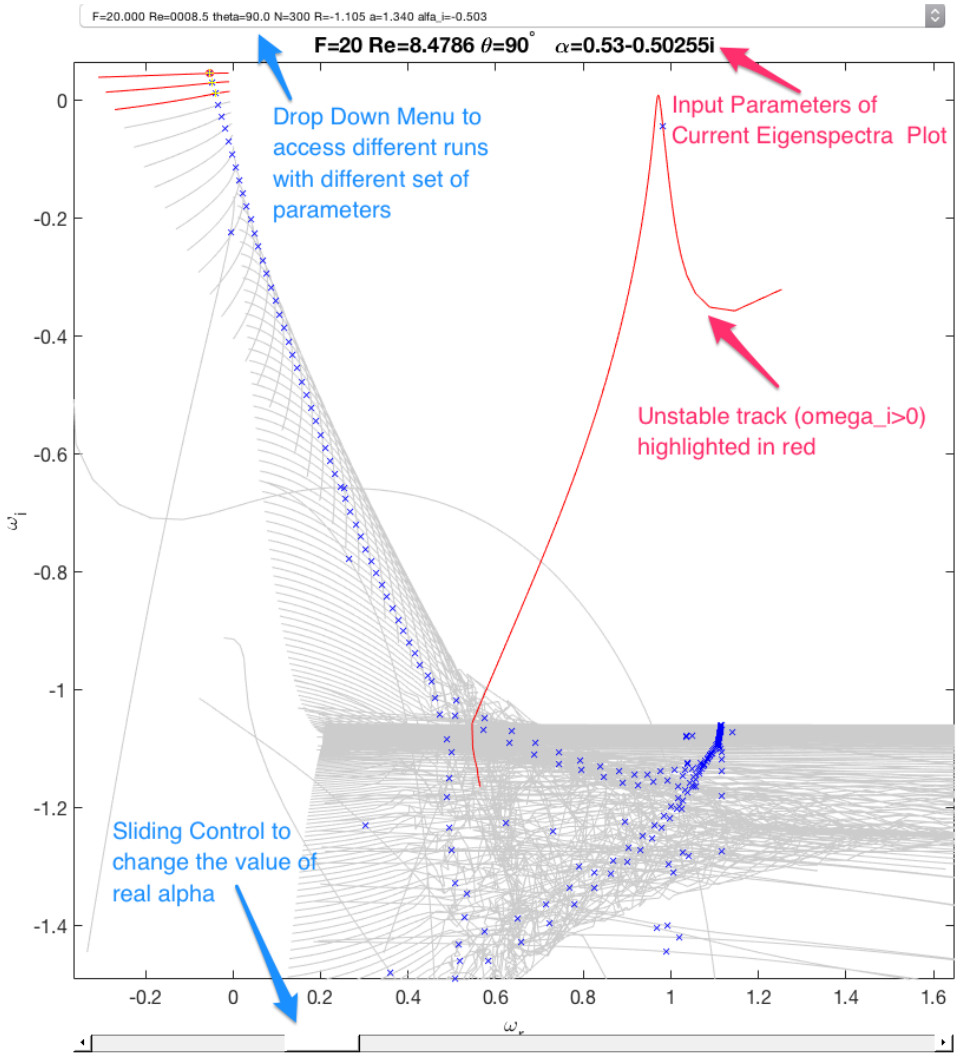


Figure A.1: Interface of the visualisation of results from the tracking algorithm, saved in a database.

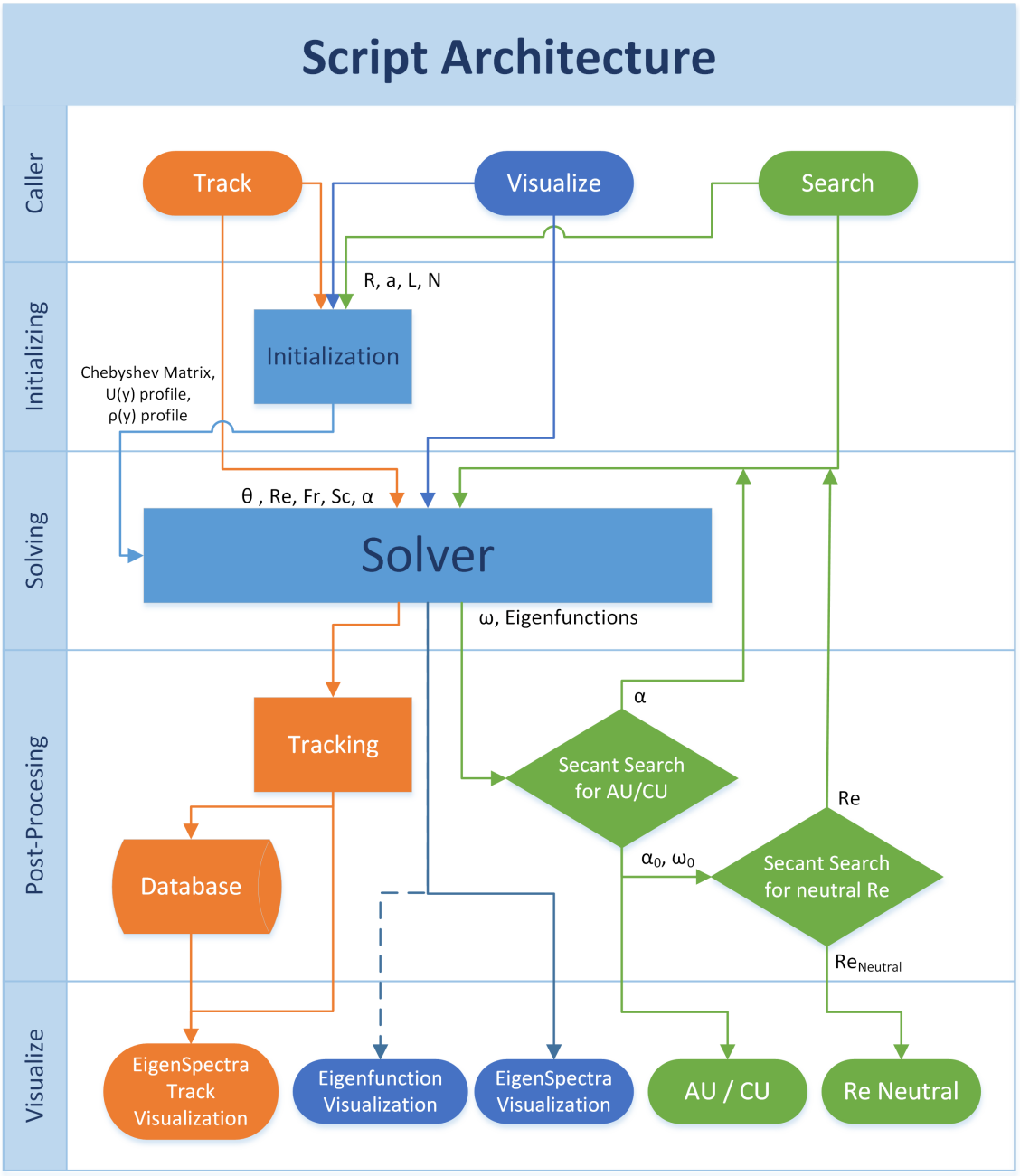


Figure A.2: The Data flow and Architecture of the Solver Script

Appendix B

The MATLAB Solver Script

B.1 Caller Scripts

B.1.1 'Meunier_2DBos_caller.m'

```
% Caller Script for the Solver to visualize the eigenspectra.
% Output the eigenspectra plot, and the first 4 most unstable eigenvalue
% clear all;
%% Initialization
addpath(genpath(pwd));% let files inside subfolder to be called
% global zi Re R a;
zi=sqrt(-1);

% Control Parameters
Re=7.9;
Sc=700;
% Profile Parameters
R=-1.105;
a=1.34;

%Straification
%Using Defintion of Froude number from P. Meunier 2012
F_Meunier=50;
L_Meunier=300;

rho_slope=-1/L_Meunier;
Fr2=F_Meunier^2/L_Meunier; % Definition of Fr_g in FYP defintion

% Input wave number
alpha=0.7713-0*zi;

%Tilting
theta=deg2rad(30);



%% ------ Mesh Initialization ---------------------------------------------
% global N;
N=100; %Mesh Density

% global I O D1 D2 D3 D4;
[ymesh,I,O,D1,D2,D3,D4]=con_diff_tran(N);

%% ------ Set basic state-------------------------------------------------
% global U DU DDU;
% global rho Drho;

[U,DU,DDU]=wake_profile(ymesh,R,a);
[rho,Drho]=rho_profile(ymesh,I,rho_slope,theta);


%% ====== OrrSommerfeld and Squire Equation Solver ========================
settings={theta,Re,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
[omega,Vec]=Meunier_2DBos(settings,alpha); %output all eigen values
% [omega,Vec]=Meunier_3DBos(settings,alpha,0); %output all eigen values

h_omg=figure('Name','omega','NumberTitle','off', 'WindowStyle', 'docked');
plot(real(omega),imag(omega),'x');
hold on
axis([-2.5 4 -3.5 0.2]);
title(['F=' num2str(F_Meunier) ' Re=' num2str(Re) ' \alpha='...
    num2str(alpha) ' \theta=' num2str(rad2deg(theta)) '^\circ']);
xlabel('\omega_r');
ylabel('\omega_i');
set(gca,'FontSize',18);
% h_c=figure('Name','c','NumberTitle','off', 'WindowStyle', 'docked');
% plot(real(omega/alpha),imag(omega/alpha),'x');
% hold on
% axis([-1.5 3.5 -2 1]);
% title(['F=' num2str(F_Meunier) ' Re=' num2str(Re) ' \alpha='...
```

```
%    num2str(alpha) ' \theta=' num2str(rad2deg(theta)) '^\circ']);


[omega1,omega2,omega3,omega4]=eigen_sort(omega(and(abs(real(omega))...
    < 2,imag(omega) < 2)))
% figure(h_omg);
% plot(real([omega1,omega2,omega3,omega4]),imag([omega1,omega2,omega3,...
%    omega4]),'ro');
% hold off;
% figure(h_c);
% plot(real([omega1,omega2,omega3,omega4]./alpha),...
%    imag([omega1,omega2,omega3,omega4]./alpha),'ro');
% hold off;

rmpath(genpath(pwd)); % let files inside subfolder not to be called
```

B.1.2 'eigen_func_selection_caller.m'

```
% Caller Script for the Solver to visualize the eigenspectra.
% Output the eigenspectra plot.
% Let user to choose an eigenvalue and plot the respective eigenfunction
% clear all;
%% Initialization
addpath(genpath(pwd)); % let files inside subfolder to be called
% global zi Re R a;
zi=sqrt(-1);

% Control Parameters
Re=7.9;
Sc=700;
% Profile Parameters
R=-1.105;
a=1.34;

%Straification
%Using Defintion of Froude number from P. Meunier 2012
F_Meunier=50;
L_Meunier=300;

rho_slope=-1/L_Meunier;
Fr2=F_Meunier^2/L_Meunier; % Definition of Fr_g in FYP defintion

% Input wave number
alpha=0.7713-0*zi;

%Tilting
theta=deg2rad(30);



%% ------ Mesh Initialization ---------------------------------------------
% global N;
N=300;

% global I O D1 D2 D3 D4;
[ymesh,I,O,D1,D2,D3,D4]=con_diff_tran(N);

%% ------ Set basic state-------------------------------------------------
% global U DU DDU;
% global rho Drho;

[U,DU,DDU]=wake_profile(ymesh,R,a);
[rho,Drho]=rho_profile(ymesh,I,rho_slope,theta);


%% ====== OrrSommerfeld and Squire Equation Solver ========================
        settings={theta,Re,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
        [omega,Vec]=Meunier_2DBos(settings,alpha); %output all eigen values
%% ====== Choosing eigenvalue ========================

%plot eigen spectra
figure(5);
% set(1, 'WindowStyle', 'docked');
plot(real(omega),imag(omega),'x');
axis([-1 3 -2 0.5]);
[chosen_r,chosen_i]=ginput(1);
omega_dist=abs(omega-(chosen_r+chosen_i*zi));
[~,chosen_pos]=min(omega_dist);
chosen_omega=omega(chosen_pos);
figure(5); hold on;
plot(real(chosen_omega),imag(chosen_omega),'ro');
hold off;

%% ====== Plotting respective eigenfunction ========================
eigfunc=Vec(:,chosen_pos);
```

```
figure(3);
plot(ymesh,abs(eigfunc(1:length(ymesh))));
figure(4);
plot(ymesh,abs(eigfunc((length(ymesh)*2+1):(length(ymesh)*3))));

% save eigenfunc2.mat eigfunc ymesh alpha chosen_omega D1 U DU rho;
```

B.1.3 'Meu_neutral_curve.m'

```
% This script generate the neutral curve on Re-Fr plane for AU.
% It searches for the saddle point (to find omega_0, k_0) on the
% k-plane, using secant method to find the point that satisfy d_omega/dk=0.
% Then iterate through another loop of Secant Search for the Re_neutral.

clear all;
%% Initialization
addpath(genpath(pwd)); % let files inside subfolder to be called
% global zi Re R a;
zi=sqrt(-1);

% Control Parameters
Sc=700;
% Profile Parameters
R=-1.105;
a=1.34;


%Straification
%Using Defintion of Froude number from P. Meunier 2012
L_Meunier=300;
rho_slope=-1/L_Meunier;

% Tilting
theta=deg2rad(90);



%% ------ Mesh Initialization ------------------------------------------------
% global N;
N=100; %Mesh Density

% global I O D1 D2 D3 D4;
[ymesh,I,O,D1,D2,D3,D4]=con_diff_tran(N);

%% ------ Set basic state----------------------------------------------------
% global U DU DDU;
% global rho Drho;

[U,DU,DDU]=wake_profile(ymesh,R,a);

[rho,Drho]=rho_profile(ymesh,I,rho_slope,theta);



%% ------ Mapping Parameters -------------------------------------------------
% Map Range
F_loop=[50:-10:10,9:-1:5,4.9:-0.1:2.4,[1.3+tan(0.8:-0.0025:0.1125)]];
Re_neutral=zeros(1,length(F_loop));
alpha_neutral=zeros(1,length(F_loop));


%% ====== Set up Search Initial Parameters ================================
k1=0.7713-0.48*zi; %First initial Position
k2=0.85-0.47*zi; %Second initial Position
Re1=7; %delta T= Temp diff between two wall, h= full channel heigh
Re2=9;

%% ====== Iterative through different Fr ================================
for j=1:length(F_loop)
    % Setting up Reynolds number (outer most loop)
    F_Meunier=F_loop(j);
    disp(['-------------------------' num2str(F_Meunier,10) '-------------------------']);

    % First Iteration (initial point) for Re

    %Straification
    %Using Defintion of Froude number from P. Meunier 2012
    Fr2=F_Meunier^2/L_Meunier; %Fr_g
    settings={theta,Re1,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
    disp([num2str(Re1)]);
    [f1_out,kr_out]=Meu_alpha_search(settings,k1,k2);
```

```
    % Setting up for second Iteration for Re
        Fr2=F_Meunier^2/L_Meunier;
        settings={theta,Re2,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
        disp([num2str(Re2)]);

    %% ------ Iterate in Secant Scheme (for Re) --------------------------

    while abs(f1_out)>1e-4 && abs(Re2-Re1)>1e-4
        k1=kr_out+0.01;
        k2=kr_out-0.01;

        [f2_out,kr_out]=Meu_alpha_search(settings,k1,k2); % Find AU Pinching Point
        disp([num2str(Re2)  '     ' num2str(kr_out) '   ' num2str(f2_out)]);
        % Setting up for next step
        Re_next=Re2-f2_out*(Re2-Re1)/(f2_out-f1_out);
        if Re_next<0
            Re_next=Re2/2;
        end
%         if Re_next>500
%             Re_next=mod(Re2*2,500);
%         end
        Re1=Re2;
        f1_out=f2_out;
        Re2=Re_next;


        settings={theta,Re2,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};

    end
    Re_neutral(j)=Re2;
    alpha_neutral(j)=kr_out;
    % Set up next iteration of Fr
    k1=kr_out*1.05;
    k2=kr_out*0.95;
    Re1=Re2*1.1;
    Re2=Re2/1.1;
end

%% ------- Saving Neutral Curve ----------------------------------------
Re_loop=Re_neutral;
alpha_loop=alpha_neutral;
save neutral.mat F_loop Re_loop alpha_loop

rmpath(genpath(pwd));% let files inside subfolder not to be called
```

B.1.4 'Meu_neutral_curve_CU.m'

```
% This script generate the neutral curve on Re-Fr plane for CU.
% It searches for the peak point on the omega_i-k_r plane,
% using secant method to find the point that satisfy d_omega_i/dkr=0.
% Then iterate through another loop of Secant Search for the Re_neutral.

clear all;
%% Initialization
addpath(genpath(pwd)); % let files inside subfolder to be called
% global zi Re R a;
zi=sqrt(-1);

% Control Parameters
Sc=700;
% Profile Parameters
R=-1.105;
a=1.34;

%Straification
%Using Defintion of Froude number from P. Meunier 2012
L_Meunier=300;
rho_slope=-1/L_Meunier;


% Tilting
theta=deg2rad(90);



%% ------ Mesh Initialization ----------------------------------------------
% global N;
N=225; %Mesh Density

% global I O D1 D2 D3 D4;
[ymesh,I,O,D1,D2,D3,D4]=con_diff_tran(N);
```

```
%% ------ Set basic state-----------------------------------------------------
% global U DU DDU;
% global rho Drho;

[U,DU,DDU]=wake_profile(ymesh,R,a);

[rho,Drho]=rho_profile(ymesh,I,rho_slope,theta);


%% ------ Mapping Parameters -------------------------------------------------
% Map Range
F_loop=[5:-0.5:2.2,[1+tan(0.8:-0.1:0.1)],1.09,1.088];
Re_neutral=zeros(1,length(F_loop));
alpha_neutral=zeros(1,length(F_loop));
%% ====== Set up Search Initial Parameters ================================
Re1=1; %delta T= Temp diff between two wall, h= full channel heigh

alpha_loop=[0.1:0.005:1];
alpha_loop_num=numel(alpha_loop);
%% ====== Iterative through different Fr ================================
for j=1:length(F_loop)
    % Setting up Reynolds number (outer most loop)
    F_Meunier=F_loop(j);
    Fr2=F_Meunier^2/L_Meunier;
    disp(['------------------' num2str(F_Meunier,10) '-----------']);
    Re_step=1;
    while Re_step>0.00001

        settings={theta,Re1,Fr2,Sc,ymesh,I,0,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
        % loop through different alpha
        omg_max_alpha_loop=zeros(1,alpha_loop_num);
        parfor i=1:alpha_loop_num
            alpha=alpha_loop(i);
        % ====== OrrSommerfeld and Squire Equation Solver =================
            %output all eigen values
            [omega,~]=Meunier_2DBos_no_w(settings,alpha);

            omg_max_alpha_loop(i)=eigen_sort(omega(real(omega)>0));
        end

        disp(['   ' num2str(Re1) '    '...
            num2str(max(imag(omg_max_alpha_loop)))]);
        if any(imag(omg_max_alpha_loop)>0)
            Re1=Re1-Re_step;
            Re_step=Re_step/10;
            Re1=Re1+Re_step;
        else
            Re1=Re1+Re_step;
        end
    end
    Re1=Re1-Re_step;
    disp(num2str(Re1));
    Re_neutral(j)=Re1;
end
%% ------- Saving Neutral Curve -----------------------------------------
% Re_loop=Re_neutral;
% alpha_loop=alpha_neutral;
save neutral_CU.mat F_loop Re_neutral

rmpath(genpath(pwd));% let files inside subfolder not to be called
```

B.1.5 'route_omg_track.m'

```
% This script gives track cusp-forming omega(k)
% along k in F_k in the complex plane of omega.
% at a specific F, Re and alpha_i.
% Attempting to recreate P. Meunier 2012
clear all;
addpath(genpath(pwd)); % let files inside subfolder to be called
zi=sqrt(-1);
%% ====== Initialization ================================================
% load neutral.mat;
% neutral_pos=281;
% alpha_i=imag(alpha_loop(neutral_pos));
% F=F_loop(neutral_pos);
% Re=Re_loop(neutral_pos);

F=0.05;
Re=70;
```

```
%% ------ alpha_r loop set up ----------------

alpha_loop=[0.1:0.01:2]-0*zi;
alpha_loop_num=length(alpha_loop);

F_loop=F*ones(1,alpha_loop_num);
Re_loop=Re*ones(1,alpha_loop_num);

%% ------ Other parameters set up ----------------
%global zi Re R a;
Sc=700;
theta=deg2rad(30);
R=-1.105;
a=1.34;

L_Meunier=300;

rho_slope=-1/L_Meunier;

%% ------ Mesh Initialization ---------------------------------------------
%global N;
N=225;

%global I O D1 D2 D3 D4;
[ymesh,I,O,D1,D2,D3,D4]=con_diff_tran(N);

%% ------ Set basic state-------------------------------------------------
%global U DU DDU;
%global rho Drho;

[U,DU,DDU]=wake_profile(ymesh,R,a);
[rho,Drho]=rho_profile(ymesh,I,rho_slope,theta);
%% ------ Compute all points at all alpha_r --------------------------------------
disp(['Calculating Eigenspectra with N=' num2str(N) ' ...']);

 % Cell Array that would store all points in the each _loop iteration
points=cell(1,length(F_loop));

parfor i=1:alpha_loop_num
    Re=Re_loop(i); %U=Umax at centre; h=half height channel
    alpha=alpha_loop(i);

    F_Meunier=F_loop(i);
    Fr2=F_Meunier^2/L_Meunier;

% ====== OrrSommerfeld and Squire Equation Solver ========================
        settings={theta,Re,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho};
        [omega,Vec]=Meunier_2DBos(settings,alpha); %output all eigen values
    points{i}=[real(omega),imag(omega)];
end

%% ------ Compute Tracks -------------------------------------
disp(['Calculating EigenValue Tracks...']);
[tracks,adjacency_tracks] = simpletracker(points);


%% ------ Creat and Export Video or visualize the track ----------------
%Video name
Video_name=['c_theta' num2str(rad2deg(theta)) '_F' num2str(F_loop(1))...
    '_Re' num2str(Re_loop(1)) '_N' num2str(N)];
% Title on the plot
title_name=['F=' num2str(F_loop(1)) ' Re=' num2str(Re_loop(1))...
    ' \theta='  num2str(rad2deg(theta)) '^\circ  '];

%Track visualization
omg_track_slider(points,tracks,adjacency_tracks,alpha_loop,...
    Video_name,title_name);
%Track visualization exported as video
% omg_track_video(points,tracks,adjacency_tracks,alpha_loop,...
%      Video_name,title_name);


%% ------ Database Export ------
settings={theta,Re,F,Sc,ymesh};
settings_others={N,R,a,L_Meunier};
database_save(points,tracks,adjacency_tracks,alpha_loop,settings,...
    settings_others);
disp('Done!');

rmpath(genpath(pwd)); % let files inside subfolder not to be called
```

B.2 Solver

B.2.1 'Meunier_2DBos.m'

```
function [omega,Vec]=Meunier_2DBos(settings,alpha)
% The Solver under \beta=0 assumption.
% with Viscousity (Re) and Diffusion (Sc).
% Bossinesq Approximation applied.
% With tilted cylinder. 2D wake assumed. No Squire Mode.
% Based on IC Aero FYP 2017, Lloyd Fung.
% INPUT: alpha, beta: stresswise and spanwise wavenumber
% OUTPUT: omega: the whole eigenspectra


%% ------ Initialization --------------------------------------------------
% global zi Re Fr2 Sc;
 zi=sqrt(-1);
 [theta,Re,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho]=settings{:};
%% ------ Mesh Initialization ---------------------------------------------
% global N;
% global I O D1 D2 D3 D4;
%% ------ Set basic state-------------------------------------------------
% global U DU DDU;
% global rho Drho;
%% ====== Uncoupled TaylorGoldstein Equation Solver ========================
% Loosely based on <<Stability and Transition in Shear Flows>> by PJ Schmid.
% and Dan D Henningson (2000), Section 3.1.4
% With additional buoyancy term and linearized equation for density

    %Define k^2
    k2=alpha*alpha;
    % Convenient nabla term
    cost=cos(theta);
    sint=sin(theta);

    nabla2=D2-k2*I;
    nabla4=k2*k2*I-2*k2*D2+D4;
    % Setting up M matrix Component
    LHS11=nabla2;
    LHS22=I;
    LHS33=I;
    % Setting up L matrix Component
    Lv=zi*alpha*(U*nabla2-DDU)-nabla4/Re;
    Lrho=zi*alpha*U-nabla2/Re/Sc;
    Lw=zi*alpha*U-nabla2/Re;

    RHS13=(k2/Fr2*sint)*I-(cost./Fr2)*D2;
    RHS23=cost/Fr2.*I;
    RHS31=Drho;
    % special cases for theta=90 and theta=0
    if theta==0
        RHS32=O;
    else
        RHS32=-Drho.*cos(theta)./sin(theta);
    end
    if theta==deg2rad(90)
        RHS23=O;
        RHS32=O;
    end
    % Solve the numerical matrix eigenvalue problem
    A=[Lv    O     RHS13;
       O     Lw    RHS23;
       RHS31 RHS32 Lrho ;
       ];
    B=[LHS11 O     O    ;
       O     LHS22 O    ;
       O     O     LHS33;
       ];

      [Vec,sigma]=eig(A,B);

%       [Vec,sigma]=eig(Lrho,LHS33); % For OrrSommerfeld Equation only

    omega=-zi*diag(sigma);

%% ------ Plotting all Eigenvalue -----------------------------------------
%   figure(1)
%   plot(real(omega),imag(omega),'x');
%   hold on
%   axis([-2 5 -2 0.5])
%   figure(2)
%   plot(real(omega/alpha),imag(omega/alpha),'x');
%   hold on
%    axis([-1.5 3.5 -2 1])
end
```

B.2.2 'Meunier_2DBos_no_w.m'

```
function [omega,Vec]=Meunier_2DBos_no_w(settings,alpha)
% The Solver under \beta=0 assumption and theta=90 assumption.
% w eliminated, giving faster computation.
% with Viscousity (Re) and Diffusion (Sc).
% Bossinesq Approximation applied.
% Set theta=90 deg.
% 2D wake assumed. No Squire Mode.
% Based on IC Aero FYP 2017, Lloyd Fung.
% INPUT: alpha, beta: stresswise and spanwise wavenumber
% OUTPUT: omega: the whole eigenspectra


%% ------ Initialization ---------------------------------------------------
% global zi Re Fr2 Sc;
 zi=sqrt(-1);
 [theta,Re,Fr2,Sc,ymesh,I,O,D1,D2,D3,D4,U,DU,DDU,rho,Drho]=settings{:};
%% ------ Mesh Initialization ----------------------------------------------
% global N;
% global I O D1 D2 D3 D4;
%% ------ Set basic state---------------------------------------------------
% global U DU DDU;
% global rho Drho;
%% ====== Uncoupled TaylorGoldstein Equation Solver ========================
% Loosely based on <<Stability and Transition in Shear Flows>> by PJ Schmid.
% and Dan D Henningson (2000), Section 3.1.4
% With additional buoyancy term and linearized equation for density

    %Define k^2
    k2=alpha*alpha;

    nabla2=D2-k2*I;
    nabla4=k2*k2*I-2*k2*D2+D4;
    % Setting up M matrix Component
    LHS11=nabla2;
    LHS22=I;
    LHS33=I;
    % Setting up L matrix Component

    if Re==Inf
        Lv=zi*alpha*(U*nabla2-DDU);
        Lrho=zi*alpha*U;
    else
        Lv=zi*alpha*(U*nabla2-DDU)-nabla4/Re;
        if Sc==Inf
            Lrho=zi*alpha*U;
        else
            Lrho=zi*alpha*U-nabla2/Re/Sc;
        end
    end

    RHS13=(k2/Fr2)*I;
    RHS31=Drho;

    % Solve the numerical matrix eigenvalue problem
    A=[Lv    RHS13;
       RHS31 Lrho ;
       ];
    B=[LHS11 O    ;
       O     LHS33;
       ];

     [Vec,sigma]=eig(A,B);

%    [Vec,sigma]=eig(Lv,LHS11); % For OrrSommerfeld Equation only

    omega=-zi*diag(sigma);

%% ------ Plotting all Eigenvalue ------------------------------------------
%   figure(1)
%   plot(real(omega),imag(omega),'x');
%   hold on
%   axis([-2 5 -2 0.5])
%   figure(2)
%   plot(real(omega/alpha),imag(omega/alpha),'x');
%   hold on
%    axis([-1.5 3.5 -2 1])
end
```

B.2.3 'cheb4c.m' (Weideman and Reddy, 2000)

```
function [x, D4] = cheb4c(N)
```

```
%  The function [x, D4] =  cheb4c(N) computes the fourth
%  derivative matrix on Chebyshev interior points, incorporating
%  the clamped boundary conditions u(1)=u'(1)=u(-1)=u'(-1)=0.
%
%  Input:
%  N:     N-2 = Order of differentiation matrix.
%               (The interpolant has degree N+1.)
%
%  Output:
%  x:       Interior Chebyshev points (vector of length N-2)
%  D4:      Fourth derivative matrix  (size (N-2)x(N-2))
%
%  The code implements two strategies for enhanced
%  accuracy suggested by W. Don and S. Solomonoff in
%  SIAM J. Sci. Comp. Vol. 6, pp. 1253--1268 (1994).
%  The two strategies are (a) the use of trigonometric
%  identities to avoid the computation of differences
%  x(k)-x(j) and (b) the use of the "flipping trick"
%  which is necessary since sin t can be computed to high
%  relative precision when t is small whereas sin (pi-t) cannot.

%  J.A.C. Weideman, S.C. Reddy 1998.

    I = eye(N-2);                    % Identity matrix.
    L = logical(I);                  % Logical identity matrix.

   n1 = floor(N/2-1);                % n1, n2 are indices used
   n2 = ceil(N/2-1);                 % for the flipping trick.

    k = [1:N-2]';                    % Compute theta vector.
   th = k*pi/(N-1);

    x = sin(pi*[N-3:-2:3-N]'/(2*(N-1))); % Compute interior Chebyshev points.

    s = [sin(th(1:n1)); flipud(sin(th(1:n2)))];   % s = sin(theta)

alpha = s.^4;                        % Compute weight function
beta1 = -4*s.^2.*x./alpha;           % and its derivatives.
beta2 =  4*(3*x.^2-1)./alpha;
beta3 = 24*x./alpha;
beta4 = 24./alpha;
    B = [beta1'; beta2'; beta3'; beta4'];

    T = repmat(th/2,1,N-2);
   DX = 2*sin(T'+T).*sin(T'-T);      % Trigonometric identity
   DX = [DX(1:n1,:); -flipud(fliplr(DX(1:n2,:)))];   % Flipping trick.
DX(L) = ones(N-2,1);                 % Put 1's on the main diagonal of DX.

   ss = s.^2.*(-1).^k;               % Compute the matrix with entries
    S = ss(:,ones(1,N-2));           % c(k)/c(j)
    C = S./S';

    Z = 1./DX;                       % Z contains entries 1/(x(k)-x(j)).
 Z(L) = zeros(size(x));              % with zeros on the diagonal.

    X = Z';                          % X is same as Z', but with
 X(L) = [];                          % diagonal entries removed.
    X = reshape(X,N-3,N-2);

    Y = ones(N-3,N-2);               % Initialize Y and D vectors.
    D = eye(N-2);                    % Y contains matrix of cumulative sums,
                                     % D scaled differentiation matrices.
for ell = 1:4
        Y = cumsum([B(ell,:); ell*Y(1:N-3,:).*X]);%Recursion for diagonals
        D = ell*Z.*(C.*repmat(diag(D),1,N-2)-D);   % Off-diagonal
      D(L) = Y(N-2,:);                           % Correct the diagonal
DM(:,:,ell) = D;                                 % Store D in DM
end

   D4 = DM(:,:,4);                   % Extract fourth derivative matrix
```

B.2.4 'chebdif.m' (Weideman and Reddy, 2000)

```
function [x, DM] = chebdif(N, M)

%  The function [x, DM] =  chebdif(N,M) computes the differentiation
%  matrices D1, D2, ..., DM on Chebyshev nodes.
%
%  Input:
%  N:        Size of differentiation matrix.
%  M:        Number of derivatives required (integer).
%  Note:     0 < M <= N-1.
%
%  Output:
%  DM:       DM(1:N,1:N,ell) contains ell-th derivative matrix, ell=1..M.
```

```
%
%  The code implements two strategies for enhanced
%  accuracy suggested by W. Don and S. Solomonoff in
%  SIAM J. Sci. Comp. Vol. 6, pp. 1253--1268 (1994).
%  The two strategies are (a) the use of trigonometric
%  identities to avoid the computation of differences
%  x(k)-x(j) and (b) the use of the "flipping trick"
%  which is necessary since sin t can be computed to high
%  relative precision when t is small whereas sin (pi-t) cannot.
%  Note added May 2003:  It may, in fact, be slightly better not to
%  implement the strategies (a) and (b).   Please consult the following
%  paper for details:   "Spectral Differencing with a Twist", by
%  R. Baltensperger and M.R. Trummer, to appear in SIAM J. Sci. Comp.

%  J.A.C. Weideman, S.C. Reddy 1998.  Help notes modified by
%  JACW, May 2003.

     I = eye(N);                          % Identity matrix.
     L = logical(I);                      % Logical identity matrix.

    n1 = floor(N/2); n2  = ceil(N/2);    % Indices used for flipping trick.

     k = [0:N-1]';                        % Compute theta vector.
    th = k*pi/(N-1);

     x = sin(pi*[N-1:-2:1-N]'/(2*(N-1))); % Compute Chebyshev points.

     T = repmat(th/2,1,N);
    DX = 2*sin(T'+T).*sin(T'-T);          % Trigonometric identity.
    DX = [DX(1:n1,:); -flipud(fliplr(DX(1:n2,:)))];   % Flipping trick.
 DX(L) = ones(N,1);                   % Put 1's on the main diagonal of DX.

     C = toeplitz((-1).^k);               % C is the matrix with
C(1,:) = C(1,:)*2; C(N,:) = C(N,:)*2;     % entries c(k)/c(j)
C(:,1) = C(:,1)/2; C(:,N) = C(:,N)/2;

     Z = 1./DX;                          % Z contains entries 1/(x(k)-x(j))
  Z(L) = zeros(N,1);                       % with zeros on the diagonal.

     D = eye(N);                          % D contains diff. matrices.

for ell = 1:M
         D = ell*Z.*(C.*repmat(diag(D),1,N) - D); % Off-diagonals
       D(L) = -sum(D');                        % Correct main diagonal of D
DM(:,:,ell) = D;                                 % Store current D in DM
end
```

B.2.5 'con_diff_tran.m'

```
function [ymesh,I,O,D1T,D2T,D3T,D4T]=con_diff_tran(N)
% Transformed Chebyshev matrices, collocation points, Identity
% and Zero Matrices generation (with BC applied)
%  The script file takes in the number of points needed (N)
%  and output the transformed (N-2)x(N-2) Chebyshev differentiation
%  matrices, with boundary condition already inbuilt.
%  as well as the 'ymesh', which is the collocation points along real
%  y-axis from -infinity to infinity (N)x(1) vector.

%% -------------- Get Chebyshev differentiation matrices ------------------
[~,DM]=chebdif(N,3); %Get full Cheby. differentiation matrices 4 1st 3 order
[xi,D4] = cheb4c(N); %Get 4th order Chebyshev differentiation, with
                     %the clamped boundary conditions
                     %u(1)=u'(1)=u(-1)=u'(-1)=0 incoperated already
% Note that xi, which is the collocation points from -1 to 1,
% already have first and last rows chopped off (Dirichlet BC).

%% -------------- Applying Boundary Condition --------------------------
% Loosely based on orrosom.m from  S.C. Reddy, J.A.C. Weideman 1998.

% Enforce Dirichlet BCs by chopping off first and last rows and columns
D1=DM(2:N-1,2:N-1,1);
D2=DM(2:N-1,2:N-1,2);
D3=DM(2:N-1,2:N-1,3);

%Generate the right dimenion matrix of Identity matrix and zero matrix
% for easier use later in the script.
I=eye(N-2,N-2);
O=zeros(N-2,N-2);

%Transformation from xi=[-1,1] to ymesh=[-inf,inf]
[ymesh,D1T,D2T,D3T,D4T]=tan_trans(xi,D1,D2,D3,D4);
```

B.2.6 'rho_profile.m'

```
function [rho,Drho]=rho_profile(ymesh,I,rho_slope,theta)
% This function take the ymesh and generate the plane rho profile along y
% direction.

%% Normalized temperature Profile
rho=diag(1-ymesh*sin(theta)*rho_slope); %rho(y)
Drho=-sin(theta)*rho_slope*I; %d rho(y)/dy
```

B.2.7 'tan_trans.m'

```
function [y,D1T,D2T,D3T,D4T]=tan_trans(xi,D1,D2,D3,D4)
% Tangent Transformation function to tranform Chebyshev matrices
% from [-1,1] to [-inf,inf].
% Input: xi: [-1,1] vector of collocation points.
%        D1,D2,D3,D4: Chevyshev differential matrices of order 1,2,3,4.
% Output: y: [-inf,inf] vector of transformed collocation points,
%            Using y=tan( pi/2 *xi);
%        D1T,D2T,D3T,D4T: Transformed Chevyshev differential matrices of
%                         order 1,2,3,4.

%For tan (pi/2) and tan(-pi/2) will give infinity (numerical error).
%This is to give a slightly smaller numerical pi to avoid error.

%  pix=pi*0.99; % for [-62,62]
pix=1.47112767430373*2; % for [-10,10]
%% Transformation dy
y=tan(pix/2.*xi);
dy=2/pix./(1+y.^2); %d(xi)/dy
ddy=-4/pix.*y./((1+y.^2).^2);  %d2(xi)/dy2
dddy=4/pix.*(3.*y.^2-1)./(1+y.^2).^3; %d3(xi)/dy3
ddddy=-48/pix.*y.*(y.^2-1)./(1+y.^2).^4;%d4(xi)/dy4
%% Transformation applied onto Chevyshev differential matrices
D1T=diag(dy)*D1;
D2T=diag(dy.^2)*D2+diag(ddy)*D1;
D3T=diag(dy.^3)*D3+3.*diag(dy.*ddy)*D2+diag(dddy)*D1;
D4T=diag(dy.^4)*D4+6.*diag(ddy.*dy.^2)*D3+...
    diag(3.*ddy.^2+4.*dy.*dddy)*D2+diag(ddddy)*D1;
```

B.2.8 'wake_profile.m'

```
function [U,DU,DDU]=wake_profile(ymesh,R,a)
% This function take the ymesh and generate the wake profile.
% Based on 2D Plane Wake Profile by (Monkewitz 1988)
%% Initization
% global R a;

%% Wake Profile
U=diag(1-R+(2*R)./(1+sinh(abs(ymesh).*asinh(1)).^(2*a)));
DU=diag(-sign(ymesh).*(4*a*R*asinh(1).*cosh(abs(ymesh).*asinh(1))...textcolorcomment
    .*sinh(abs(ymesh).*asinh(1)).^(-1 + 2*a)) ...
    ./(1 + sinh(abs(ymesh).*asinh(1)).^(2*a)).^2);
DDU=diag((4.*a.*R.*asinh(1).^2.*sinh(abs(ymesh).*asinh(1)).^(-2 + 2.*a).* ...
    (1 - a + (1 + a).*sinh(abs(ymesh).*asinh(1)).^(2.*a)...
    + a.*cosh(2.*abs(ymesh).*asinh(1)).* ...
    (-1 + sinh(abs(ymesh).*asinh(1)).^(2.*a)))...
    ./(1 + sinh(abs(ymesh).*asinh(1)).^(2.*a)).^3);
```

B.3 Search

B.3.1 'Meu_alpha_search.m'

```
function [f_out,kr_out]=Meu_alpha_search(settings,k1,k2)
% A wrapper script to search for max omega_i among alpha along Fk (defined
% by ki)
% Utilize Secant Search Method
zi=sqrt(-1);

%% ====== Set up Search Initial Parameters ================================
% k1=0.8-0.4*zi; %First initial Position
```

```
% k2=0.9-0.3*zi; %Second initial Position
omg_ini=0.9672; %Choice of omega to be searched
f1=omg_ini; %Set up

% iter=10; %Number of iteration
%% ------ Iterate in Secant Scheme ----------------------------------------
% Initial point
[fp1,f1]=Meu_domg_dk(settings,k1,f1); %the initial choice of omega f1
% is currently turned off in Meu_domg_dk
[fp2,f2]=Meu_domg_dk(settings,k2,f1);
% Loop through the iteration
while abs(f1-f2)>1e-4
%     k_record(l)=k1; % Recording search trace
    %Find next point with Secant method
    k_next=k2-fp2*(k2-k1)/(fp2-fp1);
    if real(k_next)<0
       k_next=k2/2;
    end
    if real(k_next)>3
       k_next=k2/2;
    end
    %Assign data to next search
    k1=k2;
    fp1=fp2;
    f1=f2;
    k2=k_next;
    % Evaluate point 2
    [fp2,f2]=Meu_domg_dk(settings,k2,f1);
end
% k_record(l+1)=k1;
f_out=imag(f1);
kr_out=k1;
```

B.3.2 'Meu_alpha_search_CU.m'

```
function [f_out,kr_out]=Meu_alpha_search_CU(settings,k1,k2)
% A wrapper script to search for max omega_i among alpha along Fk (defined
% by ki)
% Utilize secant method
zi=sqrt(-1);

%% ====== Set up Search Initial Parameters ================================
% k1=0.8-0.4*zi; %First initial Position
% k2=0.9-0.3*zi; %Second initial Position
omg_ini=0.9672; %Choice of omega to be searched
f1=omg_ini; %Set up

% iter=10; %Number of iteration
%% ------ Iterate in Secant Scheme ----------------------------------------
% Initial point
[fp1,f1]=Meu_domgi_dkr(settings,k1,f1);
[fp2,f2]=Meu_domgi_dkr(settings,k2,f1);
% Loop through the iteration
while abs(f1-f2)>1e-4
%     k_record(l)=k1; % Recording search trace
    %Find next point with Secant method
    k_next=k2-fp2*(k2-k1)/(fp2-fp1);
    if real(k_next)<0.1
       k_next=k_next*2;
    end
    if real(k_next)>3
       k_next=k2/2;
    end
    %Assign data to next search
    k1=k2;
    fp1=fp2;
    f1=f2;
    k2=k_next;
    % Evaluate point 2
    [fp2,f2]=Meu_domgi_dkr(settings,k2,f1);
end
% k_record(l+1)=k1;
f_out=imag(f1);
kr_out=k1;
```

B.3.3 'Meu_cont.m'

```
clfunction omega1=Meu_cont(settings,omega_ini,alpha)
% Wrapper Script to output the most unstable eigenvalue
    omega=Meunier_2DBos_no_w(settings,alpha); %output all eigen values
    omega1=eigen_sort(omega,omega_ini);%output the most unstable eigenvalue
```

B.3.4 'Meu_domg_dk.m'

```
function [domgdk,omg_cen]=Meu_domg_dk(settings,k,omega_ini)
% Evaluate the (d omega)/dk at k for AU
% The average in four direction (1-1i,1+1i,1,i) is taken as the derivatives
% OUTPUT: domgdk = (d omega)/dk @ k
%         omg_cen = omega @ k

%------ Central difference approximation in 1-1i direction ----------------
    epsilon=0.000001-0.000001*sqrt(-1);
    % An arbitarily small delta_k in 1-1i direction

%     omg_cen=Meu_cont(settings,k,omega_ini);
    omg_cen=Meu_sorted(settings,k);
    omg_for=Meu_cont(settings,omg_cen,k+epsilon);
    omg_bak=Meu_cont(settings,omg_cen,k-epsilon);
    domgdk1=cen_diff([k-epsilon k k+epsilon],[omg_bak omg_cen omg_for]);
    %Central difference

%------ Central difference approximation in 1+1i direction ----------------
    epsilon=0.000001+0.000001*sqrt(-1);
    % An arbitarily small delta_k in 1+1i direction

    omg_for=Meu_cont(settings,omg_cen,k+epsilon);
    omg_bak=Meu_cont(settings,omg_cen,k-epsilon);
    domgdk2=cen_diff([k-epsilon k k+epsilon],[omg_bak omg_cen omg_for]);
    %Central difference

%------ Central difference approximation in 1 direction -------------------
    epsilon=0.000001; % An arbitarily small delta_k in 1 direction

    omg_for=Meu_cont(settings,omg_cen,k+epsilon);
    omg_bak=Meu_cont(settings,omg_cen,k-epsilon);
    domgdk3=cen_diff([k-epsilon k k+epsilon],[omg_bak omg_cen omg_for]);
    %Central difference

%------ Central difference approximation in 1i direction ------------------
    epsilon=0.000001*sqrt(-1);
    % An arbitarily small delta_k in i direction

    omg_for=Meu_cont(settings,omg_cen,k+epsilon);
    omg_bak=Meu_cont(settings,omg_cen,k-epsilon);
    domgdk4=cen_diff([k-epsilon k k+epsilon],[omg_bak omg_cen omg_for]);
    %Central difference

%------ Taking an average in all four direction ---------------------------
    domgdk=(domgdk1+domgdk2+domgdk3+domgdk4)/4; %output
end
```

B.3.5 'Meu_domgi_dkr.m'

```
function [domgdk,omg_cen]=Meu_domgi_dkr(settings,k,omega_ini)
% Evaluate the (d omega_i)/dk_r at k for CU
% OUTPUT: domgdk = (d omega_i)/dk_r @ k
%         omg_cen = omega @ k

%------ Central difference approximation in 1-1i direction ----------------
    epsilon=0.000001; % An arbitarily small delta_k in 1 direction
%     omg_cen=Meu_cont(settings,k,omega_ini);
    omg_cen=Meu_sorted(settings,k);
    omg_for=Meu_cont(settings,omg_cen,k+epsilon);
    omg_bak=Meu_cont(settings,omg_cen,k-epsilon);
    %Central difference
    domgdk=cen_diff([k-epsilon k k+epsilon],imag([omg_bak omg_cen omg_for]));

end
```

B.3.6 'Meu_sorted.m'

```
function omega1=Meu_sorted(settings,alpha)
% Wrapper Script to output the most unstable eigenvalue

    omega=Meunier_2DBos_no_w(settings,alpha); %output all eigen values
%     c1=eigen_sort(omega/alpha);
%     omega1=c1*alpha; %output the most unstable eigenvalue (in terms of c)
    omega1=eigen_sort(omega); %output the most unstable eigenvalue
```

B.3.7 'cen_diff.m'

```
function [der,der_x]=cen_diff(x,y)
```

```
% Central Difference Scheme to evaluate the differential dy/dx
% Can accept vector x and y for a series of data
% Input: vector x
%        vector y=f(x)
% Output: vector der=f'(der_x)
%         vector der_x= x with first and last element chopped away.
    ydif=diff(y);
    xdif=diff(x);
    ydif_sum=ydif(2:end)+ydif(1:end-1);
    xdif_sum=xdif(2:end)+xdif(1:end-1);
    der=ydif_sum./xdif_sum;
    der_x=x(2:end-1);
end
```

B.3.8 'eigen_sort.m'

```
function varargout=eigen_sort(omega,omega_ini)
% Sort the eigenvalues from the most unstable ones to the least unstable.
% Output as many eigenvlaues as the output is requesting.
% If input have omega_ini, then the eigenvalue is ranked by the distance
% from omega_ini.

%% ------ Eigenvalue Sorting ------------------------------------------------
    if nargin==1
        omegai=imag(omega); %unsorted imaginary part
        [~,iorder]=sort(omegai,'descend');
    else
        omega_dist=abs(omega-omega_ini);%unsorted imaginary part
        [~,iorder]=sort(omega_dist,'ascend');
    end
    %Generate sort column of eigenvalues
    omega_sorted=omega(iorder);

%% ====== Outputing first few Eigenvalue ==================================
varargout=cell(nargout);
    if length(omega_sorted)<length(varargout)
       for k=1:length(omega_sorted)
            varargout{k}=(omega_sorted(k));
       end
       for k=(length(omega_sorted)+1):length(varargout)
           varargout{k}=NaN;
       end
    else
        for k=1:length(varargout)
            varargout{k}=(omega_sorted(k));
        end
    end
end
```

B.4 Generating Tracks

B.4.1 'hungarianlinker.m' (Jean-Yves Tinevez, 2016)

```
function [ target_indices target_distances unassigned_targets total_cost ]...
    = hungarianlinker(source, target, max_distance)
%HUNGARIANLINKER link two lists of points based on the hungarian algorithm.
%
% target_indices = HUNGARIANLINKER(source, target) finds for each point in
% 'source' the closest point in 'target'. These 2 inputs must be arrays
% with one point per row, and have their cartesian coordinates in each
% column (1D, 2D, 3D, ...). Source to target assignment is based on the
% famous hungarian algorithm using its excellent implementation by the
% excellent Yi Cao. The two arrays might not have the same number of
% points.
%
% The indices of the 'target' points are returned in an array
% 'target_indices', so that each row in 'source' matches the corresponding
% row in 'target(target_indices, :)'.
%
% The linking is exclusive: one source point is linked to at most one
% target point, and conversely. The linking is globally optimal: the sum of
% the square distance is minimized, contrary to the naive nearest neighbor
% approach.
%
% target_indices = HUNGARIANLINKER(source, target, max_distance) adds
% a condition on distance. If the nearest neighbor is found to be at a
% distance larger than the given 'max_distance', they are not linked, and
% the 'target_indices' receive the value -1 for this source point. The same
% happens if all target points are exhausted.
%
```

```
% [ target_indices target_distances ] = HUNGARIANLINKER(source, target)
% additionaly return the distance to the matched target point. Un-matched
% source points have a distance value set to NaN.
%
% [ target_indices target_distances unmatched_targets ] =
%                                       HUNGARIANLINKER(source, target)
% additionaly return the indices of the points in 'target' that have not
% been linked.
%
% [ target_indices target_distances unmatched_targets total_cost ] =
%                                       HUNGARIANLINKER(source, target)
% additionaly return the globally optimized value of the square distance
% sum.
%
% The matching algorithm used here is one of the best available and ensures
% that the resulting assignment is a optimum. However the price to pay is
% an increased complexity. The cost for the naive nearest neighbor approach
% roughly scales as O(p^2) where p is the number of source points. The
% munkres implementation of the hungarian algorithm by Yi Cao is in O(p^3),
% and is the best so far.
%
% EXAMPLE:
%
% n_points = 20;
% source = 10 * rand(n_points, 2);
% target = source + rand(n_points, 2);
% target_indices = hungarianlinker(source, target);
% colors = hsv(n_points);
% figure
% hold on
% for i = 1 :n_points
%    plot(source(i,1), source(i,2), 'o', 'Color', colors(i,:))
%    plot(target(target_indices(i),1), target(target_indices(i),2), 's', ...
%       'Color', colors(i,:))
%    plot( [ source(i,1) target(target_indices(i),1) ] , ...
%       [ source(i,2)  target(target_indices(i),2) ], ...
%        'Color', colors(i,:))
% end
%
%
% Jean-Yves Tinevez <jeanyves.tinevez@gmail.com>.
% However all credits should go to Yi Cao, which did the hard job of
% implementing the Munkres algorithm; this file is merely a wrapper for it.

    if nargin < 3
        max_distance = Inf;
    end

    n_source_points = size(source, 1);
    n_target_points = size(target, 1);

    D = NaN(n_source_points, n_target_points);

    % Build distance matrix
    for i = 1 : n_source_points

        % Pick one source point
        current_point = source(i, :);

        % Compute square distance to all target points
        diff_coords = target - repmat(current_point, n_target_points, 1);
        square_dist = sum(diff_coords.^2, 2);

        % Store them
        D(i, :) = square_dist;

    end

    % Deal with maximal linking distance: we simply mark these links as already
    % treated, so that they can never generate a link.
    D ( D > max_distance * max_distance ) = Inf;

    % Find the optimal assignment is simple as calling Yi Cao excellent FEX
    % submission.
    [ target_indices total_cost ] = munkres(D);
    % Set unmatched sources to -1
    target_indices ( target_indices  == 0 ) = -1;

    % Collect distances
    target_distances = NaN(numel(target_indices), 1);
    for i = 1 : numel(target_indices)
        if target_indices(i) < 0
            continue
        end

        target_distances(i) = sqrt ( D ( i , target_indices(i)) );
```

```
    end

    unassigned_targets = setdiff ( 1 : n_target_points , target_indices );

end
```

B.4.2 'munkres.m' (Jean-Yves Tinevez, 2016)

```
function [assignment,cost] = munkres(costMat)
% MUNKRES   Munkres (Hungarian) Algorithm for Linear Assignment Problem.
%
% [ASSIGN,COST] = munkres(COSTMAT) returns the optimal column indices,
% ASSIGN assigned to each row and the minimum COST based on the assignment
% problem represented by the COSTMAT, where the (i,j)th element represents the cost to assign the jth
% job to the ith worker.
%
% Partial assignment: This code can identify a partial assignment is a full
% assignment is not feasible. For a partial assignment, there are some
% zero elements in the returning assignment vector, which indicate
% un-assigned tasks. The cost returned only contains the cost of partially
% assigned tasks.

% This is vectorized implementation of the algorithm. It is the fastest
% among all Matlab implementations of the algorithm.

% Examples
% Example 1: a 5 x 5 example
%{
[assignment,cost] = munkres(magic(5));
disp(assignment); % 3 2 1 5 4
disp(cost); %15
%}
% Example 2: 400 x 400 random data
%{
n=400;
A=rand(n);
tic
[a,b]=munkres(A);
toc                 % about 2 seconds
%}
% Example 3: rectangular assignment with inf costs
%{
A=rand(10,7);
A(A>0.7)=Inf;
[a,b]=munkres(A);
%}
% Example 4: an example of partial assignment
%{
A = [1 3 Inf; Inf Inf 5; Inf Inf 0.5];
[a,b]=munkres(A)
%}
% a = [1 0 3]
% b = 1.5
% Reference:
% "Munkres' Assignment Algorithm, Modified for Rectangular Matrices",
% http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html

% version 2.3 by Yi Cao at Cranfield University on 11th September 2011

assignment = zeros(1,size(costMat,1));
cost = 0;

validMat = costMat == costMat & costMat < Inf;
bigM = 10^(ceil(log10(sum(costMat(validMat))))+1);
costMat(~validMat) = bigM;

% costMat(costMat~=costMat)=Inf;
% validMat = costMat<Inf;
validCol = any(validMat,1);
validRow = any(validMat,2);

nRows = sum(validRow);
nCols = sum(validCol);
n = max(nRows,nCols);
if ~n
    return
end

maxv=10*max(costMat(validMat));

dMat = zeros(n) + maxv;
dMat(1:nRows,1:nCols) = costMat(validRow,validCol);

%*************************************************
```

```
% Munkres' Assignment Algorithm starts here
%************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   STEP 1: Subtract the row minimum from each row.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
minR = min(dMat,[],2);
minC = min(bsxfun(@minus, dMat, minR));

%************************************************************************
%   STEP 2: Find a zero of dMat. If there are no starred zeros in its
%           column or row start the zero. Repeat for each zero
%************************************************************************
zP = dMat == bsxfun(@plus, minC, minR);

starZ = zeros(n,1);
while any(zP(:))
    [r,c]=find(zP,1);
    starZ(r)=c;
    zP(r,:)=false;
    zP(:,c)=false;
end

while 1
%************************************************************************
%   STEP 3: Cover each column with a starred zero. If all the columns are
%           covered then the matching is maximum
%************************************************************************
    if all(starZ>0)
        break
    end
    coverColumn = false(1,n);
    coverColumn(starZ(starZ>0))=true;
    coverRow = false(n,1);
    primeZ = zeros(n,1);
    [rIdx, cIdx] = find(dMat(~coverRow,~coverColumn)==bsxfun(@plus,minR(~coverRow),minC(~coverColumn)));
    while 1
        %************************************************************************
        %   STEP 4: Find a noncovered zero and prime it.  If there is no starred
        %           zero in the row containing this primed zero, Go to Step 5.
        %           Otherwise, cover this row and uncover the column containing
        %           the starred zero. Continue in this manner until there are no
        %           uncovered zeros left. Save the smallest uncovered value and
        %           Go to Step 6.
        %************************************************************************
        cR = find(~coverRow);
        cC = find(~coverColumn);
        rIdx = cR(rIdx);
        cIdx = cC(cIdx);
        Step = 6;
        while ~isempty(cIdx)
            uZr = rIdx(1);
            uZc = cIdx(1);
            primeZ(uZr) = uZc;
            stz = starZ(uZr);
            if ~stz
                Step = 5;
                break;
            end
            coverRow(uZr) = true;
            coverColumn(stz) = false;
            z = rIdx==uZr;
            rIdx(z) = [];
            cIdx(z) = [];
            cR = find(~coverRow);
            z = dMat(~coverRow,stz) == minR(~coverRow) + minC(stz);
            rIdx = [rIdx(:);cR(z)];
            cIdx = [cIdx(:);stz(ones(sum(z),1))];
        end
        if Step == 6
            % ************************************************************************
            % STEP 6: Add the minimum uncovered value to every element of each covered
            %         row, and subtract it from every element of each uncovered column.
            %         Return to Step 4 without altering any stars, primes, or covered lines.
            %************************************************************************
            [minval,rIdx,cIdx]=outerplus(dMat(~coverRow,~coverColumn),minR(~coverRow),minC(~coverColumn));
            minC(~coverColumn) = minC(~coverColumn) + minval;
            minR(coverRow) = minR(coverRow) - minval;
        else
            break
        end
    end
    %************************************************************************
    % STEP 5:
    %  Construct a series of alternating primed and starred zeros as
    %  follows:
    %  Let Z0 represent the uncovered primed zero found in Step 4.
```

```
    %  Let Z1 denote the starred zero in the column of Z0 (if any).
    %  Let Z2 denote the primed zero in the row of Z1 (there will always
    %  be one).  Continue until the series terminates at a primed zero
    %  that has no starred zero in its column.  Unstar each starred
    %  zero of the series, star each primed zero of the series, erase
    %  all primes and uncover every line in the matrix.  Return to Step 3.
    %*************************************************************************
    rowZ1 = find(starZ==uZc);
    starZ(uZr)=uZc;
    while rowZ1>0
        starZ(rowZ1)=0;
        uZc = primeZ(rowZ1);
        uZr = rowZ1;
        rowZ1 = find(starZ==uZc);
        starZ(uZr)=uZc;
    end
end

% Cost of assignment
rowIdx = find(validRow);
colIdx = find(validCol);
starZ = starZ(1:nRows);
vIdx = starZ <= nCols;
assignment(rowIdx(vIdx)) = colIdx(starZ(vIdx));
pass = assignment(assignment>0);
pass(~diag(validMat(assignment>0,pass))) = 0;
assignment(assignment>0) = pass;
cost = trace(costMat(assignment>0,assignment(assignment>0)));

function [minval,rIdx,cIdx]=outerplus(M,x,y)
ny=size(M,2);
minval=inf;
for c=1:ny
    M(:,c)=M(:,c)-(x+y(c));
    minval = min(minval,min(M(:,c)));
end
[rIdx,cIdx]=find(M==minval);
```

B.4.3 'nearestneighborlinker.m' (Jean-Yves Tinevez, 2016)

```
function [ target_indices target_distances unassigned_targets ] = ...
    nearestneighborlinker(source, target, max_distance)
%NEARESTNEIGHBORLINKER link two lists of points based on nearest neighbor.
%
% target_indices = NEARESTNEIGHBORLINKER(source, target) finds for each
% point in 'source' the closest point in 'target'. These 2 inputs must be
% arrays with one point per row, and have their cartesian coordinates in
% each column (1D, 2D, 3D, ...). Nearest neighbor matching is based on
% euclidean distance. The two arrays might not have the same number of
% points.
%
% The indices of the 'target' points are returned in an array
% 'target_indices', so that each row in 'source' matches the corresponding
% row in 'target(target_indices, :)'.
%
% The linking is exclusive: one source point is linked to at most one
% target point, and conversely. The linking is only locally optimal: the
% two closest points amongst the two sets are sought for first, then the
% second closest pair, excluding the first, etc... This ensures that the
% resulting linking will not depend on the order of the points in each set.
%
% target_indices = NEARESTNEIGHBORLINKER(source, target, max_distance) adds
% a condition on distance. If the nearest neighbor is found to be at a
% distance larger than the given 'max_distance', they are not linked, and
% the 'target_indices' receive the value -1 for this source point. The same
% happens if all target points are exhausted.
%
% [ target_indices target_distances ] =
%                                 NEARESTNEIGHBORLINKER(source, target)
% additionaly return the distance to the matched target point. Un-matched
% source points have a distance value set to NaN.
%
% [ target_indices target_distances unmatched_targets ]=
%                                 NEARESTNEIGHBORLINKER(source, target)
% additionaly return the indices of the points in 'target' that have not
% been linked.
%
% This is the cheapest (in term of accuracy) algorithm for linking that can
% be made. In particular, it is not guaranteed (and it is generally not the
% case) that the returned linking is an optimum for the sum of distances.
% Each source point is matched regardless of the others, there is no global
% optimization here (the Hungarian algorithm does that). Also, there exists
% refinement to nearest neighbor searches, such as the use of KD-trees;
% this contribution is exempt of such developments.
%
```

```
% EXAMPLE:
%
% n_points = 20;
% source = 10 * rand(n_points, 2);
% target = source + rand(n_points, 2);
% target_indices = nearestneighborlinker(source, target);
% colors = hsv(n_points);
% figure
% hold on
% for i = 1 :n_points
%    plot(source(i,1), source(i,2), 'o', 'Color', colors(i,:))
%    plot(target(target_indices(i),1), target(target_indices(i),2), 's', ...
%       'Color', colors(i,:))
%    plot( [ source(i,1) target(target_indices(i),1) ] , ...
%       [ source(i,2)  target(target_indices(i),2) ], ...
%        'Color', colors(i,:))
% end
%
% VERSION HISTORY
%
% * v1.0 - November 2011 - Initial release.
% * v1.1 - May 2012 - Fix a severe bug thanks to Dave Cade
%
% Jean-Yves Tinevez < jeanyves.tinevez@gmail.com> November 2011 - 2012

    if nargin < 3
        max_distance = Inf;
    end

    n_source_points = size(source, 1);
    n_target_points = size(target, 1);

    D = NaN(n_source_points, n_target_points);

    % Build distance matrix
    for i = 1 : n_source_points

        % Pick one source point
        current_point = source(i, :);

        % Compute square distance to all target points
        diff_coords = target - repmat(current_point, n_target_points, 1);
        square_dist = sum(diff_coords.^2, 2);

        % Store them
        D(i, :) = square_dist;

    end

    % Deal with maximal linking distance: we simply mark these links as already
    % treated, so that they can never generate a link.
    D ( D > max_distance * max_distance ) = Inf;

    target_indices = -1 * ones(n_source_points, 1);
    target_distances = NaN(n_source_points, 1);

    % Parse distance matrix
    while ~all(isinf(D(:)))
        % index of the closest target for each source points
        [ min_D closest_targets ] = min(D, [], 2);
        [ ~, sorted_index ] = sort(min_D);

        for i = 1 : numel(sorted_index)

            source_index =  sorted_index(i);
            target_index =  closest_targets ( sorted_index(i) );

            % Did we already assigned this target to a source?
            if any ( target_index == target_indices )

                % Yes, then exit the loop and change the distance matrix to
                % prevent this assignment
                break

            else

                % No, then store this assignment
                target_indices( source_index ) = target_index;
                target_distances ( source_index ) = ...
                    sqrt ( min_D (  sorted_index(i) ) );

            % And make it impossible to find it again by putting the target
                % point to infinity in the distance matrix
                D(:, target_index) = Inf;
                % And the same for the source line
                D(source_index, :) = Inf;
```

```
                if all(isinf(D(:)))
                    break
                end

            end

        end

    end

    unassigned_targets = setdiff ( 1 : n_target_points , target_indices );

end
```

B.4.4 'simpletracker.m' (Jean-Yves Tinevez, 2016)

```
function [ tracks adjacency_tracks A ] = simpletracker(points, varargin)
% SIMPLETRACKER  a simple particle tracking algorithm that can deal with gaps
%
% *Tracking* , or particle linking, consist in re-building the trajectories
% of one or several particles as they move along time. Their position is
% reported at each frame, but their identiy is yet unknown: we do not know
% what particle in one frame corresponding to a particle in the previous
% frame. Tracking algorithms aim at providing a solution for this problem.
%
% |simpletracker.m| is - as the name says - a simple implementation of a
% tracking algorithm, that can deal with gaps. A gap happens when one
% particle that was detected in one frame is not detected in the subsequent
% one. If not dealt with, this generates a track break, or a gap, in the
% frame where the particule disappear, and a false new track in the frame
% where it re-appear.
%
% |simpletracker| first do a frame-to-frame linking step, where links are
% first created between each frame pair, using by default the hungarian
% algorithm of |hungarianlinker|. Links are created amongst particle paris
% found to be the closest (euclidean distance). By virtue of the hungarian
% algorithm, it is ensured that the sum of the pair distances is minimized
% over all particules between two frames.
%
% Then a second iteration is done through the data, investigating track
% ends. If a track beginning is found close to a track end in a subsequent
% track, a link spanning multiple frame can be created, bridging the gap
% and restoring the track. The gap-closing step uses the nearest neighbor
% algorithm provided by |nearestneighborlinker|.
%
% INPUT SYNTAX
%
% tracks = SIMPLETRACKER(points) rebuilds the tracks generated by the
% particle whose coordinates are in |points|. |points| must be a cell
% array, with one cell per frame considered. Each cell then contains the
% coordinates of the particles found in that frame in the shape of a
% |n_points x n_dim| double array, where |n_points| is the number of points
% in that frame (that can vary a lot from one frame to another) and |n_dim|
% is the dimensionality of the problem (1 for 1D, 2 for 2D, 3 for 3D,
% etc...).
%
% tracks = SIMPLETRACKER(points, KEY, VALUE, ...)  allows to pass extra
% parameters to configure the tracking process settings. Accepted KEYS &
% VALUES are:
%
% 'Method' - a string, by default 'Hungarian'
% Specifies the method to use for frame-to-frame particle linking. By
% default, the hungarian method is used, which ensures that a global
% optimum is found for each frame pair. The complexity of this algorithm is
% in O(n^3), which can be prohibitive for problems with a large number of
% point in each frame (e.g. more than 1000). Therefore, it is possible to
% use the nearest-neighbor algorithm by setting the method to
% 'NearestNeighbor', which only achieves a local optimum for a pair of
% points, but runs in O(n^2).
%
% 'MaxLinkingDistance' - a positive number, by default Inifity.
% Defines a maximal distance for particle linking. Two particles will not
% be linked (even if they are the remaining closest pair) if their distance
% is larger than this value. By default, it is infinite, not preventing nay
% linking.
%
% 'MaxGapClosing' - a positive integer, by default 3
% Defines a maximal frame distance in gap-closing. Frames further way than
% this value will not be investigated for gap closing. By default, it has
% the value of 3.
%
% 'Debug' - boolean flag, false by default
% Adds some printed information about the tracking process if set to true.
%
% OUTPUT SYNTAX
```

```
%
% track = SIMPLETRACKER(...) return a cell array, with one cell per found
% track. Each track is made of a |n_frames x 1| integer array, containing
% the index of the particle belonging to that track in the corresponding
% frame. NaN values report that for this track at this frame, a particle
% could not be found (gap).
%
% Example output: |track{1} = [ 1 2 1 NaN 4 ]| means that the first track
% is made of the particle 1 in the first frame, the particule 2 in the
% second frame, the particle 1 in the 3rd frame, no particle in the 4th
% frame, and the 4th particle in the 5th frame.
%
% [ tracks adjacency_tracks ] = SIMPLETRACKER(...) return also a cell array
% with one cell per track, but the indices in each track are the global
% indices of the concatenated points array, that can be obtained by
% |all_points = vertcat( points{:} );|. It is very useful for plotting
% applications.
%
% [ tracks adjacency_tracks A ] = SIMPLETRACKER(...) return the sparse
% adjacency matrix. This matrix is made everywhere of 0s, expect for links
% between a source particle (row) and a target particle (column) where
% there is a 1. Rows and columns indices are for points in the concatenated
% points array. Only forward links are reported (from a frame to a frame
% later), so this matrix has no non-zero elements in the bottom left
% diagonal half. Reconstructing a crude trajectory using this matrix can be
% as simple as calling |gplot( A, vertcat( points{:} ) )|
%
% VERSION HISTORY
%
% * v1.0 - November 2011 - Initial release.
% * v1.1 - May 2012 - Solve memory problems for large number of points.
%                   - Considerable speed improvement using properly the
%                   sparse matrices.
%                   - Use the key/value pair syntax to configure the
%                   function.
% * v1.3 - August 2012 - Fix a severe bug thanks to Dave Cade
% * v1.4 - May 2017 - Modified for parallelization
% Jean-Yves Tinevez < jeanyves.tinevez@gmail.com> November 2011 - 2012
% Editted by Lloyd Fung for IC Aero FYP 2017. May 2017.

    %% Parse arguments

    p = inputParser;
    defaultDebug                = false;
    defaultMaxGapClosing        = 3;
    defaultMaxLinkingDistance   = Inf;
    defaultMethod               = 'Hungarian';
    expectedMethods = { defaultMethod, 'NearestNeighbor' };

    p.addParamValue('Debug', defaultDebug, @islogical);
    p.addParamValue('MaxGapClosing', defaultMaxGapClosing, @isnumeric);
    p.addParamValue('MaxLinkingDistance', ...
        defaultMaxLinkingDistance, @isnumeric);
    p.addParamValue('Method', defaultMethod,...
         @(x) any(validatestring(x, expectedMethods)));

    p.parse( varargin{:} );

    debug                   = p.Results.Debug;
    max_gap_closing         = p.Results.MaxGapClosing;
    max_linking_distance    = p.Results.MaxLinkingDistance;
    method                  = p.Results.Method;

    %% Frame to frame linking

    if debug
       fprintf('Frame to frame linking using %s method.\n', method);
    end

    n_slices = numel(points);

%     current_slice_index = 0;
    row_indices = cell(n_slices, 1);
    column_indices = cell(n_slices, 1);
    unmatched_targets = cell(n_slices, 1);
    unmatched_sources = cell(n_slices, 1);
    n_cells = cellfun(@(x) size(x, 1), points);

    if debug
       fprintf('%03d/%03d', 0, n_slices-1);
    end

    parfor i = 1 : n_slices-1
        current_slice_index=sum(n_cells(1:(i-1)));
        if debug
            fprintf(repmat('\b', 1, 7));
            fprintf('%03d/%03d', i, n_slices-1);
```

```
    end

    source = points{i};
    target = points{i+1};

    % Frame to frame linking
    if  lower(method)=='hungarian'
            [target_indices , ~, unmatched_targets_temp ] = ...
                hungarianlinker(source, target, max_linking_distance);
    else
            [target_indices , ~, unmatched_targets_temp ] = ...
                nearestneighborlinker(source, ...
                target, max_linking_distance);
    end
    unmatched_targets{i+1}=unmatched_targets_temp;

    unmatched_sources{i} = find( target_indices == -1 );

    % Prepare holders for links in the sparse matrix
    n_links = sum( target_indices ~= -1 );
    row_indices_temp = NaN(n_links, 1);
    column_indices_temp = NaN(n_links, 1);

    % Put it in the adjacency matrix
    index = 1;
    for j = 1 : numel(target_indices)

        % If we did not find a proper target to link, we skip
        if target_indices(j) == -1
            continue
        end

        % The source line number in the adjacency matrix
        row_indices_temp(index) = current_slice_index + j;

        % The target column number in the adjacency matrix
        column_indices_temp(index) = current_slice_index +...
            n_cells(i) + target_indices(j);

        index = index + 1;

    end
    row_indices{i}=row_indices_temp;
    column_indices{i}=column_indices_temp;

end



row_index = vertcat(row_indices{:});
column_index = vertcat(column_indices{:});
link_flag = ones( numel(row_index), 1);
n_total_cells = sum(n_cells);

if debug
    fprintf('\nCreating %d links over a total of %d points.\n', ...
        numel(link_flag), n_total_cells)
end

A = sparse(row_index, column_index, link_flag, n_total_cells,...
    n_total_cells);

if debug
    fprintf('Done.\n')
end


%% Gap closing

if debug
    fprintf('Gap-closing:\n')
end

current_slice_index = 0;
for i = 1 : n_slices-2


    % Try to find a target in the frames following, starting at i+2, and
    % parsing over the target that are not part in a link already.

    current_target_slice_index = current_slice_index + n_cells(i) ...
        + n_cells(i+1);

    for j = i + 2 : min(i +  max_gap_closing, n_slices)

        source = points{i}(unmatched_sources{i}, :);
        target = points{j}(unmatched_targets{j}, :);
```

```
        if isempty(source) || isempty(target)
            current_target_slice_index = current_target_slice_index ...
                + n_cells(j);
            continue
        end

        target_indices = nearestneighborlinker(source, ...
            target, max_linking_distance);

        % Put it in the adjacency matrix
        for k = 1 : numel(target_indices)

            % If we did not find a proper target to link, we skip
            if target_indices(k) == -1
                continue
            end

            if debug
                fprintf('Creating a link between point %d of frame %d and point %d of frame %d.\n', ...
                    unmatched_sources{i}(k), i, ...
                    unmatched_targets{j}(target_indices(k)), j);
            end

            % The source line number in the adjacency matrix
            row_index = current_slice_index + unmatched_sources{i}(k);
            % The target column number in the adjacency matrix
            column_index = current_target_slice_index ...
                + unmatched_targets{j}(target_indices(k));

            A(row_index, column_index) = 1; %#ok<SPRIX>

        end

        new_links_target =  target_indices ~= -1 ;

        % Make linked sources unavailable for further linking
        unmatched_sources{i}( new_links_target ) = [];

        % Make linked targets unavailable for further linking
        unmatched_targets{j}(target_indices(new_links_target)) = [];

        current_target_slice_index = ...
            current_target_slice_index + n_cells(j);
    end

    current_slice_index = current_slice_index + n_cells(i);

end

if debug
    fprintf('Done.\n')
end

%% Parse adjacency matrix to build tracks

if debug
    fprintf('Building tracks:\n')
end

% Find columns full of 0s -> means this cell has no source
cells_without_source = [];
for i = 1 : size(A, 2)
    if length(find(A(:,i))) == 0 %#ok<ISMT>
        cells_without_source = [ cells_without_source ; i ]; %#ok<AGROW>
    end
end

n_tracks = numel(cells_without_source);
adjacency_tracks = cell(n_tracks, 1);

AT = A';

for i = 1 : n_tracks

    tmp_holder = NaN(n_total_cells, 1);

    target = cells_without_source(i);
    index = 1;
    while ~isempty(target)
        tmp_holder(index) = target;
        target = find( AT(:, target), 1, 'first' );
        index = index + 1;
    end

    adjacency_tracks{i} = tmp_holder ( ~isnan(tmp_holder) );
end
```

```
    %% Reparse adjacency track index to have it right.
    % The trouble with the previous track index is that the index in each
    % track refers to the index in the adjacency matrix, not the point in
    % the original array. We have to reparse it to put it right.

    tracks = cell(n_tracks, 1);

    for i = 1 : n_tracks

        adjacency_track = adjacency_tracks{i};
        track = NaN(n_slices, 1);

        for j = 1 : numel(adjacency_track)

            cell_index = adjacency_track(j);

            % We must determine the frame this index belong to
            tmp = cell_index;
            frame_index = 1;
            while tmp > 0
                tmp = tmp - n_cells(frame_index);
                frame_index = frame_index + 1;
            end
            frame_index = frame_index - 1;
            in_frame_cell_index = tmp + n_cells(frame_index);

            track(frame_index) = in_frame_cell_index;

        end

        tracks{i} = track;

    end

end
```

B.5 Database Management and Track Visualisation

B.5.1 'database_compress.m'

```
% This script eliminate repeated runs from the database file 'database.mat'
db_name='database.mat';

load(db_name,'settings','tracks','points','alpha_loop',...
    'adjacency_tracks','current_index');
%% ------ Sorting ------
theta_array=[settings(:).theta]';
F_array=[settings(:).F]';
Re_array=[settings(:).Re]';
alpha_i_array=[settings(:).alpha_i]';
% beta_to_alpha_array=[settings(:).beta_to_alpha]';
N_array=[settings(:).N]';

% A=[theta_array,F_array,Re_array,alpha_i_array,beta_to_alpha_array,N_array];
A=[theta_array,F_array,Re_array,alpha_i_array,N_array];
[B,sort_order]=sortrows(A,'descend');

settings=settings(sort_order);
points=points(sort_order);
tracks=tracks(sort_order);
adjacency_tracks=adjacency_tracks(sort_order);
alpha_loop=alpha_loop(sort_order);

[uniqueB iB]=unique([B(:,1:4) [settings.R]' [settings.a]' ...
    [settings.Sc]'  [settings.L_Meunier]'],'rows');

settings=settings(iB);
points=points(iB);
tracks=tracks(iB);
adjacency_tracks=adjacency_tracks(iB);
alpha_loop=alpha_loop(iB);
current_index=numel(alpha_loop);

save(db_name,'settings','tracks','points','alpha_loop',...
    'adjacency_tracks','current_index');
```

B.5.2 'database_merge.m'

```
% This script merge two database.mat.
```

```
clear all
load database.mat;
i1=current_index+1;
load database_90.mat;
ilast=i1+current_index-1;
points2=points;
alpha_loop2=alpha_loop;
tracks2=tracks;
adjacency_tracks2=adjacency_tracks;
settings2=settings;


load database.mat;
points(i1:ilast)=points2;
alpha_loop(i1:ilast)=alpha_loop2;
tracks(i1:ilast)=tracks2;
adjacency_tracks(i1:ilast)=adjacency_tracks2;
settings(i1:ilast)=settings2;
current_index=ilast;
save('database_merged.mat','settings','tracks',...
    'points','alpha_loop','adjacency_tracks','current_index');
```

B.5.3 'database_save.m'

```
function database_save(points_p,tracks_p,adjacency_tracks_p,...
    alpha_loop_p,settings_p,settings_others)
% This script is called whenever a 'track' script is run and requires the
% data to be saved into a database file ('database.mat').
if exist('database.mat', 'file') == 2
    load 'database.mat';
    i=current_index+1;
else
    i=1;
end
    points{i}=points_p;
    alpha_loop{i}=alpha_loop_p;
    tracks{i}=tracks_p;
    adjacency_tracks{i}=adjacency_tracks_p;
    dbsettings.theta=settings_p{1};
    dbsettings.F=settings_p{3};
    dbsettings.Re=settings_p{2};
    dbsettings.Sc=settings_p{4};
    dbsettings.N=settings_others{1};
    dbsettings.ydom=max(settings_p{5});
    dbsettings.alpha_i=imag(alpha_loop_p(1));
    dbsettings.R=settings_others{2};
    dbsettings.a=settings_others{3};
    dbsettings.L_Meunier=settings_others{4};
%     dbsettings.beta_to_alpha=beta_to_alpha;
    settings(i)=dbsettings;
    current_index=i;
    save('database.mat','settings','tracks','points',...
        'alpha_loop','adjacency_tracks','current_index');
end
```

B.5.4 'database_visual.m'

```
function database_visual(db_name)
% This script is called to visualize the data in the database.mat file.
close all;
addpath(genpath(pwd));
load(db_name,'settings','tracks','points',...
    'alpha_loop','adjacency_tracks','current_index');
%% ------ Sorting ------
theta_array=[settings(:).theta]';
F_array=[settings(:).F]';
Re_array=[settings(:).Re]';
alpha_i_array=[settings(:).alpha_i]';
% beta_to_alpha_array=[settings(:).beta_to_alpha]';
N_array=[settings(:).N]';

% A=[theta_array,F_array,Re_array,alpha_i_array,beta_to_alpha_array,N_array];

A=[theta_array,F_array,Re_array,alpha_i_array,N_array];
if strcmp(version('-release'),'2017a')
    [B,sort_order]=sortrows(A,'descend');
else
    [B,sort_order]=sortrows(A);
end
settings=settings(sort_order);
points=points(sort_order);
```

```
tracks=tracks(sort_order);
adjacency_tracks=adjacency_tracks(sort_order);
alpha_loop=alpha_loop(sort_order);
%% ------ PLotting ------
omg_track_slider_db(points,tracks,adjacency_tracks,...
    alpha_loop,settings,current_index);
%save(db_name,'settings','tracks','points','alpha_loop',...
% 'adjacency_tracks','current_index');
end
```

B.5.5 'omg_track_slider.m'

```
function h1=omg_track_slider(points,tracks,adjacency_tracks,...
    alpha_loop,Video_name,title_name,h1)
% This is subroutines for visualizing data generated from a 'track' script.
    %% ------ Set up Figure ----------------------------------------
    if nargin<7
        h1=figure;
    end
        set(h1,'Position',[0 0 1200 900]);

        hold on;
        xlabel('\omega_r');
        ylabel('\omega_i');
        axis([-1  8 -1 0.4]);
        %   axis([-9 14 -2 1]);
        %   ylim([-6 1.5]);

    %% ------ Plot all Tracks ----------------------------------
    disp(['Plotting Tracks...']);

    n_tracks = numel(tracks);
    all_points = vertcat(points{:});

    unstable_track=zeros(1,n_tracks);
    k=1;

    for i_track = 1 : n_tracks

        track = adjacency_tracks{i_track};
        track_points = all_points(track, :);

    %     Detect if track cross x-axis and mark it red, and Plot track.
        if any(track_points(:,2)>0)
            unstable_track(k)=i_track;
            k=k+1;
        else
         plot(track_points(:,1), track_points(:, 2),'Color',[0.8,0.8,0.8]);
        end

    end


    %Plot red track at last to ensure it floats above
    for i=1:(k-1)
        i_track=unstable_track(i);
        track = adjacency_tracks{i_track};
        track_points = all_points(track, :);
        plot(track_points(:,1), track_points(:, 2),'Color','r');
    end

    %% ------ Choosing points to highlight -------------------------------
    %Making First Plot
    i=1;
        %Normal Points
    p_plot=plot(points{i}(:,1),points{i}(:,2),'bx');
        %Most unstable point
    [~,max_pos]=max(points{i}(:,2));
    max_plot=plot(points{i}(max_pos,1),points{i}(max_pos,2),'ro');

        %Unstable point (omega_i>0)
    unstable_points=points{i}(:,2)>0;
    if any(unstable_points)
        unstable_plot=plot(points{i}(unstable_points,1),...
            points{i}(unstable_points,2),'y+');
    else
        unstable_plot=plot([0],[1E20],'y+');
    end

    % Controls of Plot
    title([title_name ' \alpha=' num2str(alpha_loop(i))]);
    hold off;


    %% ------ Setting up Slider ----------------------------
```

```
    pos=get(gca,'position');
    Newpos=[pos(1) pos(2)-0.1 pos(3) 0.05];
    sld = uicontrol('Style', 'slider',...
            'Min',1,'Max',length(alpha_loop),'Value',1,...
            'units','normalized','Position', Newpos,...
            'Callback', @(source,event)plot_update(source));

    %% ------ CallBack function for slider ---------------------------
    function plot_update(source)
        i=floor(source.Value);
        figure(h1);
        title([title_name ' \alpha=' num2str(alpha_loop(i))]);
        set(p_plot,'XData',points{i}(:,1),'YData',points{i}(:,2));
        [~,max_pos]=max(points{i}(:,2));
        set(max_plot,'XData',points{i}(max_pos,1),...
            'YData',points{i}(max_pos,2));
        unstable_points=points{i}(:,2)>0;
        if any(unstable_points)
             set(unstable_plot,'XData',points{i}(unstable_points,1),...
                 'YData',points{i}(unstable_points,2));
        else
             set(unstable_plot,'XData',[0],'YData',[1E20]);
        end
    end
end
```

B.5.6 'omg_track_slider_db.m'

```
function omg_track_slider_db(points_db,tracks_db,adjacency_tracks_db,...
    alpha_loop_db,settings,current_index)
% This is subroutines for visualizing data generated from the database.mat.
    %% ------ Set up Figure -------------------------------------------------

        h1=figure;
        alpha_loop=alpha_loop_db{1};
        set(h1,'Position',[0 0 1200 900]);
        sld = uicontrol('Style', 'slider',...
            'Min',1,'Max',length(alpha_loop),'Value',1,...
            'Callback', @(source,event)slider_update(source));
        plot_out(1);
    %% ------ Setting up Slider ---------------------------
    pos=get(gca,'position');
    Newpos=[pos(1) pos(2)-0.1 pos(3) 0.05];
    sld.Units='normalized';
    sld.Position=Newpos;
   %% ------ Setting up pop-up menu ----------------------------
    title_list=cell(1,current_index);
    for ii=1:current_index
        title_list{ii}=['F=' num2str([settings(ii).F]','%5.3f') ' Re='...
            num2str([settings(ii).Re]','%06.1f')...
            ' theta=' num2str([rad2deg(settings(ii).theta)]','%2.1f') ...
            ' N=' num2str([settings(ii).N]','%03d') ...
            ' R=' num2str([settings(ii).R]','%06.3f') ...
            ' a=' num2str([settings(ii).a]','%05.3f') ...
            ' alfa_i=' num2str([settings(ii).alpha_i]','%5.3f')];
    end
    Newpos=[pos(1) pos(2)+pos(4) pos(3) 0.05];
    popup = uicontrol('Style', 'popup',...
               'String', title_list,...
               'units','normalized', ...
               'Position', Newpos,...
               'Callback', @(source,event)pop_up_update(source));
    %% ------ CallBack function for slider ---------------------------
    function slider_update(source)
        global p_plot unstable_plot max_plot points title_name;
        i=floor(source.Value);
        figure(h1);
        title([title_name ' \alpha=' num2str(alpha_loop(i))]);
        set(p_plot,'XData',points{i}(:,1),'YData',points{i}(:,2));
        [~,max_pos]=max(points{i}(:,2));
        set(max_plot,'XData',points{i}(max_pos,1),...
            'YData',points{i}(max_pos,2));
        unstable_points=points{i}(:,2)>0;
        if any(unstable_points)
             set(unstable_plot,'XData',points{i}(unstable_points,1),...
                 'YData',points{i}(unstable_points,2));
        else
             set(unstable_plot,'XData',[0],'YData',[1E20]);
        end
    end
    %% ------ CallBack function for pop up menu ----------------------------
    function pop_up_update(source)
        i=floor(source.Value);
        plot_out(i);
    end
```

```
    %% ------ Plot processing ---------------------------
    function plot_out(i)
        global p_plot unstable_plot max_plot points title_name;

         %% ------ Parsing data ----------------------
        points=points_db{i};
        tracks=tracks_db{i};
        adjacency_tracks=adjacency_tracks_db{i};
        alpha_loop=alpha_loop_db{i};
        F=settings(i).F;
        Re=settings(i).Re;
        theta=settings(i).theta;
        %% ------ Plot all Tracks -----------------------------------

        n_tracks = numel(tracks);
        all_points = vertcat(points{:});

        unstable_track=zeros(1,n_tracks);
        k=1;

        for i_track = 1 : n_tracks

            track = adjacency_tracks{i_track};
            track_points = all_points(track, :);

        %     Detect if track cross x-axis and mark it red, and Plot track.
            if any(track_points(:,2)>0)
                unstable_track(k)=i_track;
                k=k+1;
            else
                plot(track_points(:,1), track_points(:, 2),...
                    'Color',[0.8,0.8,0.8]);
                hold on;
            end

        end


        %Plot red track at last to ensure it floats above
        for i=1:(k-1)
            i_track=unstable_track(i);
            track = adjacency_tracks{i_track};
            track_points = all_points(track, :);
            plot(track_points(:,1), track_points(:, 2),'Color','r');
            hold on;
        end

        %% ------ Choosing points to highlight --------------------------------
        %Making First Plot
        i=1;
            %Normal Points
        p_plot=plot(points{i}(:,1),points{i}(:,2),'bx');
            %Most unstable point
        [~,max_pos]=max(points{i}(:,2));
        max_plot=plot(points{i}(max_pos,1),points{i}(max_pos,2),'ro');

            %Unstable point (omega_i>0)
        unstable_points=points{i}(:,2)>0;
        if any(unstable_points)
            unstable_plot=plot(points{i}(unstable_points,1),...
                points{i}(unstable_points,2),'y+');
        else
            unstable_plot=plot([0],[1E20],'y+');
        end

        % Controls of Plot
        title_name=['F=' num2str(F) ' Re=' num2str(Re) ' \theta=' ...
            num2str(rad2deg(theta)) '^\circ  '];
        title([title_name ' \alpha=' num2str(alpha_loop(i))]);
        xlabel('\omega_r');
        ylabel('\omega_i');
        axis([-1  5 -1 0.2]);
        %   axis([-9 14 -2 1]);
        %   ylim([-6 1.5])
        hold off;
        sld.Max=length(alpha_loop);
    end
end
```

B.5.7 'omg_track_video.m'

```
function omg_track_video(points,tracks,adjacency_tracks,...
    alpha_loop,Video_name,title_name)
% This is subroutines for visualizing data generated from a 'track' script
% and outputing it as a video file.
```

```
%% ------ Set up Video ---------------------------------------------------

Video1 = VideoWriter([Video_name '.mp4'],'MPEG-4');
open(Video1);

h1 =figure;
    set(h1,'Position',[0 0 1200 900]);
%     set(h1,'doublebuffer','off');
%     set(h1,'Visible','Off');
    hold on;
    xlabel('\omega_r');
    ylabel('\omega_i');
%     axis([-1  5 -1 0.2]);
       axis([-2 4 -4 0.5]);
    %   ylim([-6 1.5]);

%% ------ Plot all Tracks -----------------------------------
disp(['Plotting Tracks...']);

n_tracks = numel(tracks);
all_points = vertcat(points{:});

unstable_track=zeros(1,n_tracks);
k=1;

for i_track = 1 : n_tracks

    track = adjacency_tracks{i_track};
    track_points = all_points(track, :);

%     Detect if track cross x-axis and mark it red, and Plot track.
    if any(track_points(:,2)>0)
        unstable_track(k)=i_track;
        k=k+1;
    else
        plot(track_points(:,1), track_points(:, 2),'Color',[0.8,0.8,0.8]);
    end

end


%Plot red track at last to ensure it floats above
for i=1:(k-1)
    i_track=unstable_track(i);
    track = adjacency_tracks{i_track};
    track_points = all_points(track, :);

    plot(track_points(:,1), track_points(:, 2),'Color','r');
end

%% ------ Choosing points to highlight --------------------------------
%Making First Plot
i=1;
    %Normal Points
p_plot=plot(points{i}(:,1),points{i}(:,2),'bx');
    %Most unstable point
[~,max_pos]=max(points{i}(:,2));
max_plot=plot(points{i}(max_pos,1),points{i}(max_pos,2),'ro');

    %Unstable point (omega_i>0)
unstable_points=points{i}(:,2)>0;
if any(unstable_points)
    unstable_plot=plot(points{i}(unstable_points,1),...
        points{i}(unstable_points,2),'y+');
else
    unstable_plot=plot([0],[1E20],'y+');
end

% Controls of Plot
title([title_name ' \alpha=' num2str(alpha_loop(i))]);
hold off;

%% ------ Plot points for each frame ----------------------------
disp(['Plotting each frames...']);
for i=1:length(alpha_loop)
    title([title_name ' \alpha=' num2str(alpha_loop(i))]);
    set(p_plot,'XData',points{i}(:,1),'YData',points{i}(:,2));

    [~,max_pos]=max(points{i}(:,2));
    set(max_plot,'XData',points{i}(max_pos,1),'YData',points{i}(max_pos,2));

    unstable_points=points{i}(:,2)>0;
    if any(unstable_points)
         set(unstable_plot,'XData',points{i}(unstable_points,1),...
             'YData',points{i}(unstable_points,2));
    else
         set(unstable_plot,'XData',[0],'YData',[1E20]);
```

```
    end

    drawnow;
    writeVideo(Video1, getframe(h1));
end
close(Video1);
```