Master of Science Thesis

Hybrid Eulerian-Lagrangian Vortex Particle Method

A fast and accurate numerical method for 2D Vertical-Axis Wind Turbine

L. Manickathan B.Sc.

Date TBD

Faculty of Aerospace Engineering · Delft University of Technology

Hybrid Eulerian-Lagrangian Vortex Particle Method

A fast and accurate numerical method for 2D Vertical-Axis Wind Turbine

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

L. Manickathan B.Sc.

Date TBD

Faculty of Aerospace Engineering · Delft University of Technology

Delft University Of Technology
Department Of
Aerodynamics and Wind Energy

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Hybrid Eulerian-Lagrangian Vortex Particle Method"** by **L. Manickathan B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: Date TBD

Head of department: _____
prof.dr.ir. G.J.W. van Bussel

Academic Supervisor: _____
dr.ir. C.J. Simao Ferreira

Academic Supervisor: _____
dr.ir. A. Palha da Silva Clerigo

Summary

This is the summary of the thesis.

Acknowledgements

I wish to thank the following persons...

Delft, The Netherlands
Date TBD

L. Manickathan B.Sc.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

1-D	One-Dimensional
2-D	Two-Dimensional
AD	Actuator Disk
BEM	Blade Element Momentum
CFD	Computional Fluid Dynamics
CG	Continuous Galerkin
CSVM	Constant-Strength Vortex Method
DG	Discontinous Galerkin
DOF	Degrees of Freedom
FDM	Finite Difference Method
FEM	Finite Element Method
FE	Forward Euler
FMM	Fast Multipole Method
FVM	Finite Volume Method
GPU	Graphical Processing Units
HELVPM	Hybrid Eulerian-Lagrangian Vortex Particle Method
ICNS	Incompressible Navier-Stokes
IPCS	Incremental Pressure Correction Scheme
ISC	Impulsively Started Cylinder
LHS	Left Hand Side
LSTSQ	Least-Square solution method

MPI	Message Passing Interface
MRS	Modified Remeshing Scheme
PC	Population Control
PIV	Particle Image Velocimetry
PSE	Particle Strength Exchange
RWM	Random Walk Method
SCS	Simple Coupling Strategy
SRS	Simple Redistribution Scheme
VAWT	Vertical-Axis Wind Turbine
VPM	Vortex Particle Method

Chapter 1

Introduction

Conventional energy resources such as fossil fuels and nuclear energy are not only limited but also pose adverse effects on the environment. Therefore, we are striving to find a cheap and renewable source of energy. Wind energy is such source of energy, getting more popular and more affordable. Novel wind turbine designs such as Vertical-Axis Wind Turbine (VAWT) are now a promising research field that can satisfy this growing demand.

VAWTs are unlike the normal wind turbines, which are mounted on a mast away from the ground and generate energy by spinning perpendicular to the ground, figure 1.1, whereas the Horizontal-Axis Wind Turbine (VAWT), spins parallel to the ground with its hub located at the ground, figure 1.1b. The VAWT has it's generator located at the ground, allowing it to be easily accessible and maintained. However, the main advantage is the early wake dissipation of VAWTs. Near-wake experiments of Ferreira (2009) [55], and simulations of Vermeer (2003) [64] have shown that the fluid past the turbine is more turbulent. Due to this higher turbulence, the flow is able to recover much earlier than convectional wind turbines. This allows the turbines to be placed much closer, potentially

(a) VAWT: Darrieus wind turbine[17]

(b) HAWT: Offshore wind turbine [18]

Figure 1.1: VAWT vs. HAWT

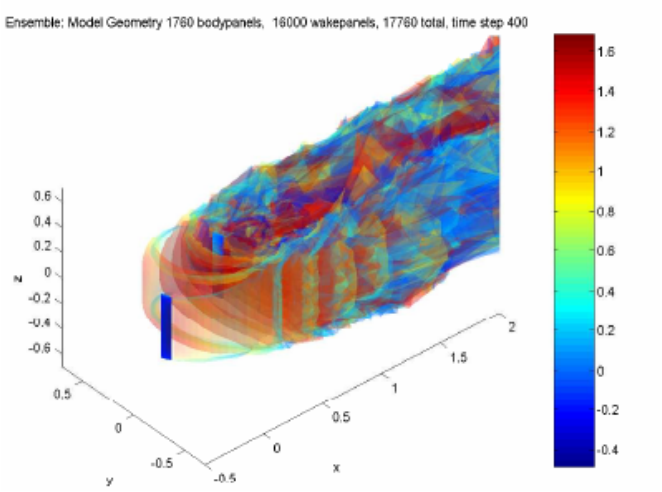


Figure 1.2: 3-D Unsteady Panel simulation of a Straight-bladed VAWT showing the strength of the shed vorticity. The VAWT blades interact with their own wake increasing the complexity of the wake geometry [26]

outputting more power per ground. Furthermore, VAWTs operate independently of the flow direction, and can operate at low wind speeds (i.e. at low tip-speed ratios).

However, there are some limitations that we must take into account. As the blades pass through their own wake, complex wake-body interactions take place, figure 1.2. These have adverse effects on the blade structure, making it more susceptible to fatigue. As the blade is constantly pitching, flow behaviors such as dynamic stall and constant vortex shedding take place [57]. These complex fluid behaviors makes it hard to predict the performance of a VAWTs and this is one of the reasons why VAWTs are not widely used.

In addition, a VAWT operates at a large Reynolds, number making accurate numerical methods computationally very expensive. So we see that we require a numerical method that can not only reproduce accurate results, but is also efficient at modeling the flow around the turbine.

1.1 Motivation and Goal

The goal of this research is to develop an efficient, reliable, and accurate numerical method for modeling the flow around a Two-Dimensional (2-D) VAWT, enabling to compute the correct performance characteristics. The two approaches of investigating the flow around a turbine are by either using a numerical method to model the flow, or by performing an experimental test, for example in a wind tunnel.

To understand the unsteady aerodynamic behavior, Particle Image Velocimetry (PIV) has been a useful tool to visualize the flow around the turbine. PIV was used by Ferreira et al. (2007) [56], showing that it is possible to measure the flow characteristics around the blade. The downside to experimental investigation is that it is very expensive to investigate all types of airfoil geometries, blade geometries and VAWT configurations. However, investigating this is vital in understanding the performance characteristics of

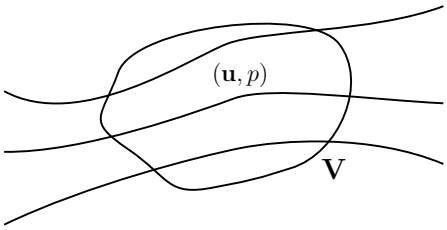


Figure 1.3: Eulerian formulation of the fluid. We observe a given volume $\mathbf{V}$ and evaluate the change in properties of the fluid, velocity $\mathbf{u}$ and pressure p at time passes.

VAWT. Furthermore, it is difficult to perform experiments on array of wind turbines in a wind tunnel.

Numerical methods are therefore a popular alternative as the cost of simulation is becoming progressively smaller, and the accuracy of the models are increasing day by day. Actuator Disk (AD) and Blade Element Momentum (BEM) models are the simplest models, built upon satisfying the momentum balance of the turbine with the fluid. The advantage is that they are very quick, however they lack the accuracy that are obtained by experimental simulation. Flow phenomenons such as dynamic stalls and flow separations cannot be modeled by these methods, and therefore we must rely on more powerful tools.

To ensure more accuracy, one has to solve the Navier-Stokes equation of the flow around the turbine without large simplifications. Computional Fluid Dynamics (CFD) methods discretize the fluid into smaller cells (or volumes) and solve the Navier-Stokes equations. This type of formulation is known as an Eulerian formulation as we are evaluating the change in flow property in a given cell/volume, figure 1.3. In order to fully resolve the flow around the turbine, we would need a fine mesh near the blade where we have small scale vorticies. However, far from the body, where these vorticies dissipated into low frequency vortical structures, we can have lower mesh resolution. This means that at various regions of the flow, we require mesh resolutions of various magnitudes. This becomes a problem when we have moving boundaries as the mesh has to be adapted depending on the location of the body.

An alternative method is to use a Lagrangian formulation of the Navier-Stokes equations, known as vortex methods. These methods employ vorticity transport equations which makes them ideal for describing the evolution of the wake vorticity. Furthermore, they do not require cells/volumes to describe the domain. In addition, they use simulation acceleration methods such as Fast Multipole Method (FMM) and parallel computation in Graphical Processing Units (GPU) making them orders of magnitude faster than the typical CFD methods. However, vortex method cannot inherently take in account the solid body. They require additional methods that can describe the effect of the body in the fluid and the vorticity generated from the body.

We see that Eulerian method is accurate when describing the blade-wake interaction but not efficient when describing multi-scale domains. The Lagrangian method is very efficient in evolving the vorticity of the fluid. Due to auto-adaptive nature of the Lagrangian method, it is an ideal choice when describing the multi-scale flow characteristics. However, it is not efficient in resolving the near-body region, where the vorticity is generated. Therefore, in order to use the advantages of both methods, we have decided to use a

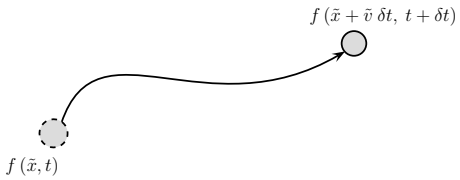


Figure 1.4: Lagrangian formulation of the fluid. We track the path of the individual fluid elements as time passes.

domain-decomposition method, referred to as Hybrid Eulerian-Lagrangian Vortex Particle Method (HELVPM).

For the HELVPM, the Eulerian grid method will be used at the near-wall region of the blades, and the Lagrangian vortex method will be used in the wake region of the body. With proper coupling of these methods, we can ensure that this numerical method can capture not only the near-wake phenomena such as vortex shedding, dynamic stall, and the wake-body interaction, but also the large-scale flow structures such as the evolution of the VAWT wake.

1.2 Research Aim and Plan

We have formulated a research that can help us accomplish our goal. The research questions that are derived from the goal of the project is as follows:

Research Questions:

- *Is it possible to develop an efficient and accurate numerical method by an hybrid approach, with the vortex particle method solving the wake, and the Navier-Stokes grid solver solving the near-body region?*
- *Will it be able to predict similar performance characteristics and flow phenomena as observed from the experiments?*
- *Will it be capable of simulating the blade-wake interaction and the dynamic stall?*
- *Where are the errors and what are their sources?*

In order to answer the research questions, the goal of the project is to develop an efficient and accurate numerical method that is not only capable of capturing the small scale flow phenomena such as the dynamic stall and the vortex shedding, but is also efficient at modeling the evolution of the wake. Once the model have been developed, we will verify the approach and validate it against cases obtained from literature.

Research aim and plan:

- *Develop the hybrid method for capturing small-scale phenomena and large scale phenomena.*

- *Verify the efficiency, reliability, and the accuracy of the model.*
- *Verify and validate the model with test cases from literature.*

The innovativeness of this project is that such hybrid modeling has not been yet applied for the wind energy problem case. Through the parallelization of the vortex particle method in a GPU and employing solver acceleration techniques such as the FMM, this simulation could give an edge in the understanding the flow behavior of a VAWT.

1.3 Introduction to Hybrid Eulerian-Lagrangian Vortex Particle Method

The Hybrid Eulerian-Lagrangian Vortex Particle Method (HELVPM) is a domain decomposition method, where the Eulerian method and the Lagrangian method solves different regions of the fluid. The domain decomposition method simply splits the domain of interest and uses appropriate method in each domain. The Eulerian formulation will be used at the near-wall region, where we need proper description of the vorticity generation at the boundary, and the Lagrangian formulation is used away from the body, where we only need to evolve the vorticity field. Figure 1.5 shows the decomposition of the domain in the gridded and the non-gridded region.

Several studies have already been done: Cottet and Koumoutsakos (2000a)[21], Guermond and Lu (2000) [30] simulated the advection dominated flows; Ould-Salhi et al. (2001) [49] blended the finite difference and vortex method together; Winckelmans et al. (2005a) [66] investigated the trailing vorticies; Daeninck (2006) [23] used a simplified coupling strategy, coupling Vortex Particle Method and Finite Diference Method; Stock (2010) [59] expanded Daeninck's strategy, coupling Vortex Particle Method and Finite Volume Method and modeled a 3-D rotor.

When evaluating the previous works, we see that not all domain decomposition methods are the same. The main difference between the methods is their coupling strategies. Most works employ the Schwartz alternating method to couple the vortex particle method and the grid solver. The Schwartz alternating method (or sometimes referred to as Schwartz iterative method), couples the vortex particle method and the grid solver by iteratively determining the boundary condition such that the stream functions in both domains, ψ_L and ψ_E in Ω_L and Ω_E respectively, match at the overlap region $\Omega_E - \Omega_L$, figure 1.5. The summary of a single iteration of the Schwartz alternating method is as follows:

- Determine the Eulerian boundary condition, the stream function ψ_{Γ_E} at the Eulerian boundary Γ_E, extracted from the Lagrangian stream function $\psi_L$in the Lagrangian domain Ω_L.
- Solve for the stream function ψ_E in the Eulerian domain Ω_E with the new boundary condition Γ_E.
- Determine the Lagrangian condition, the stream function ψ_{Γ_L} at the Lagrangian boundary Γ_L, extracted from the Eulerian stream function ψ_E in the Eulerian domain Ω_E.

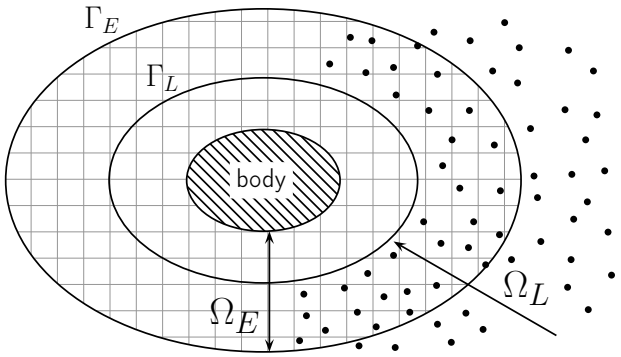


Figure 1.5: Standard domain decomposition using Schwartz iteration for coupling the two methods. Eulerian domain Ω_E (near the body), and Lagrangian domain Ω_L (away from the body). Figure is based on Guermond (2000) [30]

.

- Solve the stream function ψ_L in the Lagrangian domain with the boundary conditions ψ_{Γ_L} at the Lagrangian boundary Γ_L.

This procedure is iterated until the stream functions of both domains converge [49]. Once the stream function is determined in both the domains, the velocity field can be obtained. Using the velocity field, we can evolve the vorticity field in the Lagrangian domain.

As we realized now, the downside to this procedure is that we have to solve the stream function in both Ω_E and Ω_L iteratively, until we converge to a solution. This makes the computation very expensive, especially when we are dealing with large numbers of vortex particles. Therefore, for this project, we are using the coupling technique that is based on the research work of Daeninck (2006) [23] and Stock (2010) [59]. However we had to perform a correction to their scheme to ensure the conservation of circulation is satisfied.

1.3.1 Simple coupling strategy

This approach will be referred to as the Simple Coupling Strategy (SCS). It is simpler than the Schwartz iterative method, as no iteration is needed for the coupling procedure. The basic algorithm consists of solving the vortex method in the full fluid domain using a relatively coarse resolution on the near-wall region. Then we use the grid solver in the near-wall region to capture the detailed features of the boundary layer and transfer the vorticity field in this region to the vortex particles, figure 1.6. Therefore, the grid solver essentially acts as the correction for the under-resolved regions of the Lagrangian method. The functionality of this strategy has been demonstrated by Daeninck and was found to be significantly faster than the Schwartz coupling strategy. The features of the simple coupling strategy can summarized as follows:

- Eulerian method is used to resolve the near-wall region, at the Eulerian domain Ω_E, enabling it to capture important features of the boundary layer (such as flow separation) with great accuracy.
- Lagrangian method is used to capture the wake, at the Lagrangian domain Ω_L, and to efficiently evolve the wake.

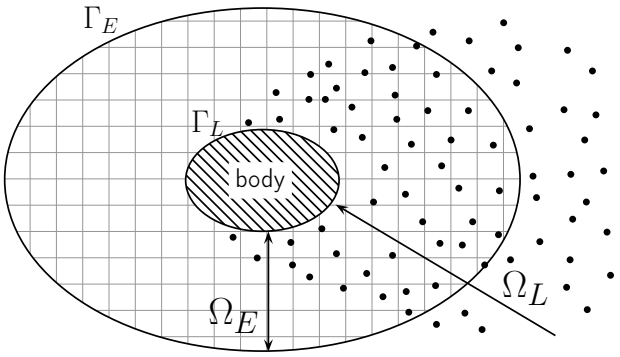


Figure 1.6: Modified domain decomposition without Schwartz alternating method. Lagrangian domain extends up to the surface of the body. Figure is based on Daeninck (2006) [23].

- The accurate solution of the Eulerian domain is transfered to the Lagrangian domain according to the coupling algorithm of Daeninck [23] and Stock [59]. In addition to their algorithm, a correction in done on the transfer to ensure conservation of circulation.
- The boundary conditions for the Eulerian domain are retrieved from the Lagrangian domain.

The algorithm to the Simple Coupling Strategy (SCS) follows from Daeninck's doctoral thesis, [23]. Figure 1.7 shows the overview to the algorithm and can be summarized as follows:

1. **Correct Lagrangian:** Use the solution of the Eulerian domain Ω_E (in the near-wall region) to correct the solution of the Lagrangian domain Ω_L, that is overlapping the Eulerian domain, ensuring that the *circulation is conserved*.
2. **Evolve Lagrangian:** With the modified solution, evolve the Lagrangian solution from time step t_n to next time step t_{n+1}.
3. **Determine Eulerian boundary conditions:** Use the Lagrangian solution of time t_{n+1} to determine the boundary conditions of the Eulerian domain at t_{n+1}.
4. **Evolve Eulerian:** With the boundary condition, evolve the Eulerian solution from t_n to t_{n+1}.

This is the basic approach for coupling the Eulerian method in the Eulerian domain Ω_E with the Lagrangian method in the Lagrangian domain Ω_L without the iterative Schwartz algorithm.

Furthermore, the SCS handles the Lagrangian boundary condition differently from the classic hybrid method. Typically during the evolution process of the Lagrangian domain, the shedding of the vorticity is also defined in the Lagrangian method. However, in our coupling strategy, the Lagrangian method is under-resolved at the boundary and cannot be used to resolve the vorticity flux at the body. Instead, we use the Eulerian method

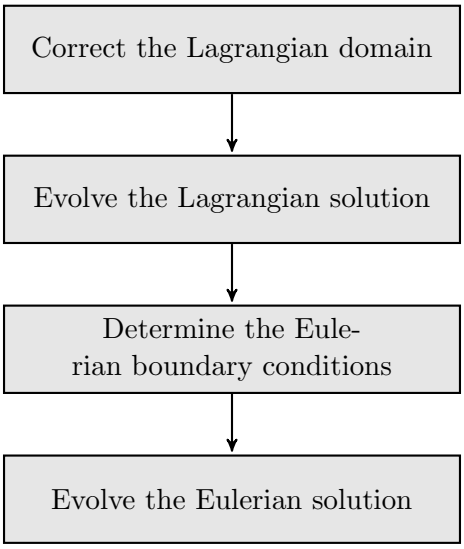


Figure 1.7: Flowchart of the simple coupling strategy. The flowchart shows the procedure to evolve both methods from t_n to t_{n+1}.

to resolve the boundary, and the Eulerian method acts as the vorticity generator for the Lagrangian method. However, there are some assumptions that we must satisfy, for this coupling strategy to be valid:

- At t_n before the evolution of both method to t_{n+1}, the Lagrangian solution matches Eulerian solution at the boundary of the near-wall region.
- After the evolution to t_{n+1}, the deviation of the Lagrangian solution (due to lack of vorticity flux at Lagrangian boundary), should be minimal.
- Even though the Lagrangian domain is under-resolved in the near-wall region, it should be able to provide accurate boundary conditions for the Eulerian external boundary.

1.4 Verification and Validation Test Cases

In order to assess the accuracy of this hybrid formulation, the following test cases haven been used:

Lamb-Oseen vortex [41] [61]
Lamb-Oseen vortex test case is an analytical solution derived from the NS equation, and is a test case for unbounded flow (without any wall). This is the first model that will be used to validate the Lagrangian method and Eulerian method separately. As it describes an unbounded flow, we do not need to concern with the vorticity generation problem. This helps us focus on just the evolution of the vorticity field.

Clercx-Bruneau dipole [15]
The Clercx-Bruneau dipole test case is the simple case of a colliding dipole with a wall. This test case will be used to verify and validate the coupling of the Eulerian and the Lagrangian method in the presence of a solid wall. This test cases focuses on the interaction of vorticity with the wall making it ideal to verify and validate the proper generation of vorticity and its transfer to the Lagrangian domain.

Impulsively started cylinder [39] [10] [6] [42]
The impulsively started cylinder test case is used to analyse the forces acting on the cylinder. This test case is used to verify and validate the lift and drag evolution of the cylinder exposed to free-stream flow.

Elliptic Airfoil [48]
The elliptic airfoil test case focuses on the flow separation past a lifting body. The elliptic airfoil is pitched at high angle of attack and the flow past the airfoil is comparatively unsteady and undergoes phenomena such as laminar separation bubble, flow separation and karman vortex shedding from the trailing edge of the airfoil. This helps us ensure the coupling strategy is accurate for complex flow phenomena.

1.5 Methodology

The initial steps of the development of the hybrid vortex method is as follows:

1. Develop the vortex particle method
2. Validate the vortex particle method against a Lamb-Oseen convection test case.
3. Develop the vortex panel method to deal with the boundaries for the vortex particle calculation.
4. Validate the vortex panel method by solving a potential flow around a cylinder.
5. Develop the grid solver that is based on the Finite Element method.
6. Validate the grid solver against test cases: impulsively starting cylinder, dipole-Wall interaction.

Once all the components have been validated, the methods will be coupled and validated against similar test cases.

7. Couple vortex particle, vortex panel and grid solver together.
8. Validate the hybrid method with test cases provide from literature.

1.6 Thesis Outline

!!! To be done at the end !!!

Chapter 2

Lagrangian Domain: Vortex Particle Method

2.1 Introduction to the Vortex Particle Method

Vortex Particle Method (VPM) is a numerical method employed in of computational fluid dynamics that deals with the evolution of the vorticity of the fluid in a Lagrangian description. In an Eulerian formulation, the fluid is viewed at a fixed window where the change in the fluid properties are evaluated. However, the Lagrangian formulation, regards the fluid as a collection of particles (or elements) carrying properties of the fluid (vorticity, mass, etc.).

Efficient discretization of the fluid domain becomes a difficult task for cases such as Vertical-Axis Wind Turbine (VAWT), where the wake geometry is usually unknown and highly unsteady. Discretizing such wake using Eulerian formulation usually becomes inefficient unless one manually adapts the mesh geometry over time. This is one of the advantage of the VPM. The VPM only needs fluid elements where there is vorticity meaning that the method is inherently auto-adaptive when discretizing the fluid domain. Furthermore, with computational acceleration methods such as Fast-Multipole Method (FMM) and parallel computation in Graphics Processing Units (GPU) enables an efficient evolution of the vorticity wake. However, the key advantage of the VPM is that it is ideal for capturing the resolving the long-time characteristics of the unsteady compact vortical structures that are shed off from the VAWT blades [59].

The advantage of the Lagrangian vortex method w.r.t the Eulerian method was summarized by Wee and Ghoniem [65]:

- Eulerian methods introduce dissipation, even in flows with zero velocity gradient. However such error as minimized at the convection of the Lagrangian method.
- The numerical stability is not restricted by the CFL condition.

- The support of the Lagrangian elements are a small fraction of the fluid domain. The support is confined to location of non-zero vorticity, making the method naturally grid adaptive.

The main literature on the VPM (the Lagrangian domain of the hybrid method), is the book of Cottet and Koumoutsakos, Vortex Methods: Theory and Practice [21]. It gives an insight on the fundamentals of the vortex method (specifically VPM) and gives a summary on hybrid methods.

2.1.1 Vorticity

The vorticity ω, the governing element of the VPM. It is given by

$$\omega = \nabla \times \mathbf{u}, \tag{2.1}$$

where $\mathbf{u}$ is the velocity vector field. The circulation Γ is defined by Stokes' theorem,

$$\Gamma = \int_L \mathbf{u} \cdot \mathrm{d}s = \int_A (\nabla \times u) \cdot \mathbf{n} \, \mathrm{d}A = \int_A \omega \cdot \mathbf{n} \, \mathrm{d}A, \tag{2.2}$$

and represents the flux integral of vorticity across the surface A, having the line s as boundary. Figure 2.1 depicts the relation with velocity $\mathbf{u}$, vorticity ω and the circulation Γ in an arbitrary domain.

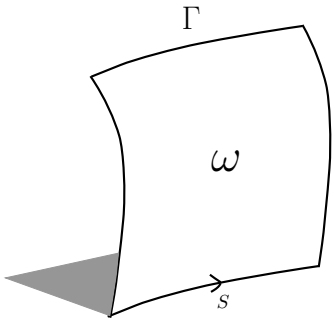


Figure 2.1: Definition of the circulation in the fluid.

2.1.2 Velocity-Vorticity formulation of the Navier-Stokes equations

The governing equation of the vortex particle method is the velocity-vorticity formulation $\mathbf{u} - \omega$ of the Navier-Stokes equations [21]. The 2-D incompressible Navier-Stokes momentum equation is given as,

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u}, \tag{2.3}$$

relating the velocity field $\mathbf{u}(\mathbf{x}, t)$ to the pressure field $p(\mathbf{x}, t)$, the kinematic viscosity ν and density ρ. Furthermore, the incompressibility constraint given as,

$$\nabla \cdot \mathbf{u} = 0, \tag{2.4}$$

must also be satisfied. To obtain the velocity-vorticity formulation, we should take the curl of equation 2.3,

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega, \tag{2.5}$$

which only relates the vorticity to the velocity enabling us to remove the pressure term. Note that as we are dealing with 2D flows, 3D terms (such as the stretching term) does not appear.

2.1.3 Viscous splitting algorithm

The VPM was initially used to model the evolution of incompressible, inviscid flows. However, in order to simulate a real flow, we must also deal with the viscous behavior of the fluid. Chorin in 1973 [13], showed that using the viscous splitting algorithm, it is possible to take the viscous effects of the flow into account.

The viscous splitting algorithm is a fractional step method, where the viscous and the inviscid part of the vorticity transport equation are dealt with in two subsequent sub-steps,

1. **Convection** (sub-step 1):
$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = 0; \tag{2.6}$$
2. **Diffusion** (sub-step 2):
$$\frac{\partial \omega}{\partial t} = \nu \nabla^2 \omega. \tag{2.7}$$

The first sub-step of the evolution deals with the convection of vorticity. The diffusion of vorticity field is dealt with in the second sub-step. There are several advantages to this type of evolution. As the convection and diffusion are handled separately, there is minimum dissipation during the convection step and furthermore, there is no restriction of the advection CFL number, see Wee (2006) [65].

There are many ways of treating the diffusion of the vorticity field. In this work, we initially used the modified interpolation kernel by Wee (2006) [65] that simultaneously discretizes diffusion and redistributes the vortex particles. Later, we used a simple re-distribution scheme of Tutty (2010) [62], that did not constraint the minimum time-step size, see section 2.4.

2.2 Spatial Discretization: Generation of Vortex Blobs

The particle in the Vortex Particle Method arises from the discretization of the vorticity field, equation 2.1.

2.2.1 Discrete form of vorticity field

The vorticity field is discretized using N vortex particles. The discrete vorticity field is given as,

$$\omega(\mathbf{x},t) \simeq \omega^h(\mathbf{x},t) = \sum_p \alpha_p(t)\,\delta\left[\mathbf{x}-\mathbf{x}_p(t)\right], \tag{2.8}$$

where δ is the Dirac delta function, α_p is the circulation carried by the particle at $\mathbf{x}_p$. We must note that ω^h is the discrete vorticity field and therefore in an approximation of the continuous vorticity field ω.

2.2.2 Biot-Savart Law

A velocity field $\mathbf{u}$ that satisfies the incompressibility constraint, equation 2.4, can be decomposed using the Helmholtz decomposition,

$$\mathbf{u} = \mathbf{u}_\omega + \mathbf{u}_\phi, \tag{2.9}$$

where $\mathbf{u}_\omega$ is the rotational component of the velocity and $\mathbf{u}_\phi$ is the irrotational component, i.e. the solenoidal and the potential velocity respectively. The irrotational component is equal to the free-stream velocity $\mathbf{u}_\infty$ for an incompressible, unbounded flows. Whereas for bounded flows, we must include the presence of the body as the full Helmholtz decomposition becomes,

$$\mathbf{u} = \mathbf{u}_\omega + \mathbf{u}_\phi + \mathbf{u}_b, \tag{2.10}$$

where $\mathbf{u}_b$ is the velocity due to the body, see section 2.5. The velocity can be related to the vorticity using the Biot-Savart law given as,

$$\mathbf{u}_\omega = \mathbf{K} \star \omega, \tag{2.11}$$

where $\star$ is the convolution of the vorticity with the 2-D Biot-Savart kernel $\mathbf{K}$ given by,

$$\mathbf{K} = \frac{1}{2\pi\left|\mathbf{x}\right|^2}\left(-x_2, x_1\right). \tag{2.12}$$

From the kernel, we see that as the distance to the kernel center approaches zero ($\mathbf{x} \to 0$), the kernel goes to infinity. The singularity of the kernel $\mathbf{K}$ is removed by mollifying the kernel distribution ensuring smooth velocity distribution.

2.2.3 Mollified vortex kernels

A mollified vortex particle is called the vortex blob having a non-zero vortex core size σ. A smoothing function ζ is used to mollify the kernel $\mathbf{K}$, satisfying the constraint $\int \zeta = 1$, such that the circulation is conserved. An ideal choice for a smoothing function is the Gaussian distribution, given as,

$$\zeta_\sigma = \frac{1}{k\pi\sigma^2}\exp\left(-\frac{\left|\mathbf{x}\right|}{k\sigma^2}\right), \tag{2.13}$$

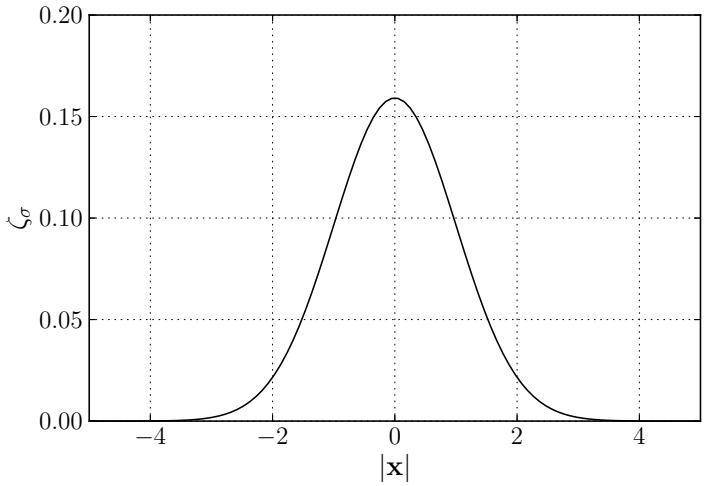


Figure 2.2: The smoothing function ζ_σ for a gaussian distribution with $k = 2$, $\sigma = 1$.

where typically k is either 1, 2 or 4 and determines the width of the kernel, with σ being the core-size of the vortex blob. Figure 2.2 depicts the smoothing function ζ_σ with $k = 2$ and $\sigma = 1$, decaying quickly away from the center of the core. The mollified Biot-Savart kernel $\mathbf{K}_\sigma$ is given as,

$$\mathbf{K}_\sigma = \mathbf{K} \star \zeta_\sigma, \tag{2.14}$$

giving us the discrete mollified vorticity field as,

$$\omega^h(\mathbf{x}, t) = \sum_p \alpha_p(t)\, \zeta_\sigma\left[\mathbf{x} - \mathbf{x}_p(t)\right], \tag{2.15}$$

and the discrete mollified velocity field as,

$$\mathbf{u}^h(\mathbf{x}, t) = \sum_p \mathbf{K}_\sigma\left[\mathbf{x} - \mathbf{x}_p(t)\right] \alpha_p(t). \tag{2.16}$$

Koumoutsakos and Chorin [21], explained that in order to ensure the smoothness of the velocity field, the vortex blobs need to have an overlap with each other. The overlap ratio Ov is defined as,

$$\mathrm{Ov} = \frac{\sigma}{h}, \tag{2.17}$$

where h is the nominal particle spacing, and σ is the vortex blob core size. Figure 2.3 shows the visual representation of this definition.

The overlap constraint is violated during the convection of the vortex blobs. Due to the strains in the flow, the vortex blobs cluster together at certain region and disperse at others. This localized clustering effect is seen as a Lagrangian grid distortion, which is dealt with using a remeshing technique, section 2.3.1.

2.2.4 Vortex blob initialization

Now the question arises on how should we initialize the particle's circulation strengths α_p. The common approach that is used as a standard is to estimate the particles strength

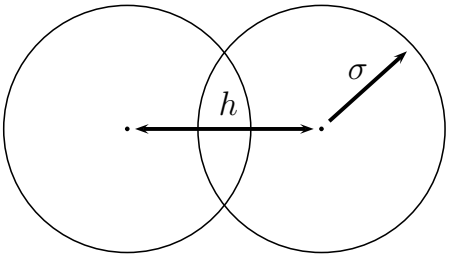


Figure 2.3: Vortex blob with an overlap $Ov = \sigma/h$

by the local properties,

$$\alpha_p = \omega_p \cdot h^2, \tag{2.18}$$

meaning that the particle carry the circulation of its local area. This might seem like a valid assumption as the circulation of a given area is the integral of the vorticity in the area, given by equation 2.2, and therefore we will be conserving the circulation as all the circulation in the fluid is represented by the blobs.

However, this type of initialization suffers some accuracy in the vorticity field itself as the vorticity field represented by the blobs is no longer the initial vorticity field, but the mollified vorticity field, equation 2.15. Barba and Rossi [1], has described this problem as gaussian blurring of the original vorticity field. Even though the particle have acquired the correct circulation strengths (i.e the local property), when evaluating the mollified vorticity field, we see that there is a mismatch with the original vorticity field, figure 2.4.

Another way of viewing this phenomenon is to say that the conservation of circulation is only valid globally (for an infinite domain), but once we try to conserve circulation locally (e.g. in a given sub-domain), is does not satisfy anymore. Figure 2.4 shows this effect for

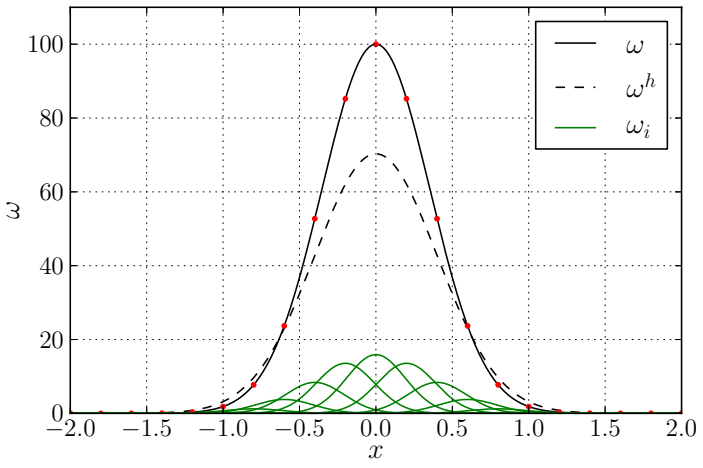


Figure 2.4: Mollified vorticity field of a gaussian vorticity distribution with overlap $= 1.0$, $\sigma = 0.19$, and $h = 0.19$. Vortex blob strengths was assigned using equation 2.18, sampling the exact vorticity [•, red dot]. Figure depicts the exact vorticity distribution ω [—, solid black], vorticity distribution of each blob ω_i [—, dashed green], and the mollified vorticity field from the blobs ω^h [- -, dashed black].

a simple gaussian initial vorticity distribution ω. The mollified vorticity field is given as ω^h and even though the integral of both function is the same (conservation of circulation is satisfied), the vorticity functions do not match. This is does not cause a lot of issues when we only dealing with vortex particle method, however once we start using domain decomposition methods, this problem is an issue. As the vorticity is the communication between both methods, we must have accurate vorticity distribution.

A common strategy, used by Koumoutsakos, Cottet, and other for recovering the initial vorticity field is to perform the Beale's method [3] [21].

Beale's Iterative Method

The Beale's method is particle circulation processing scheme where the circulation of the particles are modified such that the mollified vorticity field matches the indented vorticity field (the initial vorticity field). The recovery of the vorticity field is done by performing a discrete deconvolution,

$$\sum_{j}^{N} \beta_j \zeta_\sigma \left(\mathbf{x}_i - \mathbf{x}_j\right) = \omega_i, \tag{2.19}$$

where β_j is the circulation of the particles at positions $\mathbf{x}_j$ such that it matches the exact vorticity ω_i at the position $\mathbf{x}_i$ that we are evaluating. As we are try to solve for a N unknown problem, we must set up an N system of equations. Multiplying both sides with the area associated to the blobs, we get

$$\mathbf{A}_{ij}\beta_j = \alpha_i^{\text{exact}}, \tag{2.20}$$

where

$$\mathbf{A}_{ij} = \zeta_\sigma \left(\mathbf{x}_i - \mathbf{x}_j\right) \cdot h^2. \tag{2.21}$$

This is an $N \times N$ matrix containing the weights of the influence of each particle on each other. When we are dealing with large number of vortex blobs, we see that it is very expensive to invert the matrix $\mathbf{A}$, meaning that we would have to use a more efficient method. Furthermore as the deconvolution problem is a severely ill-condition problem [21], we should not directly invert the matrix. Beale's proposition to this problem was to iteratively solve for the solution,

$$\beta_j^{n+1} = \alpha_i + \beta_i^n - \mathbf{A}_{ij} \cdot \beta_j^n \tag{2.22}$$

We see that with just two iterations, the error between the mollified and exact vorticity field reduces drastically, figure 2.5. Koumoutsakos and Cottet [21], had shown that there was a drastic improvement in the velocity with just two to three iterations. However, the average vorticity of the blobs cell $\tilde{\omega} = \beta/h^2$ (red dot in figure 2.5), is more peaky and no longer matches the initial vorticity distribution. This means that if we try to fix the mollified vorticity distribution to match the correct initial vorticity distribution, we corrupt the local circulation even more.

The another downside of using the Beale's correction method that it is only valid for an infinite domain as it performs a discrete deconvolution of a gaussian kernel with an

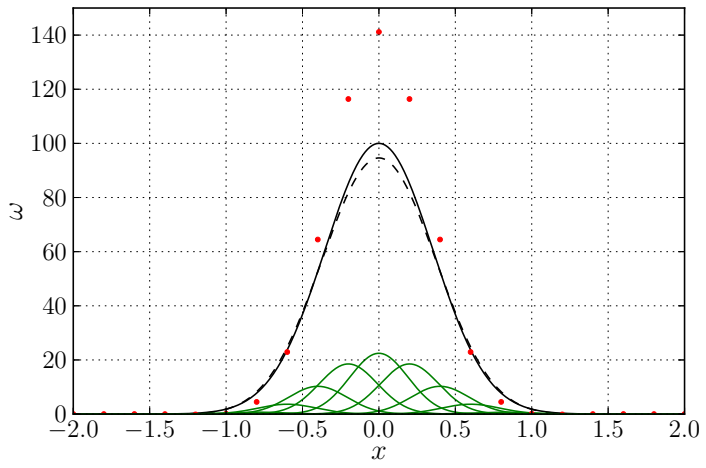


Figure 2.5: Mollified vorticity field after two Beale's iteration, with overlap $= 1.0$, $\sigma = 0.19$, $h = 0.19$. Figure depicts exact vorticity distribution ω [—, solid black], vorticity distribution of each blob ω_i [—, dashed green], the mollified vorticity field ω^h [- -, dashed black], and the corrected blob cell vorticity $\tilde{\omega} = \beta/h^2$ [•, red dot].

infinite span. Therefore it applies to all of the fluid domain, meaning that if we are dealing with a decomposed domain with finite bounds, the Beale's correction cannot be used and would result in spurious results. So the Beale's correction can and should only be used for correcting the vorticity field of the whole fluid domain.

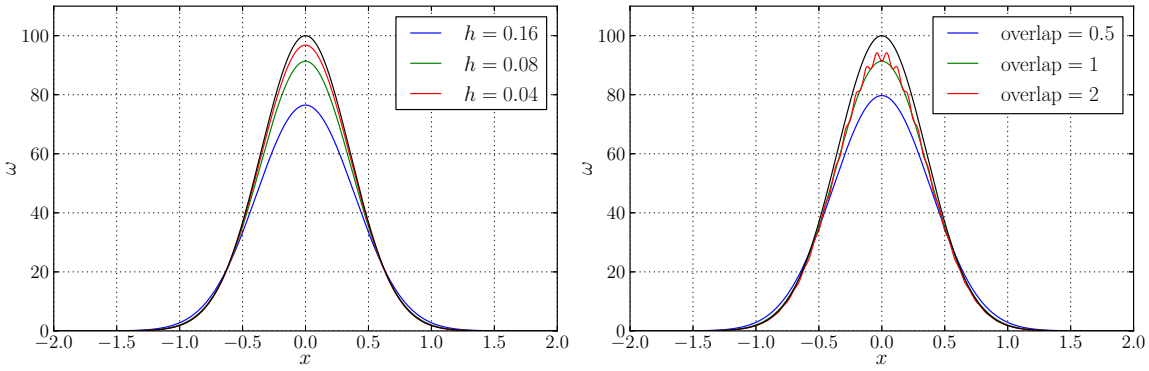
Convergence of particle discretization

An alternate method to reduce the gaussian blurring of the vorticity field is to reduce the overlap (i.e. increase the overlap number) of the vortex blobs, and also increase the spatial resolution. This approach does not solve the gaussian blurring problem, but only minimizes its effect.

Figure 2.6 shows mollified vorticity field results from modifying the spatial resolution parameters. Figure 2.6a shows the convergence of the mollified vorticity field ω^h to the exact vorticity field ω by reducing the nominal particle spacing h. The overlap ratio is set to overlap $= 1$, meaning that the blob core-size σ is equal to h. We see that by reducing blob core size, and simultaneously increasing the number of particles, the mollified vorticity converges to the exact vorticity.

The second parameter we can adjust is the overlap of the blobs, as seen in figure 2.6b. The blob spacing h is set to $h = 0.08$, and we see that by increasing the overlap number, the mollified vorticity approaches the exact vorticity field. However, as shown by Koumoutsakos [21], if the overlap is too low, we lose the smooth recovery of the vorticity field. This can be seen when the overlap $= 2.0$. We see that the mollified vorticity field has a fluctuating solution. This would results in non-smooth velocity field which is an acceptable solution.

Therefore, to minimize the error between the mollified vorticity field and the exact vorticity field, we will use an overlap $= 1.0$, and reduce the nominal blob spacing h to a



(a) Convergence of h with overlap $= 1.0$ **(b)** Convergence of overlap with $h = 0.08$

Figure 2.6: Convergence of the vorticity initialization by modifying the spatial resolution. Figure depicts exact vorticity field ω [—, solid black], the initialized vorticity distribution with various parameters.

minimum. The advantage of this approach is that we can employ this correcting in a finite domain unlike the Beale's correction. However, as this correction method only minimizes the effect and not directly solve the problem of mismatched vorticity fields, it is recommended that we find a solution to this problem in future.

2.3 Convection of vortex blobs

The convection equation 2.6 of the viscous-splitting algorithm, is solved as system of ODEs, where,

$$\frac{\mathrm{d}\mathbf{x}_p}{\mathrm{d}t} = \mathbf{u}\left(\mathbf{x}_p\right), \tag{2.23}$$

with

$$\frac{\mathrm{d}\alpha_p}{\mathrm{d}t} = 0. \tag{2.24}$$

This problem in now solved using a Lagrangian formulation where the vortex blobs is used to discretize the vorticity field. Using the Biot-Savart law, equation **??**, we can determine the induced velocities acting on each particles. The calculation of the induced velocities an N-body problem and is parallelized using GPU hardware and simplified using an FMM approach.

Once we determine the induced velocity acting on each vortex blobs, they can be convected using the equation 2.23. To retain accuracy during the convection process, we used a 4^{th} order Runge-Kutta method, an explicit time marching scheme. As the diffusion is done at the next sub-step, the strengths of the particles do not change during the convection process.

2.3.1 Remeshing scheme: Treating lagrangian grid distortion

During the convection step, the vortex blobs will start to cluster together at certain regions of the flow, where as at the other regions, we see that there are no blobs. This

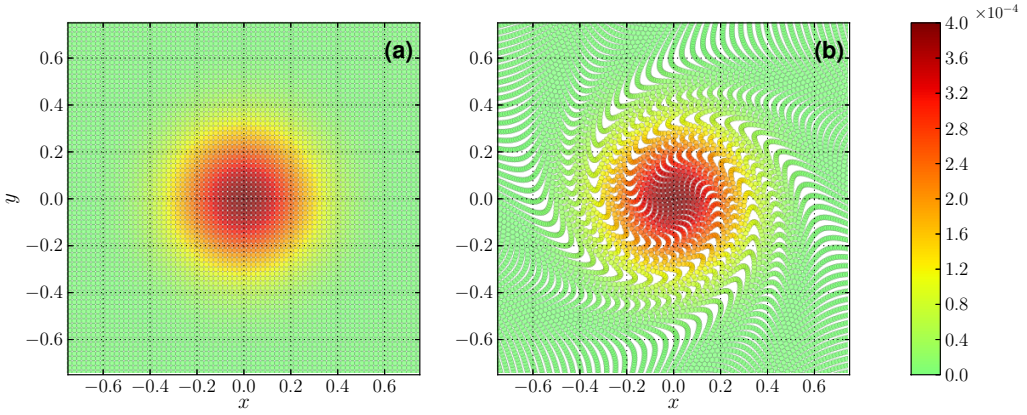


Figure 2.7: Lagrangian distortion of the vortex blobs after 100 time steps. The initial vorticity field is $\omega(\mathbf{x}, 0) = \exp(-12|\mathbf{x}|)$ with $\Delta t = 0.1$, $\sigma = 0.02$, and overlap $= 1.0$. Figure depicts **(a)** the initial distribution of the vortex blobs, and **(b)** the final distribution of the vortex blobs after 100 time steps.

clustering and dispersing effect of the blobs is due to the high strains in the flow, figure 2.7. This means that the overlap ratio is not satisfied everywhere in the flow. As we have seen before, in figure 2.6b, overlap ratio has a large impact on the mollified vorticity field and must be satisfied. In order to treat this Lagrangian grid distortion, we can remesh the vortex blobs on to a uniform grid, regaining our intended overlap ratio.

The remeshing is done by interpolating the strengths of the vortex blobs from distorted lagrangian grid on to a uniform grid. The strengths of the blobs of the new uniform grid is defined as,

$$\alpha_p = \sum_q \tilde{\alpha}_q W\left(\frac{x_p - \tilde{x}_q}{h}\right), \tag{2.25}$$

where the strengths of the blobs $\tilde{\alpha}_q$ of the distorted Lagrangian grid $\tilde{x}_q$ is transfered to the regular Lagrangian grid x_p using the interpolation kernel W. Figure 2.8 shows the remeshing of one vortex blob of the distorted grid on to the structured uniform grid. During this transfer, we must ensure that the properties of the fluid are conserved. The interpolation kernel is constructed by ensuring that the circulation, the linear impulse, and the angular impulse of the fluid is conserved. For this project, we used the M'_4 interpolation kernel.

M'_4 interpolation kernel

The M'_4 interpolation kernel is an efficient interpolation kernel that has been used to reconstruct a smooth distribution, and was introduced by Monaghan in 1985 [46]. For a One-Dimensional (1-D) problem, the M'_4 interpolation kernel is given as,

$$\mathrm{M'}_4(\xi) = \begin{cases} 1 - \frac{5\xi^2}{2} + \frac{3|\xi|^3}{2} & |\xi| < 1, \\ \frac{1}{2}(2 - |\xi|)^2 (1 - |\xi|) & 1 \leqslant |\xi| < 2, \\ 0 & 2 \leqslant |\xi|, \end{cases} \tag{2.26}$$

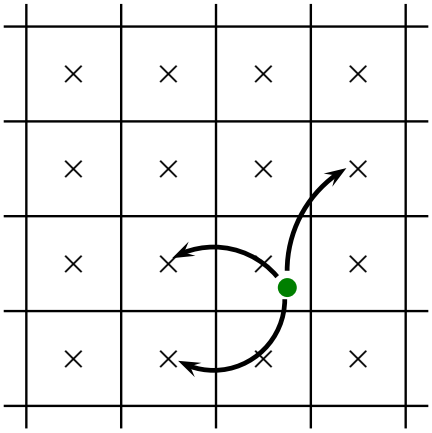


Figure 2.8: Remeshing of a single vortex blob [•, green dot] onto a uniform grid defined by the (4×4) 2-D stencil.

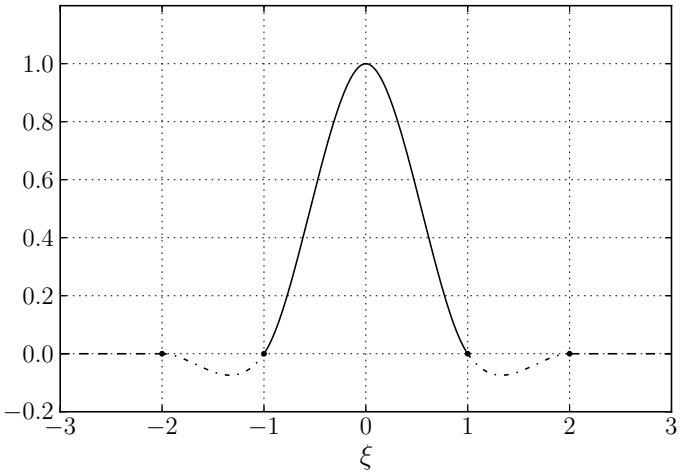


Figure 2.9: M_4' interpolation kernel, a third-order, piecewise smooth, B-spline kernel by Monaghan [46]

where

$$\xi = \frac{x_\nu - x_i}{h}, \tag{2.27}$$

is a non-dimensional parameter, relating the position of the particle x_ν to the position of the i^{th} interpolation node x_i. The M'_4 is a third-order accurate, piecewise smooth, B-spline kernel, and with the $m = 4$, it has 4 support nodes in each dimension. Figure 2.9 shows the distribution of the kernel. For the two dimensional problem, the 2-D interpolation kernel is just the tensor product of the 1-D interpolation kernel, and will have a $4^2 = 16$ support nodes, as seen in figure 2.8.

2.4 Diffusion of Vortex Methods

So far, we have ignored the viscous effect of the flow, and in order to simulate a real flow, we must perform the diffusion of the vorticity field. Chorin simulated a viscous flow using the viscous splitting algorithm. The flow is segregated to inviscid and viscous component and during the second sub-step we deal with the diffusion of the vorticity, as show in equation 2.7. The diffusion term of the viscous splitting algorithm can be solved as a system of ODEs, similar to the convection step, given as,

$$\frac{\mathrm{d}\mathbf{x}_p}{\mathrm{d}t} = 0, \tag{2.28}$$

with

$$\frac{\mathrm{d}\alpha_p}{\mathrm{d}t} = \nu\Delta\alpha_p. \tag{2.29}$$

So in the diffusion step, we fix the position of the vortex blobs, and only modify the strengths of the particles. This process mimics the diffusion of vorticity. Chorin in 1973 [13], initially employed the Random Walk Method (RWM), which generates and disperse vorticity using pseudo-random number algorithm. However, this method suffers some limitations in accuracy, and since then methods such as Particle Strength Exchange (PSE) method [25], has been a common approach to treat the diffusion.

Particle Strength Exchange

The Particle Strength Exchange method, first proposed by Mas-Gallic in 1989 [25], showed that diffusion can be treated for a particle method with an isotropic, or an anisotropic viscosity by approximating the diffusion operator (a Laplacian) with an integral operator, and discretize the method with particles [58].

Vortex Redistribution Method

An alternative method to simulate the diffusion is to use the Vortex Redistribution Method (VRM) [53]. The model simulates diffusion by distributing the fraction of circulation of the vortex blobs to each other satisfying the diffusion. The model is based on conserving the moments of the particles by satisfying a linear system of equations. The circulation of the particle are transfer to the nearby particles that are,

$$h_\nu = \sqrt{\nu\Delta t_d} \tag{2.30}$$

where h_ν is the diffusion distance and is directly related to the kinematic viscosity ν and the diffusive time-step Δt_d of the simulation. This means that the VRM scheme (and also the PSE) requires a search algorithm to determine the particles that are within the zone of influence. A direct evaluation would require an $\mathcal{O}\left(N^2\right)$ evaluation. However, this can be optimized using a search tree algorithm, speeding up the search to an $\mathcal{O}\left(\log N\right)$.

2.4.1 Modified remeshing for treating diffusion

From further investigation of the VRM, Ghoniem and Wee saw that the VRM is similar to the remeshing strategy that we used to counter the lagrangian distortion during the convection process. So, Ghoniem and Wee in 2006 [65], proposed to combine the remeshing and the diffusion together in one process. The application of this methodology was later validated by Speck in 2011 [58]. This Modified Remeshing Scheme (MRS), performs diffusion by modifying the interpolation kernel of the remeshing scheme. The key advantage of the MRM is that, now we are dealing with a uniform grid, and no longer requires a search algorithm to find the particles in the zone of influence. This significantly reduced the computational cost, making this diffusion scheme practical for a large scale simulations.

Ghoniem and Wee rederived the interpolation kernel that can perform both the remeshing and the diffusion simultaneously. The kernel transfers the correct fraction of circulation during the remeshing such that it satisfies the diffusion equation. The M'_4 kernel for treating the diffusion is given as,

$$\mathrm{M}'_4(\xi, c) = \begin{cases} 1 - \frac{5\xi^2}{2} + \frac{3|\xi|^3}{2} - c^2\left(2 - 9\xi^2 + 6|\xi|^3\right) & |\xi| < 1, \\ \frac{1}{2}(2 - |\xi|)^2(1 - |\xi|) - c^2(2 - |\xi|)^2(1 - 2|\xi|) & 1 \leqslant |\xi| < 2, \\ 0 & 2 \leqslant |\xi|, \end{cases} \tag{2.31}$$

where

$$c^2 = \frac{\nu \Delta t_d}{h^2}, \tag{2.32}$$

is a non-dimensional number that corresponds to the transfer weight for the diffusion. This additional terms in the interpolation kernel accounts for the diffusion process. When $c \to 0$, the interpolation kernel simply turns in to the standard remeshing kernel. Ghoniem and Wee also investigated the error growth and the stability properties of the interpolation kernel in the Fourier space and have determined that for a stable diffusion and remeshing, we have to ensure that,

$$\frac{1}{6} \leqslant c^2 \leqslant \frac{1}{2}. \tag{2.33}$$

However, we see that this c^2 constraint imposes a direct constraint on the maximum and the minimum Δt_d. This would mean that the diffusion time step size Δt_d will be a multiple of the convection step Δt_c,

$$\Delta t_d = k_d \cdot \Delta t_c, \tag{2.34}$$

so the diffusion of the vortex blobs will not be done at every step of the evolution process. This is a problem for hybrid method as we couple the Lagrangian and the Eulerian method at every step. If the Lagrangian method does not perform diffusion at every step, from the Eulerian method's point of view, it would seem that the Lagrangian vorticity diffuses in a discontinuous fashion. This discontinuous behavior of the Lagrangian method (w.r.t. the Eulerian method), can cause stability issues during the coupling process, and should take care of.

We could minimize this problem by modifying the Δt_c such that it matches the diffusion time step (i.e $\Delta t_c = \Delta t_d$), so the diffusion is performed at every step. This was a feasible

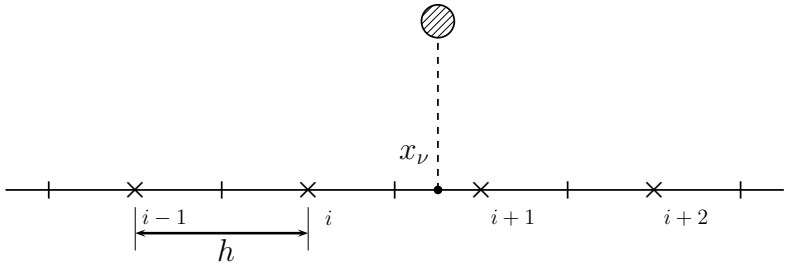


Figure 2.10: One dimensional simple redistribution scheme, diffusing the vortex blobs at $x_i \leqslant x_\nu \leqslant x_{i+1}$, onto the four stencil points $k = i-1, \ldots, i+2$, with a grid spacing h.

solution for our initial investigations with low Reynolds number flows, however when had to deal with the high Reynolds number flows, where the convection time step have to be small, we needed a scheme that was not constrained by the minimum diffusion time step.

Simple redistribution scheme

The simple redistribution scheme, based on the Shankar and Van Dommelen [53], developed by Tutty in 2010 [62], makes it possible to remesh and diffuse the vorticity after every convection step. The strengths of the particles after the remeshing and diffusion is given as,

$$\alpha_i^{n+1} = \sum_k \alpha_k^n W_{ki}^n, \tag{2.35}$$

where W_{ki}^n is the fraction of circulation from vortex blob k transferred to vortex blob i by the redistribution during the time step n. Like the MRS for treating diffusion, section 2.4.1, this Simple Redistribution Scheme (SRS) works by transferring the strengths of the particles to a fixed nodes rather than the neighboring nodes, figure 2.10. Therefore, if we use a uniform nodes, we do not need to employ a search algorithm to determine the neighboring nodes of the interpolation kernel. This makes the scheme as efficient as the MRS.

The fractions W_{ki}^n are calculated by conserving the vorticity, the center of the vorticity, the linear, and the angular momentum of the vortex blobs [62]. For the two dimensional problem, the redistribution fractions are simple tensors products of the x, y one dimensional redistribution fractions, given as,

$$W_{kl} = F_k G_l, \quad k = i-1, \cdots, i+2, \quad l = j-1, \ldots, j+2, \tag{2.36}$$

giving it a 16 point stencil. The one-dimension redistribution fractions for x-direction is a linear combination of the two basis solution for a conservative redistribution,

$$F_k = (1-\Delta) \cdot f_k + \Delta \cdot g_k, \quad k = i-1, \ldots, i+2, \tag{2.37}$$

having a four point stencil, as show in figure 2.10. The basis solution of redistribution

are

$$f_{i-1} = \frac{1}{2}(1 - f_i - \xi) \tag{2.38a}$$

$$f_i = 1 - 2\left(\frac{h_\nu}{h}\right)^2 - \xi^2 \tag{2.38b}$$

$$f_{i+1} = \frac{1}{2}(1 - f_i + \xi) \tag{2.38c}$$

$$f_{i+2} = 0 \tag{2.38d}$$

and

$$g_{i-1} = 0 \tag{2.39a}$$

$$g_i = \frac{1}{2}(1 - g_{i+1} - \xi_1) \tag{2.39b}$$

$$g_{i+1} = 1 - 2\left(\frac{h_\nu}{h}\right)^2 - \xi_1^2 \tag{2.39c}$$

$$g_{i+2} = \frac{1}{2}(1 - g_{i+1} + \xi_1) \tag{2.39d}$$

where ξ is given by equation 2.27. The $\xi_1 = \xi - 1$ is the distance between the k^{th} stencil node x_k and the vortex blob that is to be diffused with $x_i \leqslant x_\nu \leqslant x_{i+1}$. In the above equation, h_ν is the characteristic diffusion distance during the time Δt_d,

$$h_\nu = \sqrt{\Delta t_d \cdot \nu}. \tag{2.40}$$

The only constraint imposed for a positive redistribution fraction is,

$$\frac{h_\nu}{h} < \frac{1}{\sqrt{2}}. \tag{2.41}$$

giving us the maximum time step size constraint of,

$$\Delta t_d < \frac{h^2}{2\nu}. \tag{2.42}$$

So we see that, this scheme only posses a constraint on the maximum diffusion time step size. For high Reynolds number flows, the convection time step is usually much smaller than the diffusion time step, meaning that now we can ensure that diffusion is performed with every convection step (i.e $k_d = 1$).

2.5 Boundary conditions at solid boundary

So far, we have only dealt with unbounded flow. During bounded flow simulation, we must impose an addition constraint for the boundary, to simulate the flow with a geometry. Using the Helmholtz decomposition, we can have decompose the velocity field in to the rotational and the irrotational components, equation 2.9. We can use the potential component to prescribe the boundary conditions at the solid wall boundary,

$$\mathbf{u}_\phi = \nabla\Phi. \tag{2.43}$$

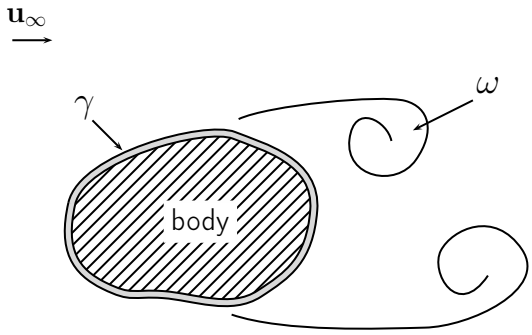


Figure 2.11: Extended vorticity field separated into vorticity in the fluid and the vortex sheet distribution confined to the body.

The incompressibility constraint results in a Laplace's equation for the potential field and a unique solution is obtained by enforcing the wall boundary conditions,

$$\mathbf{u}_b \cdot \hat{\mathbf{n}} = (\mathbf{u}_\omega + \nabla\Phi) \cdot \hat{\mathbf{n}}, \tag{2.44}$$

enforcing the no-through flow at the solid boundary wall, moving at $\mathbf{u}_b$, where $\hat{\mathbf{n}}$ is the normal vector of the solid boundary. The approach for determine the solution to the Laplace's equation is by Green's function formulation. This approach required a singularity distribution over the body resulting in the appropriate boundary condition. Doublets and/or source panels are used to attain the required potential such that equation 2.44 is satisfied.

Linked boundary conditions

However Koumoutsakos, Leonard and Pepin in 1994 [40], suggested to use vortex sheets to enforce the boundary conditions. This alternate approach of enforcing the solid boundary condition is by not to decompose the velocity field into potential and rotational but to consider the solid boundary as an extension of the vorticity field through vortex sheets γ, figure 2.11. Due to the non-zero tangential velocity at the surface, a sudden discontinuity in the velocity field can be considered as vortex sheet. So to enforce the boundary conditions of the solid wall, we must satisfy the no slip velocity at the boundary,

$$\mathbf{u} \cdot \hat{\mathbf{s}} = \mathbf{u}_b \cdot \hat{\mathbf{s}}. \tag{2.45}$$

Koumoutsakos [38], relied only on the vortex sheets to enforce the no-slip velocity. Koumoutsakos, Leonard and Pepin's paper in 1994 [40] stated that, by satisfying no-slip boundary condition, it directly satisfies the no-through boundary conditions, as these boundary conditions are linked. This was also been stated by Shiels [54] and have been further employed by Cooper, Mar and Barba in 2009 [19]. So enforcing the no-slip boundary condition directly satisfies the no-through constraint at the surface.

2.5.1 Boundary integral equations

The Lagrangian method that we are using for the hybrid scheme, is modified according Stock [59]. The Lagrangian method under-resolved the vorticity field in the near-wall

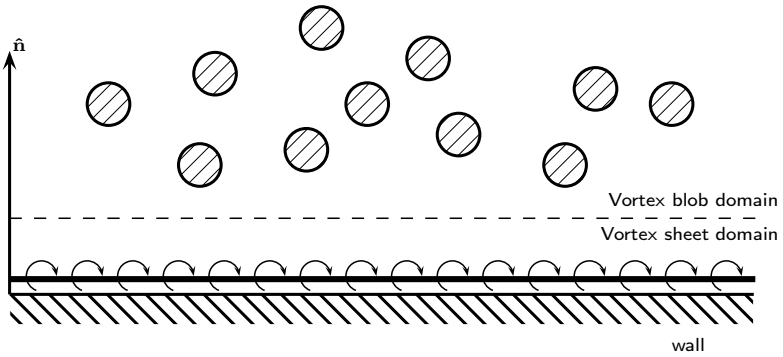


Figure 2.12: Extended vorticity field: Vortex sheet being an extension to the vorticity field (resolved by the vortex blobs), capable of capturing the body bounded vorticity distribution.

region. Furthermore, the vorticity of the fluid is segregated between the vortex blob domain and the vortex sheet domain, as seen in figure 2.12. The figure shows that, very near the wall the vorticity of the fluid is represented by the vortex sheet. In other words, the vortex sheet is an extension to the vorticity represented by the vortex blobs. The extended velocity field (of the extended vorticity field) is summarized as,

$$\mathbf{u} = \mathbf{u}_\omega + \mathbf{u}_\gamma + \mathbf{u}_\infty \tag{2.46}$$

where the $\mathbf{u}_\gamma$ denotes the velocity field induced by the vortex sheet. Enforcing the no-slip boundary conditions, we state that,

$$(\mathbf{u}_{\text{ext}} + \mathbf{u}_\gamma) \cdot \hat{\mathbf{s}} = \mathbf{u}_b \cdot \hat{\mathbf{s}} \tag{2.47}$$

where $\mathbf{u}_{\text{ext}} = \mathbf{u}_\omega + \mathbf{u}_\infty$ is the velocity field induced from the vortex blob domain. The equation states that the tangential component of the total velocity acting on the body should be equal to the tangential velocity of the body. So the induced velocity of the vortex sheet is given as,

$$(\mathbf{u}_{\text{ext}} - \mathbf{u}_b) \cdot \hat{\mathbf{s}} = \mathbf{u}_\gamma \cdot \hat{\mathbf{s}}. \tag{2.48}$$

Note that as the boundary condition is enforced inside the body (under the vortex sheet), the velocity induced by the vortex sheet $\mathbf{u}_\gamma$ is the negative of equation 2.48. Koumoutsakos [38], expressed the relation of the vortex sheet strengths to the no-slip boundary condition at the surface of the body (inside the body) through the Fredholm integral equation of the second kind,

$$-\frac{\gamma(s)}{2} + \frac{1}{2\pi} \oint \frac{\partial}{\partial n} \left[\log \left|\mathbf{x}(s) - \mathbf{x}\left(s'\right)\right|\right] \gamma\left(s'\right) ds' = \mathbf{u}_{\text{slip}} \cdot \hat{\mathbf{s}}. \tag{2.49}$$

where $\gamma(s)$ is the strength of the vortex sheet, and $\mathbf{u}_{\text{slip}} = (\mathbf{u}_{\text{ext}} - \mathbf{u}_b)$, is the slip velocity that we have to cancel. The Left Hand Side (LHS) of the equation states at the point s, the velocity discontinuity is due to the vortex sheet of that point and integral of all the other vortex sheets around the body. However, the equation 2.49 is not unique and accepts arbitrary solution for the vortex sheet strengths, therefore we must imposed an

additional constraint on the strength of the vortex sheet. The addition constraint is a constraint on the circulation of the fluid. To ensure conservation of circulation,

$$\Gamma = \Gamma_{\text{body}} + \Gamma_{\text{fluid}} = 0, \tag{2.50}$$

stating that the total circulation in the Lagrangian domain is the sum of the circulation of the body and the circulation of fluid, which should be equal to zero. As the body is in the domain of the vortex sheet, the circulation of the body is represented by the vortex sheet,

$$\Gamma_{\text{body}} = \Gamma_{\gamma_{\text{body}}}. \tag{2.51}$$

The circulation of the body is due to the motion of the body traveling at $\mathbf{u}_b$. One can consider the body as filled with uniform vorticity due to the rotation of the body. Therefore the circulation of a moving body can calculated simply integrating the "vorticity" inside the body,

$$\Gamma_b = \iint_{body} \nabla \times \mathbf{u}_b \; dA. \tag{2.52}$$

Furthermore, as we are using the Lagrangian method according to Stock [59], the boundary layer of the body is not resolved with the vortex blobs, as explained in figure 2.12. Therefore, the vortex sheet has to carry this neglected circulation of the boundary layer, and the circulation of the fluid is given as,

$$\Gamma_{\text{fluid}} = \Gamma_{\gamma_{\text{BL}}} + \Gamma_{\omega}, \tag{2.53}$$

where $\Gamma_{\gamma_{\text{BL}}}$ is the circulation of the boundary layer region represented by the vortex sheet, and Γ_{ω} is the total circulation captured by the vortex blobs. Combining equation 2.51 and 2.53 into equation 2.50, we derive the net circulation of the Lagrangian domain,

$$\Gamma = \Gamma_{\gamma_{\text{body}}} + \Gamma_{\gamma_{\omega}} + \Gamma_{\omega} = 0, \tag{2.54}$$

where the total circulation represented by the vortex sheet is given as,

$$\Gamma_{\gamma} = \Gamma_{\gamma_{\text{body}}} + \Gamma_{\gamma_{\text{BL}}}. \tag{2.55}$$

Thus, the constraint imposed on the net circulation of the vortex sheets is given as,

$$\Gamma_{\gamma} = -\Gamma_{\omega}. \tag{2.56}$$

ensuring that the total circulation of the fluid is zero. The total circulation of the vortex sheet is calculated by integrating the vortex sheet,

$$\Gamma_{\gamma} = \oint_{S} \gamma\,(s) \; ds. \tag{2.57}$$

So we have to solve for the vortex sheet satisfying both the equation 2.49 and the equation 2.57, which can be done using a panel method.

2.5.2 Panel method for treating no-slip boundary condition

Equation 2.49 is solved by discretizing the body into M vortex panels, giving us a system of M equation to determine the M unknowns of the strength of the vortex panels. This is the panel method, which has been extensively demonstrated by Katz and Plotkin [36]. Katz and Plotkins have shown several types of panel distributions with various orders of accuracy; from 0^{th} order point vortex or up to 2^{nd} order non-linear vortex panel. For this project, we have used a constant-strength vortex distribution that discretized the vortex sheet into straight segments, classified as Constant-Strength Vortex Method (CSVM).

Panel methods are constructed by discretizing the integral equation 2.49, and forming a system of M equations to solve the M unknowns of the vortex panel,

$$\underbrace{\begin{pmatrix} -\frac{1}{2} & a_{12} & \cdots & a_{1M} \\ a_{21} & -\frac{1}{2} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & -\frac{1}{2} \end{pmatrix}}_{\mathbf{A}_{MM}} \underbrace{\begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_M \end{pmatrix}}_{\vec{\gamma}} = \underbrace{\begin{pmatrix} \text{RHS}_1 \\ \text{RHS}_2 \\ \vdots \\ \text{RHS}_M \end{pmatrix}}_{\overrightarrow{\text{RHS}}}, \tag{2.58}$$

where A_{MM} contains the coefficients of the tangential induced velocity of vortex panels $\vec{\gamma}$ acting on each other. The $\overrightarrow{\text{RHS}}$ is given as,

$$\text{RHS} = \mathbf{u}_{\text{slip}} \cdot \hat{\mathbf{s}}, \tag{2.59}$$

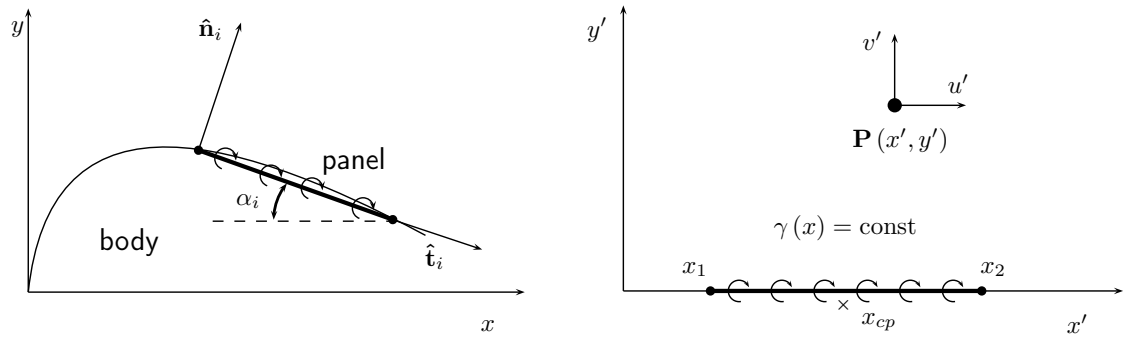
and contains the boundary condition to the system of equations. In addition, we have another constraint on the net circulation of the vortex panels, equation 2.57. In the discrete form, the equation is given as,

$$\sum_{i}^{M} \gamma_i \Delta s = \Gamma_\gamma, \tag{2.60}$$

and results in a $M+1$ system of equations for solving the M unknown strengths of the vortex panels,

$$\underbrace{\begin{pmatrix} -\frac{1}{2} & a_{12} & \cdots & a_{1M} \\ a_{21} & -\frac{1}{2} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & -\frac{1}{2} \\ \Delta s_1 & \Delta s_2 & \cdots & \Delta s_M \end{pmatrix}}_{\mathbf{B}_{(M+1)M}} \underbrace{\begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_M \end{pmatrix}}_{\vec{\gamma}} = \underbrace{\begin{pmatrix} \text{RHS}_1 \\ \text{RHS}_2 \\ \vdots \\ \text{RHS}_M \\ \Gamma_\gamma \end{pmatrix}}_{\overrightarrow{\text{RHS}}}, \tag{2.61}$$

the last equation imposing the circulation constraint on the panels. However, now we see that we have a set of $M+1$ equations with M unknowns, making it an overdetermined problem. The approach to solve such a problem is either by using a Least-Square solution method (LSTSQ), introducing a new unknown, or as used by Katz, eliminating an equation. Enforcing the no-slip boundary condition at each panel location is a vital criteria, and therefore we used the LSTSQ method for solving the problem.



(a) Panel discretization of the body in the global cartesian coordinates system (x, y) with the local panel coordinates system rotated by α_i.

(b) Constant-Strength Vortex panel in the local panel coordinate system (x', y') inducing the velocity $\mathbf{u}' = (u', v')$ on the point P.

Figure 2.13: The two coordinate system of the panel method problem. The figure depicts **(a)** the global panel coordinate system, and **(b)** the local panel coordinates system, as defined by Katz and Plotkins [36].

Constant-Strength Vortex Method

The Constant-Strength vortex method (CSVM) is based on the flat (straight) discretization of the vortex sheet, where the panel have constant vortex strength. To solve the strengths of the panel problem, we enforce the Dirichlet velocity boundary conditions at the collocation point x_{cp}, that is located just below the vortex sheet, shown in figure 2.13b. The coefficient a_{ij} of the influence matrix $\mathbf{A}$ is defined as,

$$a_{ij} = \hat{\mathbf{u}}_{ij} \cdot \hat{\mathbf{t}}_i, \tag{2.62}$$

which represents the tangential influence coefficient of j^{th} panel on the i^{th} panel. The influence coefficient is determined by prescribing the strengths of the vortex panels $\hat{\gamma}_i = 1$, resulting in an induced velocity $\hat{\mathbf{u}}_{ij} = (\hat{u}, \hat{v})_{ij}$ for a unit strength panel.

Figure 2.13a shows the discretization of the body into panels in the global coordinates system, defined by (x, y), where each panel is rotated by an angle α_i w.r.t to the global coordinate system. Rotating the axis (x, y) by α_i, we arrive at the local panel coordinate system (x', y'), as shown in figure 2.13b. In general, the induced velocity of the vortex panels are calculated in the local panel coordinate system, where the induced velocity of the vortex panel j on the collocation point i (in the panel coordinate system) is given as,

$$u'_{ij} = \frac{\gamma_j}{2\pi}\left[\tan^{-1}\frac{y'_i - y'_{j,2}}{x'_i - x'_{j,2}} - \tan^{-1}\frac{y'_i - y'_{j,1}}{x'_i - x'_{j,1}}\right], \tag{2.63a}$$

$$v'_{ij} = -\frac{\gamma_j}{4\pi}\ln\frac{\left(x'_i - x'_{j,1}\right)^2 + \left(y'_i - y'_{j,1}\right)^2}{\left(x'_i - x'_{j,2}\right)^2 + \left(y'_i - y'_{j,2}\right)^2} \tag{2.63b}$$

where $(x'_1, y'_1)_j$ and $(x'_2, y'_2)_j$ are the coordinates of the panel start and end points in its local panel coordinate system, as shown in figure 2.13. The transformation of this vector

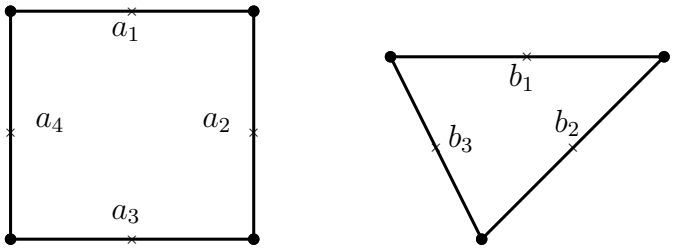


Figure 2.14: Multi-body panel problem: two bodies with different numbers of panels. The figure depicts a square body with 4 panels (a_1, a_2, a_3, a_4), and a triangular body with 3 panels (b_1, b_2, b_3).

(u'_{ij}, v'_{ij}) to the global coordinates is given as,

$$\begin{bmatrix} u_{ij} \\ v_{ij} \end{bmatrix} = \begin{bmatrix} \cos\alpha_j & \sin\alpha_j \\ -\sin\alpha_j & \cos\alpha_j \end{bmatrix} \cdot \begin{bmatrix} u'_{ij} \\ v'_{ij}. \end{bmatrix} \tag{2.64}$$

Note that when $i = j$, the influence coefficient becomes $a_{ij} = -1/2$, represented by the diagonal terms of equation 2.58. If we are dealing with multiple panel bodies (i.e. multiple geometries), as seen in figure 2.14, the panel problem can be solved by constructing a global influence matrix,

$$\underbrace{\begin{pmatrix} c_{a_1a_1} & \cdots & c_{a_1a_N} & c_{a_1b_1} & \cdots & c_{a_1b_M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{a_Na_1} & \cdots & c_{a_Na_N} & c_{a_Nb_1} & \cdots & c_{a_Nb_M} \\ c_{b_1a_1} & \cdots & c_{b_1a_N} & c_{b_1b_1} & \cdots & c_{b_1b_M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{b_Ma_1} & \cdots & c_{b_Ma_N} & c_{b_Mb_1} & \cdots & c_{b_Mb_M} \\ \Delta s_{a_1} & \cdots & \Delta s_{a_N} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \Delta s_{b_1} & \cdots & \Delta s_{b_M} \end{pmatrix}}_{\begin{pmatrix} C_{aa} & C_{ab} \\ C_{ba} & C_{bb} \\ \Delta s_a & 0 \\ 0 & \Delta s_b \end{pmatrix}} \underbrace{\begin{pmatrix} \gamma_{a_1} \\ \vdots \\ \gamma_{a_N} \\ \gamma_{b_1} \\ \vdots \\ \gamma_{b_M} \end{pmatrix}}_{\begin{pmatrix} \gamma_a \\ \gamma_b \end{pmatrix}} = \underbrace{\begin{pmatrix} \mathrm{RHS}_{a_1} \\ \vdots \\ \mathrm{RHS}_{a_N} \\ \mathrm{RHS}_{b_1} \\ \vdots \\ \mathrm{RHS}_{b_M} \\ \Gamma_{\gamma,a} \\ \Gamma_{\gamma,b} \end{pmatrix}}_{\begin{pmatrix} \mathrm{RHS}_a \\ \mathrm{RHS}_b \\ \Gamma_{\gamma,a} \\ \Gamma_{\gamma,b} \end{pmatrix}} \tag{2.65}$$

where the diagonal matrices (C_{aa}, C_{bb}), are the self-induction matrix of the panel body, $a \to a$ and $b \to b$ respectively. The non-diagonal terms (C_{ab}, C_{ba}) are the inter-induction matrix containing the panel influence of $b \to a$ and $a \to b$ respectively. The final two rows of the LHS matrix contains the circulation constraint for each body.

2.6 Validation of Lagrangian method

During the validation study, we are will not be able to validate the vorticity generation of the Lagrangian boundary. This is due to the nature of the hybrid method that we are

using for this project. The generation of the vorticity from the boundary is defined in the Eulerian method, which is then interpolated onto the Lagrangian domain. Therefore, test cases that relies on the generation of the vorticity from the boundary cannot be used to validate our Lagrangian method. Thus the validation study for the Lagrangian method was done separately; first ensuring no-through flow on vortex panels is properly handled; second ensuring that the vorticity of the flow is properly handled by the vortex blobs.

2.6.1 Error analysis of panel method

The validation of the panel method was done by performing a convergence study of a cylinder. The vortex panels where used to simulate a potential flow around a cylinder and the solution of the panels were compared with the analytical solutions.

To test the solution of the vortex panels with the analytical solution, the problem was first ran for the parameters in the table 2.1. The velocity field of the potential flow solution is shown in figure 2.15. The figure shows the magnitude of the velocity, and we see that it shows the velocity field of a potential flow solution, with an infinitely thin boundary layer, stagnating to zero velocity inside the body.

The jagged velocity field around the surface of the cylinder is simply due to the sampling resolution of the field, and for a higher sampling resolution, this will vanish. In order to determine the accuracy of the solution, the velocity field of the panel solution was compared with the analytical solution. The analytical velocity field around a cylinder is given as,

$$u_r = u\left(1 - \frac{R^2}{r^2}\right)\cos\theta, \tag{2.66a}$$

$$u_\theta = -u\left(1 + \frac{R^2}{r^2}\right)\cos\theta, \tag{2.66b}$$

where u_r and u_θ are the radial and the angular velocity respectively. The equation 2.66 is a function of the radius from the center of the cylinder (in our case $r_0 = [0, 0]$) and the radius of the cylinder R.

The velocity field of the vortex panel was compared with this analytical solution along the y-axis from $y = 0 \rightarrow 10$. Figure 2.16a plots the magnitude of analytical velocity $\|\mathbf{u}\|$ and the vortex panel velocity field $\left\|\mathbf{u}^h\right\|$. Comparing the solutions of the plot we see that the solution of the vortex panels and the analytical potential flow solution matches everywhere except at the surface. This happens because the potential flow solution has a slip velocity (i.e non-zero velocity) at the surface of the body, whereas the vortex panels

Table 2.1: Panel study parameters

Parameters	Value	Description
R	1 m	Radius of cylinder
$\mathbf{u}_\infty$	1 m s^{-1}	Free-stream velocity
N_{panels}	100	Number of panels

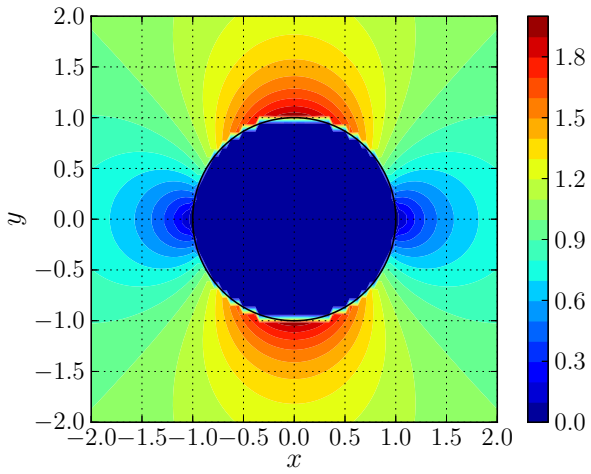
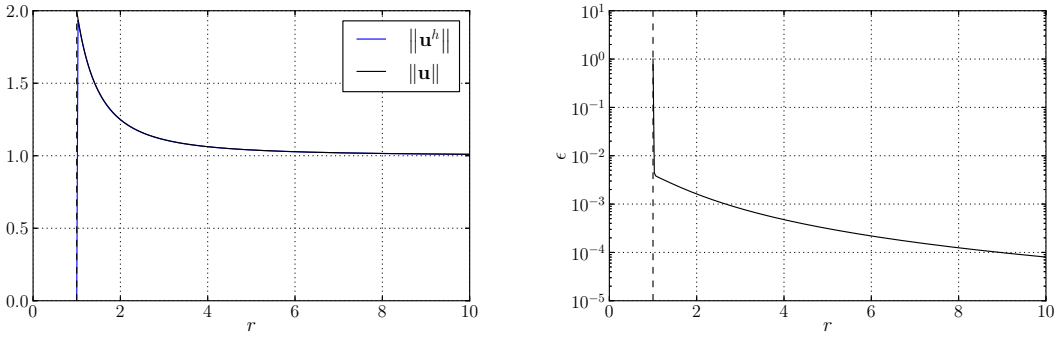


Figure 2.15: Panel method solution: the potential velocity field around a unit cylinder with $R = 1$, $\mathbf{u}_\infty = (1, 0)$, and $N_{\text{panels}} = 100$. The figure depicts the magnitude of velocity field $\|\mathbf{u}\|$, with a zero velocity inside the body.



(a) Comparison of the velocity field.

(b) Error in the velocity field

Figure 2.16: Comparison of the velocity field along the y-axis, $0 \to 10$. Figure **(a)** shows both the solutions, the numerical $\|\mathbf{u}^h\|$ [—, solid blue] and the analytical solution [—, solid black]. Figure **(b)** shows the relative error ϵ in velocity between the solution, given by equation 2.67.

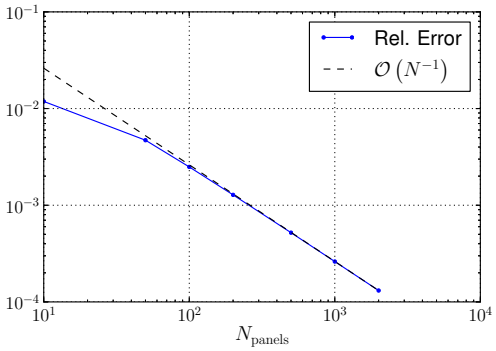


Figure 2.17: Convergence plot of the Constant-Strength Straight Vortex panels. The figures depicts the converges of the relative error ϵ at an $\mathcal{O}\left(N^{-1}\right)$.

solves for a no-slip boundary condition at the collocation points of the surface. This explains the sudden drop of the velocity from $2 \rightarrow 0$ at the surface.

The figure 2.16b shows the relative error ϵ between the numerical and the analytical solution,

$$\epsilon = \frac{\left\|\mathbf{u} - \mathbf{u}^h\right\|}{\|\mathbf{u}\|} \tag{2.67}$$

where $\mathbf{u}$ is the analytical solution and the $\mathbf{u}^h$ is the numerical (panel method) solution. Ignoring the solution right at the surface ($r = R$), we see that the error between the numerical and the analytical solution reduces from $\epsilon = 5 \times 10^{-3} \rightarrow 8 \times 10^{-5}$ as we go from $r = 1 \rightarrow 10$. This behavior of the error tells us that the solution of the constant-strength vortex panels gets more accurate as we go further away from the panels; right next to the panels, we have the largest error. This is because the vortex panels discretizes the body using a first-order approximation (straight panels) and also discretizes the vortex sheet strength using a first-order approximation.

The convergence analysis of the vortex panels was done by determine the error of the vortex panel velocity field w.r.t to the analytical solution for the number of panels $N = 10 \rightarrow 1000$, figure 2.17. The error of the velocity field was determine at $R = 1.5$, and we see that the error converges at an $\mathcal{O}(N)$. This validates that the vortex panel that we have used is a 1^{st} order vortex panel.

2.6.2 Evolution of the vortex blobs

In order to verify and validate the vortex blobs, we simulate the evolution of a Lamb-Oseen vortex. The results of the simulation was used to compare against the analytical results, which we used to determine the accuracy of our vortex blobs.

Lamb-Oseen vortex

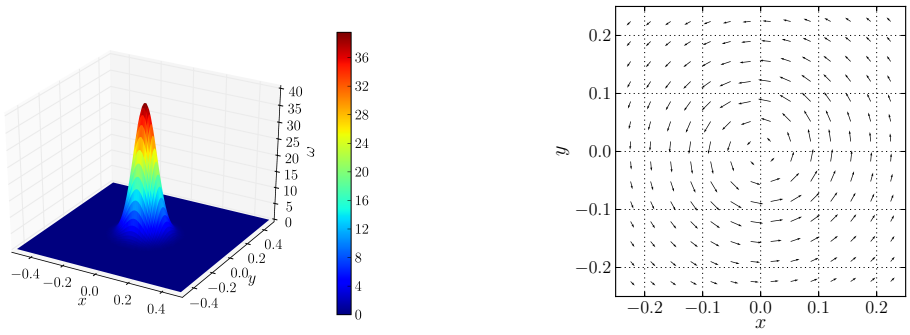
The Lamb-Oseen vortex is a simple solution of unbounded laminar Navier-Stokes equation of a vortex core diffusing as time passes. It was first demonstrate by Lamb and Oseen [61]. The vorticity distribution ω of the core at a given time is defined as,

$$\omega(\mathbf{x}, \tau) = \frac{\Gamma_c}{4\pi\tau} \exp\left(-\frac{r^2}{4\tau}\right), \tag{2.68}$$

and is a function of core strength Γ_c, the scaled viscous time τ, and distance from the core center r. The scaled viscous time is defined as,

$$\tau \equiv \nu t. \tag{2.69}$$

At $\tau = 0$, the Lamb-Oseen vorticity distribution is a Dirac delta function, and the core starts to diffuse and expand as the time progresses. The velocity field of the Lamb-Oseen



(a) Vorticity field **(b)** Velocity Field

Figure 2.18: Lamb-Oseen Vortex problem with $\Gamma_c = 1$, $\tau = 2 \times 10^{-3}$, and $\nu = 5 \times 10^{-4}$. The figure depicts **(a)** the vorticity distribution, and **(b)** the velocity distribution.

vortex is given as,

$$u_\theta = \frac{\Gamma_c}{2\pi r}\left[1 - \exp\left(-\frac{r^2}{4\tau}\right)\right] \tag{2.70a}$$

$$u_r = 0 \tag{2.70b}$$

where u_θ is the circumferential velocity, and u_r, the radial velocity is zero. Figure 2.18 shows the vorticity distribution, and the velocity field of the Lamb-Oseen vortex with a unit core strength $\Gamma_c = 1$, at scaled viscous time $\tau = 2 \times 10^{-3}$. As the time progresses, the core of the Lamb-Oseen vortex will diffuse and we use this analytical solution to valid our vortex blobs.

The lagrangian method ran the case of Lamb-Oseen vortex simulation according to the parameters tabulated in table 2.2. The problem was initialized by discretizing the vorticity field of the Lamb-Oseen vortex with $\Gamma_c = 1$, $\nu = 5 \times 10^{-4}$, and $\tau_0 = 2 \times 10^{-3}$, the initial scaled viscous time. The vorticity field was discretized over the domain $[x, y]$-domain $[-0.5, 0.5]^2$ using vortex blobs with $\sigma = 0.01$, and overlap $= 1$, giving it a fixed blob spacing of $h = 0.01$. We employed the standard initialization method of vortex blobs from section 2.2.4.

However as explained in section 2.2.4, we have to take in account of the gaussian blurring of the original vorticity field due to the initialization process. This posses a problem when evaluating the error between the numerical and the analytical solution. Barba [2], had also encountered the problem before when investigating with a Lamb-Oseen vortex. The solution to the problem was to apply a "time-shift correction", to compensate for the gaussian blurring, solving the problem of this very particular discretization of the diffusion equation. Therefore, this is a special method and this approach can only be used for the Lamb-Oseen vortex problem.

The "time-shift correction" is derived by determining the diffusion effect caused by the discretization of the diffusion equation using the gaussian vortex blobs (with $k = 2$).

Table 2.2: Summary of the parameters for the Lamb-Oseen vortex evolution. Table shows the parameters of Tutty's diffusion method [62]

Parameters	Value	Unit	Description
Γ_c	1	$\mathrm{m^2\,s^{-1}}$	Core strength
Ω	$[-0.5, 0.5]^2$	m	Initial particle domain
$\mathbf{u}_\infty$	$[0, 0]$	$\mathrm{m\,s^{-1}}$	Free-stream velocity
ν	5×10^{-4}	$\mathrm{kg\,s^{-1}\,m^{-1}}$	Kinematic viscosity
(τ_0, τ_f)	2×10^{-3} to 2.5×10^{-3}	$\mathrm{m^2}$	Initial and final scaled viscous time
(t_0, t_f)	4 to 5	s	Initial and final simulation time
$\Delta t_c = \Delta t_d$	0.01	s	Diffusion and convection time step size
$N_{\mathrm{t-steps}}$	100	-	Number of time integration steps
σ	0.01	m	Vortex blob core size
overlap	1	-	Overlap ratio
k	2	-	Gaussian kernel width spreading
f_{redist}	1	-	Redistribution occurance
$f_{\mathrm{popControl}}$	1	-	Population control occurance
$\Gamma_{\mathrm{threshold}}$	$(1 \times 10^{-14}, 1 \times 10^{-14})$	$\mathrm{m^2\,s^{-1}}$	Population control threshold

Barba determined that discretization of the diffusion equation (i.e. the Lamb-Oseen vortex) reconstructs the vorticity field that has been diffused by a time $\sigma^2/2\nu$. So when initializing the particles with a certain strength, we will have to reverse the time by $\sigma^2/2\nu$. The initial particles strengths α_0 of vortex blobs from the Lamb-Oseen vorticity field is given as,

$$\alpha_0 = \omega_0 \cdot h^2 = \left\{ \frac{\Gamma_c}{4\pi\nu\,(t - \sigma^2/2\nu)} \exp\left[-\frac{r^2}{4\nu\,(t - \sigma^2/2\nu)} \right] \right\} \cdot h^2. \tag{2.71}$$

This method was used to investigate the error growth of the vortex blob method. The vortex blobs where convected and diffused according to the parameters in table 2.2. For the primary and the main investigation, we used diffusion method proposed by Tutty [62]. The advantage of this approach is that the we can perform diffusion after every convection step. This makes the method less prone to time integration error and eliminates any discontinuous behaviour in the evolution. We will see that when coupling the Lagrangian method and Eulerian method, discontinuity in the problem introduces additional errors.

So, using the Tutty's diffusion method (the simple redistribution scheme, section 2.4.1), we convect and diffuse the vortex blobs with a $\Delta t_c = \Delta t_d = 0.01$. The time integration of the problem is done using a 4^{th}-order Runge-Kutta method (RK4), ensuring accurate convection of the blobs. After the convection process, we will have to treat the Lagrangian distortion of the blobs using the remeshing scheme discussed in section 2.3.1. Generally, the remeshing is typically done every 10 iterations [2]. However, as our diffusion scheme and hybrid method requires structured lattice of vortex blobs for efficient calculations, we will remesh after every step, $f_{\mathrm{redist}} = 1$.

In additional, after the evolution process, we can perform a population control on the vortex blobs to keep the simulation as efficient as possible. The Population Control (PC)

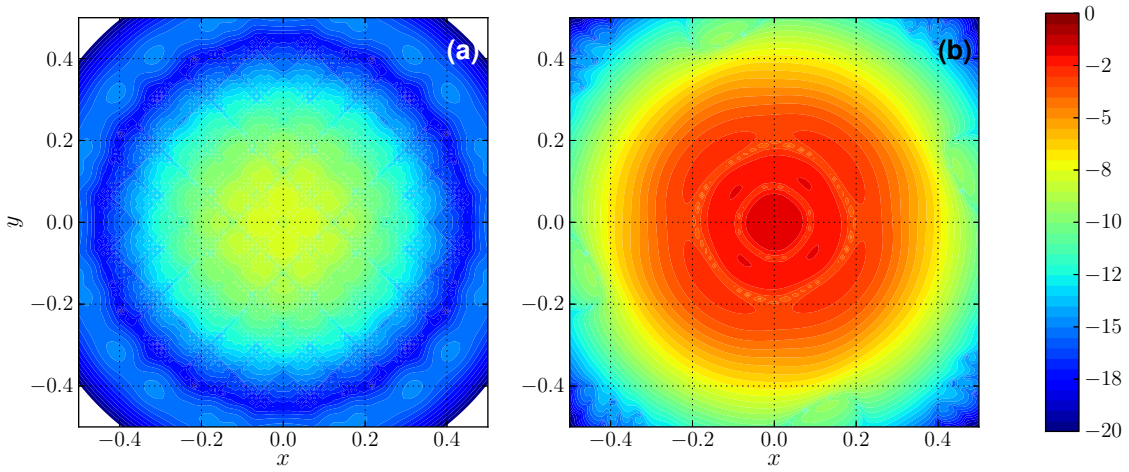


Figure 2.19: Relative error growth of Lamb-Oseen vorticity during the evolution (in logarithmic scale). The figure shows **(a)**, the initial relative error at $t_0 = 4$, and **(b)** the final relative error in vorticity at $t_f = 5$.

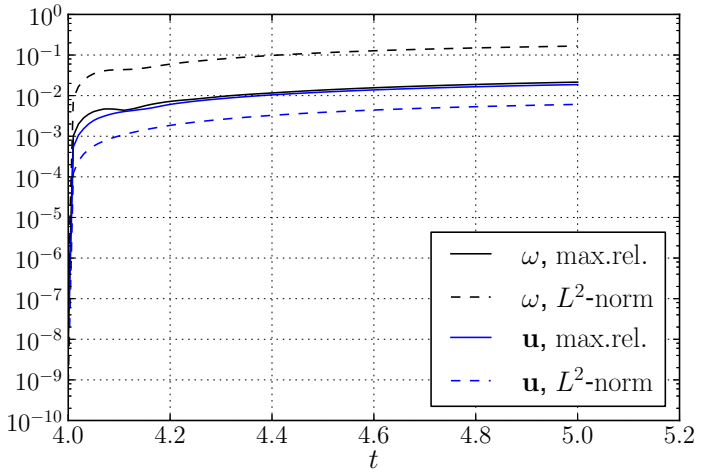


Figure 2.20: Relative error growth of Lamb-Oseen vortex during the evolution from $t_0 = 4$ to $t_f = 5$. Figure depicts the error in vorticity: maximum relative error [—, solid black], and error in $L^2 - \text{norm}$ [- -, dashed black]; and error in velocity: maximum relative error [—, solid blue], and error in $L^2 - \text{norm}$ [- -, dashed blue].

removes vortex blobs that have very small circulation strengths. After the diffusion and remeshing, we will be left with vortex blobs with strengths close to numerical precision, as they have minimal impact on the accuracy of the vorticity field, we can removed them. When performing the population control, we to ensure that the loss of total circulation is at minimal. We place a tolerance on the conservation of circulation, and from literature [2], we see that a cap of $\Gamma_{\text{threshold}} = 1 \times 10^{-14}$ is acceptable.

To verify whether our Lagrangian scheme is function according to theory, we evaluated the error growth of the simulation. Figure 2.19, shows the initial and the final relative error in vorticity. We see that initially we have a maximum relative error around 10^{-8}, located at the center of the Lamb-Oseen core. After 100 time integration steps from $t_0 = 4$ to $t_f = 5$, we see that the maximum relative error in vorticity increases from 10^{-8} to 10^{-2}. The error of the vorticity are predominantly localized at the center of the core. This is because, this is where we have the highest gradient in the vorticity, as seen from figure 2.18a. Therefore, with the current vortex blob discretization, we are not able to reconstruct this large gradient of the vorticity field.

Figure 2.20, shows the error growth of the Lamb-Oseen vortex from $t_0 = 4$ to $t_f = 5$. The figure plots the error growth of the vorticity and the velocity. Due to the relation of the vorticity and the velocity, the error of the vorticity is an order higher that the error in the velocity. The figure shows the both the maximum relative error, and the error in the $L^2 -$ norm. The maximum relative error (e.g. for vorticity), is determined as,

$$\left\| \omega^{\text{exact}} - \omega^{\text{discrete}} \right\|_{\infty} = \frac{\max\{\left| \omega^{\text{exact}} - \omega^{\text{discrete}} \right|\}}{\max\{|\omega^{\text{exact}}|\}}, \tag{2.72}$$

where ω^{exact} is analytical vorticity field, and ω^{discrete} is the numerical vorticity field from the vortex blobs. The error in the $L^2 -$ norm of the vorticity is calculated as

$$\left\| \omega^{\text{exact}} - \omega^{\text{discrete}} \right\|_{2} = \left(\sum_{i}^{N} \left| \omega^{\text{exact}} - \omega^{\text{discrete}} \right|^2 \cdot h^2 \right)^{\frac{1}{2}}, \tag{2.73}$$

and the error in velocity is calculated using the same principle. Investigating the plot, we see that after the first iteration, there is sudden increase in the error, but as time progresses the error growth stabilizes. From literature, we see that this trend has also been observed by Barba [2] and Speck [58]. For comparison, we used similar parameters, and we observe that the sudden jump in error has been similar to the literature.

Comparison of diffusion schemes: SRS vs. MRS

To observe how both the diffusion scheme compare, we ran test cases with both diffusion schemes. From the simulation, we were able to observe that the Tutty's diffusion scheme, SRS, performed better that Wee's diffusion scheme, MRS. Figure 2.21 shows the error growth of maximum relative error in vorticity for both diffusion schemes. The solid lines represents the solution of the MRS and the dashed lines shows the SRS. The convection and the diffusion time steps where controlled by modifying the blob spacing h, and keeping the convection time step size $\Delta t_c = 0.01$.

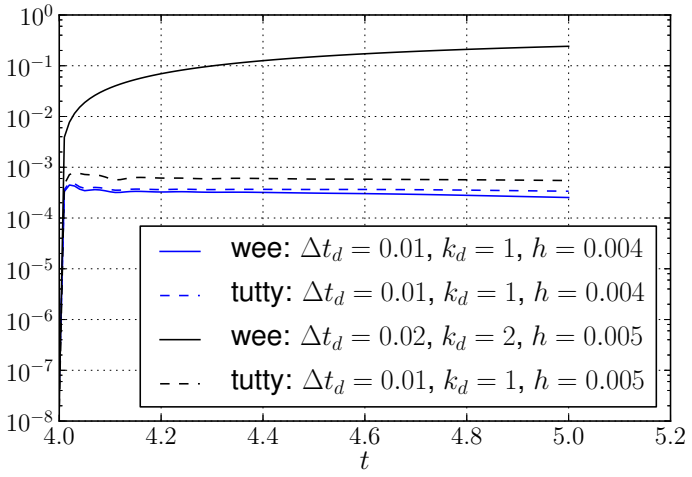


Figure 2.21: Comparison of Tutty's, simple redistribution scheme and Wee's modified interpolation method for treating diffusion. Figure depicts the growth in maximum relative error in vorticity from $t_0 = 4$ to $t_f = 5$ at $\Delta t_c = 0.01$. The Wee diffusion scheme with $\Delta t_d = \Delta t_c = 0.01$ [—, solid blue], and $\Delta t_d = 2\Delta t_c = 0.02$ [—, solid black]. The Tutty's diffusion scheme, $c^2 = 1/3$, with $\Delta t_d = \Delta t_c = 0.01$ [- -, dashed blue], and $\Delta t_d = \Delta t_c = 0.02$ [- -, dashed black].

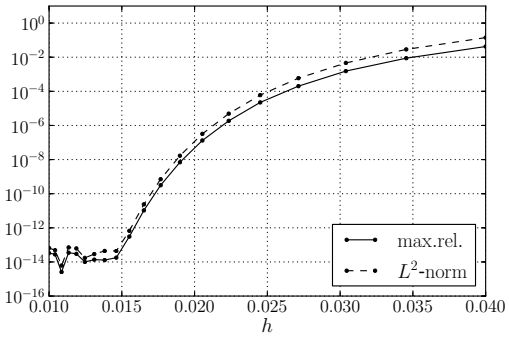
At $h = 4 \times 10^{-3}$, both schemes have the diffusion time step $\Delta t_d = 0.01$ meaning that the diffusion occurs at every iteration $k_d = 1$. During this time we see that the MRS performs slightly better that the SRS scheme. This is because, the diffusion of the vortex blobs is directly incorporated into the remeshing process, whereas the Tutty's SRS segregates the diffusion and the remeshing into two subsequent process.

At $h = 5 \times 10^{-3}$, the diffusion constraint on the Wee's MRS changes the minimum allowable diffusion time step to $\Delta t_d = 0.02$, meaning that diffusion has to be performs at every second step, $k_d = 2$. Whereas for Tutty's SRS, we are not constraint by the minimum diffusion time step as so we can perform diffusion at $k_d = 1$. When investigating the growth in error for MRS, we see that the diffusion scheme has a large increase in the error, whereas the SRS does not have this limitation and is still able to diffuse at every step and has only a slight increase in error, figure 2.21 (solid black line). The diffusion scheme over-estimates the diffusion, and results in an incorrect diffusion process. However, we see that Tutty's SRS still performs a stable diffusion (dashed black line).

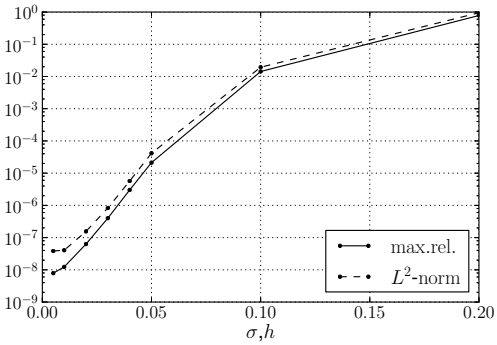
This verification of the diffusion scheme showed that if Wee's MRS diffuses at $k_d > 1$, it will over-estimate the diffusion of the vorticity. Therefore, when using this scheme we will have to ensure that $k_d = 1$, by adjusting the convective time step appropriately. However, for large Reynolds number flows this is not possible, and so we must rely on the more stable and versatile Tutty's SRS diffusion scheme.

2.6.3 Convergence study of the viscous vortex method

Finally, we can perform a converge study, to validate that our scheme works according to the theory. For a scheme that is numerically stable, the error due to discretization must converge as the resolution of the discretization increases.



(a) Error in vorticity vs. h with $\sigma = 0.02$



(b) Error in vorticity vs. h, σ with $\text{overlap} = 1$.

Figure 2.22: Convergence in spatial discretization of the vortex blobs. Figure **(a)** shows the convergence by fixing the core size σ and **(b)** shows the convergence when overlap ratio is fixed.

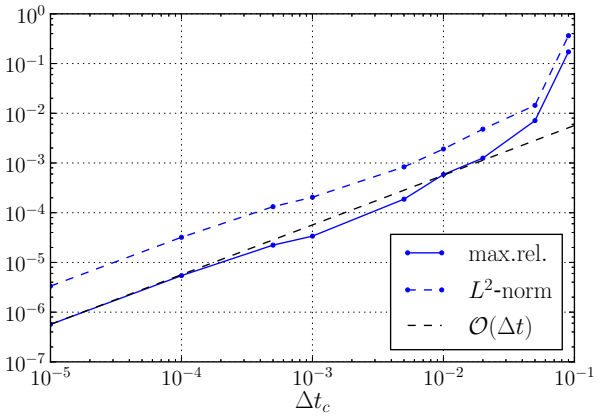


Figure 2.23: Error growth of Lamb-Oseen vorticity field after one-step.

First, we investigate the convergence in spatial discretization. As we are dealing with vortex blobs, there are multiple ways of increasing the resolution. The straight forward method would be to increase the density of particles in a given area, i.e. reducing the blob spacing h and maintaining the core spreading σ. Figure 2.22a shows the convergence of the spatial discretization when the core size σ is maintained at $\sigma = 0.02$. At this case, the overlap ratio changes with the blob spacing, described by equation 2.17. At small blob spacing h, the error in vorticity quickly drops to near machine precision. When investigating the order of converges, we that the error converges in a non-linear fashion and similar results have been obtained by Barba [2].

Figure 2.22b, shows the convergence of error when the overlap ratio is fixed, overlap = 1. In this test case, the core size scaled with the blob spacing, $h = \sigma$, and when increasing the spatial resolution, the error converges at non-linear fashion.

To investigate the convergence in temporal discretization, we determined the growth in Lamb-Oseen vorticity field after convecting with various convection step size Δt_c. Note that to perform this convergence study accurately, we had to rely on the Tutty's SRS diffusion scheme, so that $\Delta t_d = \Delta t_c$. Figure 2.23 shows the convergence of the error for various temporal discretization. With the combined convection and diffusion scheme, we see that error converges at $\mathcal{O}(\Delta t)$, meaning that the vortex blobs evolution is 1^{st}-order in time.

2.7 Summary of the Lagrangian method

In summary, we have investigated the Lagrangian domain of our hybrid method in this chapter. The Lagrangian method was used to described the evolution of the wake past the geometry. Vortex Particle Method (VPM) was an ideal choice to describe the wake, as we require only to evolve the wake, and the generation of the vorticity is dealt with in the Eulerian domain. Unlike the Eulerian method, VPM only required the fluid elements where there was vorticity, meaning that the VPM was inherently auto-adaptive. Using the Population Control method, we were able to remove vortex blobs where they were not needed. Furthermore, the computation of the these elements were accelerated using an FMM, and simultaneously was parallelized using a GPU hardware.

For the VPM, the choice of fluid elements was vortex blobs. This had non-zero core size, removing the singularity when performing Biot-Savart calculations. The strengths of these particles were initialized by assigning the local circulation strength to the particle, given by equation 2.18. When we perform coupling, we will see that the gaussian blurring of the original vorticity field during the initialization is the fundamental source of error. Strategies such as Beale's iterative method, cannot be used as it is defined for an unbounded domain. The only approach to minimize the gaussian spreading initialization error was to increase the overlap ratio to $overlap = 1$, and minimize the blob spacing h as much as possible. So, the proper handling of the initialization of the vortex blob strength is still an open question, and recommend that if solved can significantly improve the accuracy and the efficiency of the hybrid coupling. More on this will be discussed in section **??**. !!! citation !!!

The convection of the vortex blobs was done by using a 4^{th}-order Runge-Kutta time integration method. However, due to high strains in the fluid, we had to deal with the

Lagrangian grid distortion of the vortex blob lattice. To deal with this, we used a M_4' interpolation kernel that remeshed the particles onto a structured grid.

The diffusion of the vortex blobs was also import when simulation viscous flows. Initially, we employed the modified interpolation kernel by Wee [65], which integrated the diffusion process into the standard interpolation kernel. However, the method posed two problems: the diffusion time step Δt_d had a lower limit; the scheme over-estimated the diffusion when the diffusion did not occur at every step, $k_d > 1$. To overcome this problem, we used the simple redistribution scheme by Tutty [62], which enabled us to perform diffusion after every convection step. This also ensured that the diffusion process was continuous, which was important when performing the coupling algorithm.

To deal the body boundary in the flow, we used the vortex panels to enforce the no-through/no-slip boundary condition. Koumoutsakos, Leonard, and Pepin [40], have shown that enforcing no-slip boundary condition is equivalent to a no-through boundary condition, as they are linked. Constant-Strength Vortex panels, based of Katz [36], were used to treat the body boundary condition. This panel method was then verified and validated with the analytical solution of a potential flow around a cylinder. There was one difference between our vortex panel method and the standard boundary element method that is used in a typical VPM. Our panel method cannot be used to determine the vorticity generation at the body, as this is done in our Eulerian method. So we validated our Lagrangian method separately; first ensuring that the no-through boundary condition is satisfied at the vortex panels, second ensuring that the vorticity of the flow is handled properly by the vortex blobs.

To validate the vortex blob, we used the analytical solution of the Lamb-Oseen vortex problem. We verified that the Wee's MRS diffusion method indeed did not perform optimally for all parameters. The validation was done by investigating the error in the vorticity and the velocity field and was able to conclude that the vortex blobs performed according to literature of Barba [2].

Lagrangian method algorithm

The flowchart to one time step of the Lagrangian method is given by figure 2.24.The algorithm to the Lagrangian method can be summarized as follows:

1. **Solve for panels:** Determine the strengths of the vortex panels γ, such that the no-slip boundary condition at the collocation points of the vortex panels is enforced. When determining the strengths, we also have to ensure that the total circulation of the vortex panels satisfied the conservation of circulation, equation 2.50.

2. **Evaluate the total velocity field:** Evaluate the total velocity field $\mathbf{u}$, which is the sum of velocity field induced by the vortex blobs $\mathbf{u}_\omega$, the velocity field induced by the vortex panels $\mathbf{u}_\gamma$, and the free-stream velocity field $\mathbf{u}_\infty$.

3. **Convect the vortex blobs:** Use the velocity field to time step the vortex blobs from t_n to t_{n+1} to the new position with a convection time step Δt_c.

4. **Remesh the Lagrangian field:** Remesh the vortex blobs onto a structured square lattices using the M_4' interpolation kernel.

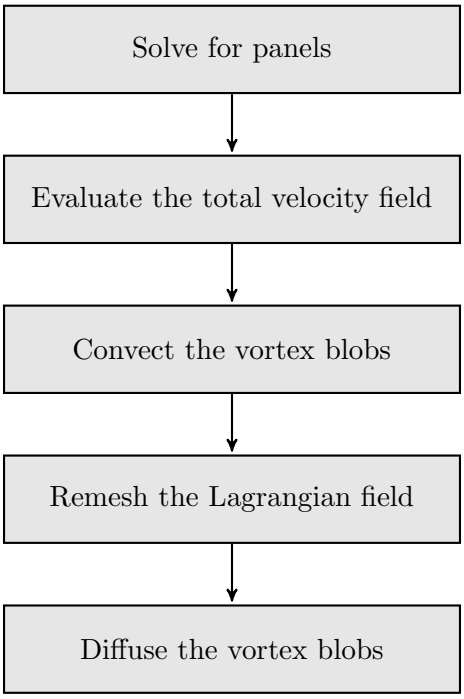


Figure 2.24: Flowchart of the Lagrangian method. The flowchart shows coupling between vortex panels and vortex blobs to evolve from t_n to t_{n+1}.

5. **Diffuse the vortex blobs:** Diffuse the vortex blobs using the Δt_d diffusion time step, by modifying the strengths of the vortex blobs according to Wee's MRS or Tutty's SRS method.

The generation of the vorticity is dealt with in the Eulerian domain, which is explained in chapter 3. The vorticity is then transfered into the Lagrangian domain using the Hybrid coupling scheme, which was summarized in the introduction, chapter 1, and full elaborated in chapter 4. s

2.8 Chapter Nomenclature

Latin Symbols

$\mathbf{A}$	Vortex panel influence matrix	-
c^2	Diffusion parameter	-
$\mathcal{E}$	Enstrophy	$\mathrm{m}^2\,\mathrm{s}^{-2}$
h	Nominal particle spacing	m
h_ν	Characteristic diffusion distance	m
k_d	Frequency of vortex blob diffusion	-
$\mathbf{K}$	Biot-Savart kernel	-
$\mathbf{K}_\sigma$	Vortex blob kernel	-

$\hat{\mathbf{n}}$	Unit normal vector	-
N	Number of vortex blobs (particles)	-
Ov	Overlap ratio	-
p	Pressure	Pa
r	Radial position	m
$\hat{\mathbf{s}}$	Unit tangent vector	-
t	Simulation time	s
$\mathbf{u}$	Velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_b$	Velocity of the body	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_\gamma$	Vortex sheet induced velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_{\text{ext}}$	External induced velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}^h$	Discrete velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_\infty$	Free-stream velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_\phi$	Free-stream velocity	$\mathrm{m\,s^{-1}}$
u_r	Radial velocity	$\mathrm{m\,s^{-1}}$
u_θ	Angular velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_{\text{slip}}$	Boundary slip velocity	$\mathrm{m\,s^{-1}}$
$\mathbf{u}_\omega$	Vorticity velocity	$\mathrm{m\,s^{-1}}$
W	Interpolation kernel weight	-
$\mathbf{x}$	Position vector	m
$\mathbf{x}_\nu$	Position vector of particle to be diffused	m
$\mathbf{x}_p$	Position vector of vortex blob (particle)	m

Greek Symbols

α_p	Circulation of the particle	$\mathrm{m^2\,s^{-1}}$
β_p	Corrected circulation of the particle	$\mathrm{m^2\,s^{-1}}$
Δt_c	Convection time step size	s
Δt_d	Diffusion time step size	s
ϵ	Relative error	-
Γ	Circulation	$\mathrm{m^2\,s^{-1}}$
Γ_b	Circulation inside a moving body	$\mathrm{m^2\,s^{-1}}$
γ_c	Vortex sheet strength	$\mathrm{m^2\,s^{-1}}$
γ_γ	Circulation of the vortex sheet	$\mathrm{m^2\,s^{-1}}$
γ_ω	Circulation of the fluid	$\mathrm{m^2\,s^{-1}}$
ν	Kinematic viscosity	$\mathrm{m^2\,s^{-1}}$
ω	Vorticity	$\mathrm{s^{-1}}$
$\tilde{\omega}$	Vortex blob cell vorticity	$\mathrm{s^{-1}}$
ω^h	Discrete vorticity field	$\mathrm{s^{-1}}$
ρ	Density	$\mathrm{kg\,m^{-3}}$

σ	Core size	m
τ	Scaled viscous time	m^2
ξ	Scale relative position of particle to stencil node	-
ζ_σ	Smooth cut-off function of the blobs	-

Chapter 3

Eulerian Domain: Finite Element Method

Standard Computation Fluid Dynamics (CFD) method discretizes the fluid into smaller regions, known as grids, and solves the set of Navier-Stokes equations in this region. This type of formulation is referred to as an Eulerian method, as we are evaluating the change of flow property in a given volume.

For the hybrid method, we use the Navier-Stokes grid formulation in the near-body region. The advantage of using the Eulerian method at this region is that it is much more efficient in resolving the boundary layer than the Vortex Particle Method. We can directly enforce the wall boundary condition at the wall boundary of the Eulerian domain, solving the problem of vorticity generation of the body. In the hybrid coupling strategy, we can then interpolate this newly resolved near-wall solution on to the Lagrangian domain, where the vortex blobs can efficiently evolve the particles.

The various approaches to solve the fluid dynamics problem from a Eulerian reference frame. Finite Volume Method (FVM), Finite Difference Method (FDM), and Finite Element Method (FEM) are the common choice for solving the Navier-Stokes problem and differ by the way they approach to solve the problem. FVM divides the domain into volumes where it enforces the conservation of mass and momentum in each sub-domains. FDM divides the domain into nodes and use local Taylor expansions to approximate the partial differential equations. FEM divides the domain into elements and solves the problem using variational calculus. So in the end, the choice of Eulerian method does not have a direct impact on the coupling with the Lagrangian method as the purpose of the Eulerian method is only to efficiently, and accurately resolve the near-body region of the body.

We have decided to use the FEM packages provided by the FEniCS project as they have be already implemented efficient, multi-threaded algorithms for solving the partial differential equation. Furthermore, they provide extensive features for future developments such as adaptive mesh refinement, fluid-structure interaction, and efficient computation of turbulent flow.

3.1 Introduction to Finite Element Method

Finite Element Method (FEM) is numerical method to solve for the solution of a given partial differential equation. It is solved by describing it as a variation problem, giving us an approximate solution for the boundary value problem [7]. So the FEM approximates the unknown variables and converts the partial differential equations to a set of algebraic equations, which makes them easier to solve. It was traditionally used for solid mechanics (e.g for the analysis of aircraft structures [50]), but have since then used to solve fluid dynamics problems [31] [34] [32].

Finite element discretization

The finite element solves problem by dividing the domain of interest into smaller, simpler regions known as "elements". These "elements" are connected at the joints which are called nodes or nodal points. We use these sets of node and elements to represent the actual variation in the field (such as the displacement, the velocity, the pressure or the temperature) using simple functions, known as the basis functions. Thus, we have transformed the domain of interest into finite number of Degrees of Freedom (DOF). We combine the set of equations of the element into a global system of equations to solve for the unknown.

A finite element discretization in 2-D can be seen in figure 3.1. The figures shows two connected elements, where the cells represent the area of the element, and the vertices of the cell represents the nodes of the element. The finite number of cells $\mathcal{T}_h = \{T\}$ of the fluid domain Ω, together makes the mesh of the Eulerian domain. As shown in figure 3.1, the cells of the finite element in 2-D, are made of simple geometrical shapes such as triangles or quadrilaterals. There are two approaches to discretize the domain: structured or unstructured mesh. The structured mesh has cells oriented in a structured pattern, and is the simplest approach in discretizing the mesh. The advantage of such a discretization is that it is possible to make a simple data structure which can be used to perform efficient computation. The downside to such discretization is that the mesh quality deteriorates as one increases the complexity of the domain. However, the FEM enables us to perform an unstructured discretization of the domain, as shown in figure 3.2. The figure shows

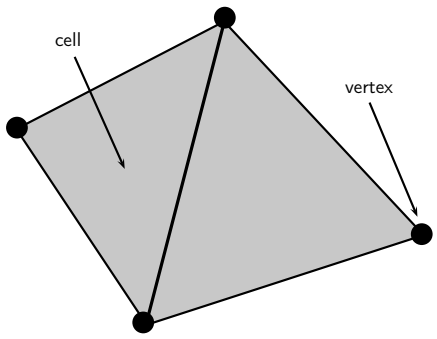
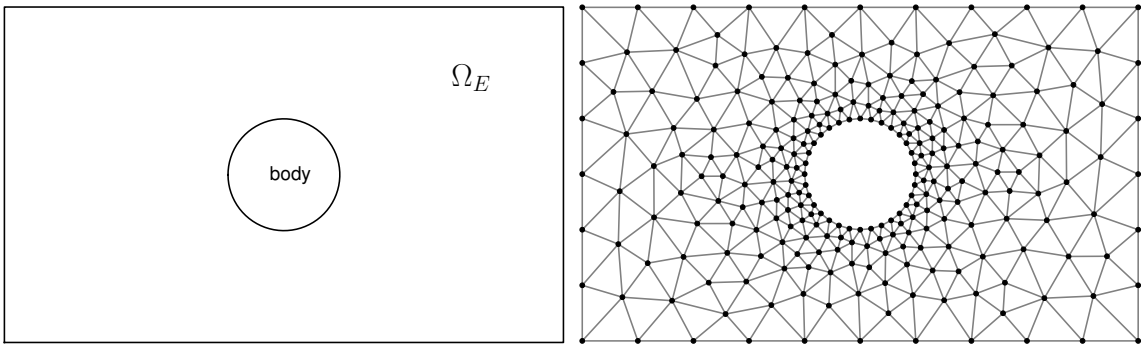


Figure 3.1: A two-dimensional finite element geometry. The cell represents the area of the element, and vertices are the edges of the cell.



(a) Fluid domain Ω_E around the cylinder **(b)** Delaunay triangulation of the fluid

Figure 3.2: Delaunay triangulation of the fluid around a cylinder resulting in unstructured mesh with controllable cell sizes.

the unstructured discretization of the fluid domain around the cylinder Ω_E, connecting the rectangular outer boundary of the fluid to the circular no-slip boundary of the body in a simple fashion. This shows that even though the unstructured method formulation is more complicated that the structured formulation, we have the advantage that the mesh quality does not deteriorate as the domain becomes more complex.

There are several algorithms for mesh generation. The standard approach is to employ the Delaunay triangulation method derived from the Voronoi diagram concept [9]. This divides the domain into a set of triangles, as shown in figure 3.2. This type of mesh generation allows us to connect different shapes of boundary with each other. Furthermore, this triangulation method be controlled by predefining the boundary element nodes using a transfinite interpolation.

Finite element function and function space

The finite element is defined using a triple $(T, \mathcal{V}, \mathcal{L})$, as defined Ciarlet [14] and used by the FEniCS Project [44]. The domain Ω is divided into cells T, the space $\mathcal{V} = \mathcal{V}(T)$ is a finite dimensional function space on T of dimension n, and $\mathcal{L} = \{\ell_1, \ell_2, ..., \ell_n\}$ is the set of degrees of freedom forming the basis for the dual space $\mathcal{V}'$ of $\mathcal{V}$.

When the domain Ω is divided into cells T, we can define the function and the function space of the finite element problem. For each cell, a local function space $\mathcal{V}$ can be defined to collectively construct the global function space V. Any given function $u \in V$ is a expressed in a linear combination of basis functions $\{\phi_1, \phi_2, ..., \phi_N\}$, of the function space V,

$$u(x) = \sum_{j=1}^{N} U_j \phi_j(x). \tag{3.1}$$

There are several types of finite element families: the Brezzi-Douglas-Marini, the Crouzeiz-Raviart, the Discontinuous Lagrange, the Hermite, and the Lagrange elements [44]. Each has its own advantage such as the Discontinous Lagrange, or Discontinous Galerkin (DG) element consists of discontinuous functions, which was originally introduced for solving hyperbolic problem by Reed and Hill in 1973 [51]. The method was able to conserve

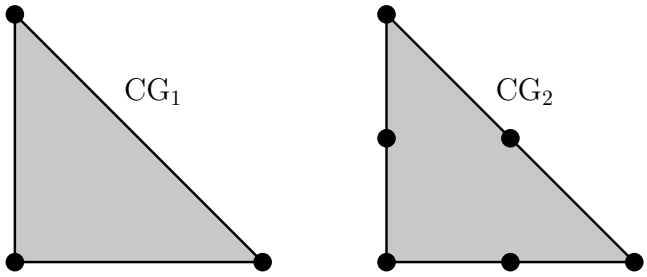


Figure 3.3: The Lagrange CG_q triangle for $q = 1, 2$. The triangles have 3 and 6 DOFs respectively (•, black dot).

mass at each element, had a high-order accuracy, and was robust in solving the advection problem. However for the current problem, we will rely on the Lagrange elements, also known as the Continuous Galerkin (CG), which are based on the Lagrange polynomials [11]. These elements are widely used and are the simplest to implement for our project.

Lagrange elements belong to the space H^1, which a Sobolev space containing functions u such that u^2 and $|\nabla u|^2$ have finite integral in the domain Ω [44]. The Lagrange element uses point evaluation for the degrees of the freedom, where a DOF in (x, y) denotes the point evaluation of the function u, $\ell(u) = u(x, y)$. We can have a Lagrange elements of various orders $q = 1, 2, ...$, where q is the degree of the Lagrange polynomial $\mathcal{P}_q$ on the domain at T. For the 2-D case, the dimension n of the finite element is given as,

$$n(q) = \frac{1}{2}(q + 1)(q + 2). \tag{3.2}$$

For $q = 1$, we have a simple Lagrange element CG_1, known as the Courant triange [22], with 3 DOFs. For a higher order finite element, we can set $q = 2$, giving us a Lagrange element CG_2 with 6 DOFs per cell. Figure 3.3 shows the two Lagrange triangles CG_1 and CG_2 for $q = 1$ and $q = 2$ respectively. The Courant triangle has the DOFs located at the vertices of the cell, and the higher order CG_2 has 3 additional DOFs, all located midway between the vertices. To describe our Eulerian problem of our hybrid scheme, we will rely on the CG_1 and CG_2 Lagrange elements.

Variational formulation

To solve a basic problem such as a Poisson equation numerically, we need to convert it into a variational problem. The methodology is followed from the FEniCS tutorial provide by Langtangen [44]. A 1D Poisson problem is given as,

$$\begin{aligned} -\nabla^2 u(x) &= f(x), \qquad x \text{ in } \Omega, \\ u(x) &= u_0(x), \qquad x \text{ on } \partial\Omega. \end{aligned} \tag{3.3}$$

We can transform equation 3.3 into a variational form by multiplying it with a test function v, and integrating it over the domain Ω,

$$-\int_\Omega \left(\nabla^2 u\right) v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x, \qquad \forall \, v \in \hat{V}. \tag{3.4}$$

In variational form equation 3.4, the function u is known as the trial function, and is what we are trying to approximate. The trial function u lies in the trial function space V, and the test function v lies in the test function space $\hat{V}$. When performing integration by parts, the test function v is required to be zero at regions where u is known. So, the additional terms cancel and we get,

$$-\int_\Omega \nabla u \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x \qquad \forall \; v \in \hat{V} \tag{3.5}$$

This form is referred to as the "weak-form" of the original Poisson equation and is valid for all v in the trial space $\hat{V}$. An inner product of any two function f and g in domain Ω is defined as,

$$\langle f, g \rangle = \int_\Omega f g \, \mathrm{d}x, \tag{3.6}$$

so we can simplify equation 3.5 to,

$$\langle \nabla u, \nabla v \rangle = \langle f, v \rangle, \qquad \forall \; v \in \hat{V}. \tag{3.7}$$

In order to solve this continuous problem numerically, we must transform it into a discrete variational problem,

$$-\langle \nabla u_h, \nabla v \rangle = \langle f, v \rangle \qquad \forall \; v \in \hat{V}_h \subset \hat{V}, \tag{3.8}$$

where u_h is the discrete function in the discrete space V_h which is a subset of V, and the discrete function space $\hat{V}_h$ is a subset of $\hat{V}$. A common choice for the function space is the linear triangular element (Courant triangle) that has three nodes, as shown in figure 3.3, where $\hat{V}_h$ and V_h are described by piecewise linear functions of the triangle. At the boundary, the functions in the test space is zero, whereas the functions in the trial space is equal the boundary condition u_0. The equation 3.8 can be simplified as,

$$a\,(u, v) = L(v), \tag{3.9}$$

where,

$$a\,(u, v) = -\langle \nabla u, \nabla v \rangle, \tag{3.10}$$

and

$$L(v) = \langle f, v \rangle. \tag{3.11}$$

The variable $a(u, v)$ and $L(v)$ is the denoted as the bilinear and linear form, respectively. For simplicity, we will ignore the discrete notation (i.e $\{\cdot\}_h \rightarrow \{\cdot\}$). To solve for the discrete solution we substitute,

$$u = \sum_{j=1}^{N} U_j \phi_j, \tag{3.12}$$

the linear combination of the basis function ϕ_j, spanning the function space V, into $a\,(u, v)$. The test function is a linear combination of the basis functions $\hat{\phi}_i$, spanning the test space $\hat{V}$, defined as,

$$v = \sum_{i=1}^{N} \hat{\phi}_i. \tag{3.13}$$

The test function v is taken to be zero at the boundary and one everywhere else. Substituting equation 3.12 and 3.13 into equation 3.9 gives,

$$\sum_{j=1}^{N} a(\phi, \hat{\phi}_i)\ U_j = L(\hat{\phi}_i). \tag{3.14}$$

Thus, we can solve the linear system of equations of form,

$$\mathbf{A}U = b, \tag{3.15}$$

where $\mathbf{A}_{ij} = a(\phi_j, \hat{\phi}_i)$ contains the coefficients, and the Right-Hand Side (RHS) b contains the knowns of the problem.

3.2 Solving the Finite Element problem

To solve this linear system of equations, equation 3.15, we used DOLFIN , the finite element library of the FENICS Project. This library uses high performance linear algebra kernels, and provide a scripting interface in PYTHON for computational experience, ease similar to the MATLAB interface. Such environment helps us to focus on the development of the theory (i.e the high-level algorithms). In order to generate the mesh of the fluid domain, we used GMSH, a three-dimensional finite element mesh generator which proves a fast, light and user-friendly meshing tools.

3.2.1 Introduction to FEniCS Project

The FENICS Project is a collaborative work of various universities, that developed tools to perform automated finite element algorithms, which can be used to solve partial differential equation. It was a project originated in 2003 with the research collaboration of University of Chicago and Chalmers University of Technology with Logg, Mardal, and Wells [44]. Since then, it has been expanded to various institutes such as Royal Institute of Technology, Simula Research Laboratory, University of Cambridge, and Delft University of Technology.

The consists of various libraries such as UFC, UFL, FIAT, INSTANT and mainly DOLFIN. DOLFIN is the core library aimed at automating the solution of partial differential equations using finite element method [45]. It uses automated code generation thus maintaining high-level mathematical expressions but still providing efficient, multi-threaded performance (with Message Passing Interface (MPI)) internally.

We used the DOLFIN library wrapped in PYTHON to solve the finite element problem. For example, we can demonstrate the procedures of solving the Poisson problem, equation 3.3. We can take $f = 2$ with the boundary conditions,

$$u(x) = u_0(x) = \sin x \cdot \cos y, \tag{3.16}$$

at $\partial\Omega$. The finite element code generation is automated with DOLFIN, leaving only the explicit expression of the problem in python, see source code listing 3.1.

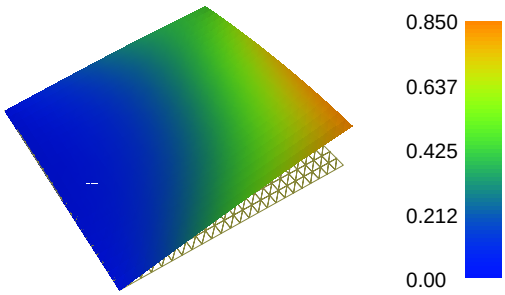


Figure 3.4: DOLFIN VTK plot of the Poisson solution, given by the problem, source code listing 3.1.

```
1  from dolfin import *
2
3  # Generate unit square mesh: 24 × 24
4  mesh = UnitSquareMesh(24, 24)
5
6  # Define Function space: 1st order, Continuous-Galerkin
7  V = FunctionSpace(mesh,"CG",1)
8
9  # Define boundary conditions
10 # u_0 = sin x · cos y
11 u0 = Expression("sin(x[0])*cos(x[1])")
12
13 def u0_boundary(x, on_boundary):
14     return on_boundary
15
16 # Define the boundary condition
17 # u(x) = u_0(x), x on ∂Ω
18 bc = DirichletBC(V, u0, u0_boundary)
19
20 # Define the variational problem
21 u = TrialFunction(V)  # Trial function
22 v = TestFunction(V)   # Test function
23 f = Constant(2.)      # f = 2
24 a = -inner(nabla_grad(u), nabla_grad(v))*dx # LHS: a = −∫ ∇u∇v dx
25 L = f*v*dx            # RHS: L = ∫ fv dx
26
27 # Solve the Poisson problem
28 u = Function(V)       # Define the solution
29 solve(a == L, u, bc)  # a(u, v) = L(v)
30
31 # Plot the result
32 plot(u)
```

Listing 3.1: A complete program for solving the Poisson problem and plotting the solution. The Poisson problem is given as $-\nabla^2 u = f$, where $u_0 = \sin x \cdot \cos y$ on the boundary and $f = 2$. The code is written in PYTHON using DOLFIN 1.2 library

3.2.2 Mesh generation using GMSH

The generation of the mesh is achieved by GMSH, an open-source software developed by Geuzaine & Remacle [28], which has implemented a user-friendly interface and fast algorithms. The GMSH implemented kernels that use BLAS and LAPACK linear algebra packages in C++ for fast computation. Furthermore, it allows for scriptability making it ideal to integrate it with our current PYTHON code project for future automation.

3.3 Solving Incompressible Navier-Stokes Equations

Using the DOLFIN library for constructing the finite element problem, we can now solve the Eulerian method of our hybrid scheme. The Eulerian method will formulated the problem use the primitive variables velocity-pressure $\mathbf{u} - p$, which we can use to directly enforce the no-slip velocity boundary condition at the wall of the body.

3.3.1 Velocity-pressure formulation

The velocity-pressure $\mathbf{u} - p$ formulation of the fluid, is the standard formulation of the Navier-Stokes equations of the fluid dynamics problem. The 2-D incompressible Navier-Stokes equations of a fluid with unit density (i.e $\rho = 1$) is given as,

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \sigma = f, \tag{3.17a}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3.17b}$$

where σ is the Cauchy stress tensor defined as,

$$\sigma(\mathbf{u}, p) = 2\nu\epsilon(\mathbf{u}) - p\mathbf{I}. \tag{3.18}$$

The Cauchy stress tensor is a function of pressure p, the fluid kinematic viscosity ν, and the symmetric gradient ϵ defined as,

$$\epsilon(\mathbf{u}) = \frac{1}{2}\left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathrm{T}}\right). \tag{3.19}$$

describing the stresses in fluid due to the velocity gradient and the pressure. The incompressible 2-D Navier-Stokes equations have two unknowns, the vector velocity field $\mathbf{u}$, that lies on the vector-valued function space V, and the scalar pressure field p, which lies on the scalar-valued function space Q. Onces we solve these we can determine the vorticity field, which we will transfer to the Lagrangian domain.

3.3.2 Determining the vorticity field

The coupling between the Eulerian to Lagrangian method is through the transfer of the vorticity field ω from the Eulerian domain to the Lagrangian vortex blobs. The vorticity field ω, is defined as,

$$\omega = \nabla \times u, \tag{3.20}$$

```
1  # Define the trial and test function
2  omega = TrialFunction(W)
3  v = TestFunction(W)
4
5  # Define the variation problem for vorticity
6  a = inner(omega,v)*dx    #  ⟨ω, v⟩
7  b = inner(curl(u),v)*dx  #  ⟨∇×u, v⟩
8
9  # Pre-Assemble the LHS
10 A = assemble(a)
11
12 ...
13
14 # During the time-stepping
15 omega = Function(W) # Define the function
16 B = assemble(b)     # Assemble b
17 solve(A, omega.vector(), B) # Solve for vorticity
```

Listing 3.2: The PYTHON implementation of the vorticity calculation

and is defined as the curl of the velocity field $\mathbf{u}$, which lies on the scalar-valued function space W. Due to the constant change in the velocity field, we have to recalculate the vorticity at every time step $t_n, t_{n+1}, \ldots$. To solve this problem in a efficient manner, we can use the `assemble` function of DOLFIN to pre-construct the problem. We must first define the equation 3.20 in the variational (integral) form,

$$\int_\Omega \omega \cdot \mathbf{v} \, \mathrm{d}x = \int_\Omega (\nabla \times \mathbf{u},) \cdot \mathbf{v} \, \mathrm{d}x, \tag{3.21}$$

where $\omega = \sum_{j=1}^{N} \hat{\omega}_j \psi_j$, is a linear combination of basis function ψ_j, spanning the function space W. The variational form is summarized as,

$$a(\omega, \mathbf{v}) = L(\mathbf{u}, \mathbf{v}) \tag{3.22}$$

where $a(\omega, v)$ contains the knowns of the problem, which are fixed during the simulation. This can be pre-calculated to optimize the problem. $(\mathbf{u}, \mathbf{v})$ is the unknown of the problem which has to be recalculated every time as it is a function of the current velocity. The PYTHON implementation of the algorithm is show in listing 3.2. Using the DOLFIN library, we can used the `assemble` function to pre-calculated the LHS of the problem (line 10). So using the algorithms of the hybrid coupling scheme, we can transfer this vorticity field of the Eulerian domain on the vortex blobs.

3.3.3 Taylor-Hood finite element family for solving ICNS

To solve the Incompressible Navier-Stokes (ICNS) problem, we must choose an appropriate finite element function spaces for the velocity $\mathbf{u}$, the pressure p, and the vorticity ω, by ensuring that we satisfy the Ladyzhenskaya-Babuška-Brezzi (LBB) compatibility condition, also known as the inf-sup compatibility condition [8]. The Lagrange finite element spaces must have the order of velocity q_{vel}, one order higher than the order of the

Table 3.1: Summary of the Lagrange element CG_q of order q, that was used for solving the incompressible Navier-Stokes problem. The variable names of the function space, the trial functions, and the test functions are tabulated together.

Variable	Finite element	Function space	Trial function	Test function
Velocity	CG_2	V	$\mathbf{u}$	$\mathbf{v}$
Pressure	CG_1	Q	p	q
Vorticity	CG_1	X	w	x

pressure q_{pres},

$$q_{\text{vel}} = q_{\text{pres}} + 1. \tag{3.23}$$

Brezzi and Fortin [8] showed that if both are of same order, it will result to an unstable problem. To solve the ICNS problem, we will used the Taylor-Hood family [60], examined by Boffi [5]. The method use velocity order $q_{\text{vel}} = 2$ and pressure order $q_{\text{pres}} = 1$. We decided to choose this method, as it is the most conventional method, that is simple, and that shows a stable behavior.

In addition, we have to choose an appropriate function space for the vorticity. As vorticity is the curl of the velocity, to reduce interpolation error during the projection of the solution, we will used function space one order lower than the velocity, $q_{\text{vort}} = 1$. Table 3.1 shows the list of the function spaces, the finite element type and their orders. In additional, we have included the variable names of the function space, trial functions and the test functions, associated to the function element that we have chosen for the problem.

3.3.4 Incremental pressure correction scheme

The algorithm to solve the NS problem was first demonstrated by Chorin in 1968 [12], referred to as the Chorin's projection method or sometimes known as the non-incremental pressure correction scheme. The process relied on first computing a tentative velocity by initially neglecting the pressure in the momentum equation of the Navier-Stokes problem, equation 3.17. The velocity field is corrected by determining the pressure field satisfying a divergence free vector field. This method however does not satisfy the discrete incompressibility constraint exactly and so, Goda in 1979 [29], introduced an improved Incremental Pressure Correction Scheme (IPCS). The method computed the viscous term at the incremented time $(t_{n-1} + t_n)/2$, and used the stress formulation to determine the corrected pressure [44]. The detailed algorithms to the IPCS scheme, as demonstrated by the FEniCS Project [44], can be summarized as follows:

1. **Compute the tentative velocity:** The tentative velocity $\mathbf{u}^\star$ is determined by solving,

$$\begin{aligned}\langle D_t^n \mathbf{u}^\star, \mathbf{v}\rangle + \langle \mathbf{u}^{n-1} \cdot \nabla \mathbf{u}^{n-1}, \mathbf{v}\rangle + \langle \sigma(\mathbf{u}^{n-\frac{1}{2}}, p^{n-1}), \epsilon(\mathbf{v})\rangle \\ + \langle p^{n-1}\hat{\mathbf{n}}, \mathbf{v}\rangle_{\partial\Omega} - \langle \mathbf{v}\ \hat{\mathbf{n}} \cdot (\nabla \mathbf{u}^{n-\frac{1}{2}})^{\mathrm{T}}, \mathbf{v}\rangle_{\partial\Omega} = \langle f^n, \mathbf{v}\rangle,\end{aligned} \tag{3.24}$$

is valid for all $\mathbf{v} \in V$, where $\mathbf{u}^{n-\frac{1}{2}}$ is defined as,

$$\mathbf{u}^{n-\frac{1}{2}} = \frac{\mathbf{u}^{\star} + \mathbf{u}^{n-1}}{2}, \tag{3.25}$$

With the Dirichlet velocity boundary conditions at the boundary $\partial\Omega$, we can solve the equation 3.24. The additional term,

$$\langle \mathbf{v}\ \hat{\mathbf{n}} \cdot (\nabla \mathbf{u}^{n-\frac{1}{2}})^{\mathrm{T}}, \mathbf{v} \rangle_{\partial\Omega}, \tag{3.26}$$

is resulted from the integration by parts, when we evaluate the viscous term at $(t_{n-1} + t_n)/2$ and we use the stress formulation instead of the Laplacian formulation as done for the Chorin scheme. This difference ensures that the velocity profile at the inlet and the outlet of the domain is more accurate that the ones obtained for the Chorin scheme.

The source code for solving the tentative velocity problem is shown in listing 3.3. First, we pre-define all the terms need for the tentative velocity problem formulation (lines 3 to 16). We can also pre-assemble the LHS of the problem (line 19) outside of the time-integration loop, as it remains constant. During the time integration, we first assemble the RHS of the problem (line 26), then apply the Dirichlet velocity boundary condition (line 29) which consist of the wall boundary condition, and external Dirichlet velocity boundary condition (e.g. the free-stream). Finally, we can solve the problem using GMRES solver for solving the system of linear equations.

2. **Determine the corrected pressure:** The corrected pressure p^n is determined by solving,

$$\langle \nabla p^n, \nabla q \rangle = \langle \nabla p^{n-1}, \nabla q \rangle - \langle \nabla \cdot \mathbf{u}^{\star}, q \rangle / \Delta t_n \tag{3.27}$$

valid for all $q \in Q$. We use the previously calculated tentative velocity $\mathbf{u}^{\star}$ to determine the corrected pressure. We can solve the problem using the Neumann pressure boundary condition at the pressure outlet of the domain. We define a boundary as the pressure outlet, if we do not know the velocity boundary condition at that boundary. This is true for the region were the exit flow is perturbed. However, for the coupled Eulerian method (that we will use), all the boundary conditions are available as a velocity boundary condition from the Lagrangian domain. This means that do not have to assume any pressure boundary condition.

The source code for solving the corrected pressure problem is shown in listing 3.4. As done for the tentative velocity, we can formulate and pre-assemble the problems before the time loop. In the time loop, we only need to assemble the RHS, apply the boundary condition (if it exists), and finally solve for the corrected pressure. Using the corrected pressure, we can determine the corrected velocity field.

3. **Determine the corrected velocity:** The corrected velocity field u^n is determined by solving,

$$\langle \mathbf{u}^n, \mathbf{v} \rangle = \langle \mathbf{u}^{\star}, \mathbf{v} \rangle - \Delta t_n \langle \nabla (p^n - p^{n-1}), \mathbf{v} \rangle, \tag{3.28}$$

which is valid for all $\mathbf{v} \in V$. We correct the tentative velocity $\mathbf{u}^{\star}$ by the pressure difference to determine the correct velocity field. We will have to apply the Dirichlet velocity boundary condition at the boundary again, to solve for the problem.

```
# Before the time-stepping:

# Define: u^{n-1/2} = (u* + u^{n-1})/2
U  = 0.5*(u0 + u)

# Formulate the tentative velocity problem
F1 = (1/k)*inner(v, u - u0)*dx \
             + inner(v, grad(u0)*u0)*dx \
             + inner(epsilon(v), sigma(U, p0, nu))*dx \
             + inner(v, p0*n)*ds \
             - beta*nu*inner(grad(U).T*n,v)*ds \
             - inner(v, f)*dx

# Extract the LHS, and the RHS
a1 = lhs(F1)
L1 = rhs(F1)

# Pre-assemble the LHS
A1 = assemble(a1)

...

# During the time-stepping:

# Assemble the RHS
b = assemble(L1)

# Apply the Dirichlet velocity boundary condition b.c
[bc.apply(A1, b) for bc in bcVelocity]

# Solve for the Tentative velocity
solve(A1, u1.vector(), b, "gmres", "default")
```

Listing 3.3: The source code for solving the tentative velocity $\mathbf{u}^{\star}$, using the equation 3.24

The source code of the solving the corrected pressure problem in shown in listing 3.5. We first initialize the problem, by formulating the problem and assembling the LHS outside the time loop (line 3 to 8). In the time integration loop, we assemble the RHS, apply the velocity boundary condition and finally solve for the corrected velocity field.

This algorithm was implemented using DOLFIN's Krylov GMRES solver with an absolute and a relative error tolerance of 10^{-25} and 10^{-12} respectively. The program structure was based on the collection of benchmark and solvers provided by the FENICS examples scripts [43]. The algorithm described above an explicit time marching scheme, Forward Euler (FE), the simplest time marching scheme. Therefore, for the time marching scheme to be stable, we require the CFL number to satisfy the following condition:

$$\mathrm{CFL} = \Delta t_{\max} \frac{\|\mathbf{u}\|_{\max}(\nu + \Delta h_{\min}\|\mathbf{u}\|_{\max})}{\Delta h_{\min}^2} \leqslant 1. \tag{3.29}$$

This gives us the direct constraint on the maximum Eulerian time step size $\Delta t_{E,\max}$ which is function of the CFL number, maximum fluid velocity in the Eulerian domain $\|\mathbf{u}\|_{\max}$, the fluid viscosity ν and the minimum mesh cell size Δh_{min}.

```
# Before the time-stepping:

# Formulate the pressure correction problem
a2 = inner(grad(q), grad(p))*dx          # ⟨∇q, ∇p^n⟩
L2 = inner(grad(q), grad(p0))*dx\        # ⟨∇ q, ∇ p^{n-1} ⟩ − ⟨ ∇ · u*, q⟩/Δ t_n
          - (1/k)*q*div(u1)*dx

# Pre-assemble the LHS
A2 = assemble(a2)

...

# During the time-stepping:

# Assemble the RHS
b = assemble(L2)

# Apply the Dirichlet velocity boundary condition b.c
if len(bcPressure) == 0: normalize(b)
[bc.apply(A2, b) for bc in bcPressure]

# Solve for the corrected pressure
solve(A2, p1.vector(), b)
if len(bcPressure) == 0: normalize(p1.vector())
```

Listing 3.4: The source code for solving the corrected pressure p^n using the equation 3.27

```
# Before the time-stepping:

# Formulate the velocity correction problem
a3 = inner(v, u)*dx      # ⟨u^n, v
L3 = inner(v, u1)*dx - k*inner(v, grad(p1 - p0))*dx # ⟨u*, v − Δt_n⟨∇(p^n − p^{n-1}), v⟩

# Pre-assemble the LHS
A3 = assemble(a3)

...

# During the time-stepping:

# Assemble the RHS
b = assemble(L3)

# Apply the Dirichlet velocity boundary condition b.c
[bc.apply(A3, b) for bc in bcVelocity]

# Solve for the corrected pressure
solve(A3, u1.vector(), b, "gmres", 'default')
```

Listing 3.5: The source code for solving the corrected pressure p^n using the equation 3.27

```
...

def epsilon(u):
    "Returns symmetric gradient"
    return 0.5*(grad(u) + grad(u).T)

def sigma(u,p,nu):
    "Returns stress tensor"
    return 2*nu*epsilon(u) - p*Identity(u.cell().d)

# Define the normal function
n = FacetNormal(mesh)

# Define the unit vectors
eX = Constant((1.0, 0.0))
eY = Constant((0.0, 1.0))

# Define the line integrator
ds = Measure("ds")[boundaryDomains]
noSlip = 2 # No-slip boundary identification = 2

# Determine the forces
# Integrate the forces over the boundaryDomain == noSlip
L = assemble(inner(inner(sigma(u,p,nu), n), eY)*ds[noSlip]) # Lift
D = assemble(inner(inner(sigma(u,p,nu), n), eY)*ds[noSlip]) # Drag
```

Listing 3.6: The PYTHON implementation of the force calculation

3.3.5 Determining the body forces

After we determine the flow fields, we can perform comparisons on the lift and the drag generated by the body. To determine there parameters, we first need to determine the friction and pressure forces acting on the no-slip boundary, which can be determined from the stress tensor σ acting on the surface of the body. The stress tensor σ is given by,

$$\sigma(\mathbf{u}, p) = 2\nu\epsilon(\mathbf{u}) - p\mathbf{I}, \tag{3.30}$$

where ϵ is the symmetric gradient, equation 3.19, and is a function of the velocity $\mathbf{u}$ and the pressure p acting on the surface.The lift coefficient and the drag coefficient is computed as,

$$L = \int_{\partial\Omega} [\sigma(\mathbf{u}, p) \cdot \hat{\mathbf{n}}] \cdot \hat{\mathbf{e}}_y \, \mathrm{d}s, \tag{3.31a}$$

$$D = \int_{\partial\Omega} [\sigma(\mathbf{u}, p) \cdot \hat{\mathbf{n}}] \cdot \hat{\mathbf{e}}_x \, \mathrm{d}s, \tag{3.31b}$$

where $\hat{\mathbf{e}}_x$ and $\hat{\mathbf{e}}_y$ are the 2D unit Cartesian vectors,

$$\hat{\mathbf{e}}_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \hat{\mathbf{e}}_y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3.32}$$

The lift coefficient and the drag coefficient, C_l and C_d respectively, is the lift and drag normalized with the dynamics pressure and reference length c (in 2D), where the lift

perpendicular to the free-steam and the drag is tangential to it,

$$C_l = \frac{L}{\frac{1}{2}\|\mathbf{u}\|_\infty^2 c}, \qquad C_d = \frac{D}{\frac{1}{2}\|\mathbf{u}\|_\infty^2 c}. \tag{3.33}$$

3.4 Validation of eulerian method

To validate our Eulerian method, we will first investigate the problem of the Lamb-Oseen vortex. Then we will compare the results of the Clercx-Bruneau dipole collision at $Re = 625$. Finally, we will investigate the problem of the Impulsively started cylinder at $Re = 550$, which we can use to validate the evolution of lift and drag.

3.4.1 Lamb-Oseen Vortex

The Lamb-Oseen vortex is an analytical solution by Lamb and Oseen, describing the diffusion of a vortex core [61]. We solved the same problem as the one described in the Lagrangian validation problem 2.6.2.

Problem Definition

In the Lagrangian method, the Lamb-Oseen vortex was initialized using the vorticity field as the vortex blobs carry circulation strengths. However, Eulerian domain use the primitive variables $\mathbf{u} - p$ for formulating the problem. Therefore, we use the Lamb-Oseen velocity field as the initial conditions for the problem. The velocity field is given as,

$$u_\theta = \frac{\Gamma_c}{2\pi r}\left[1 - \exp\left(-\frac{r^2}{4\tau}\right)\right] \tag{3.34a}$$

$$u_r = 0, \tag{3.34b}$$

where Γ_c is the vortex core strength, $\tau \equiv \nu t$ is the scaled viscous time, and r is the distance from the core center. The parameters of the simulation is tabulated in table 3.2. To ease the comparison of the Eulerian to the Lagrangian method, we performed ensure similar spatial resolution. The figure 3.5 shows the domain of the problem, discretized the domain $\Omega = [-1, 1]^2$ in a structure grid with the number of finite element cells $N_{\text{cells}} = 200^2$ in x and y direction, minimum cell size $h = \sqrt{2}/100$.

Furthermore, the figure 3.5 also shows the boundary domains $\partial\Omega$ of the fluid domain. For the Lamb-Oseen problem, as we have the analytical solution of the velocity field for all time, we can use this solution to prescribe the external domain boundary condition. So for this problem, we only need an external Dirichlet velocity boundary condition, at the boundary domain identified as, $\text{ID}_{\text{ext}} = 3$. This would imply that we do not need to explicitly apply the pressure boundary condition, as we already have a velocity boundary condition. With all the boundary conditions, we can evolve the initial velocity distribution of the Lamb-Oseen vortex from $t_0 = 4$ to $t_f = 5$, using the IPCS algorithm described in section 3.3.4.

Table 3.2: Summary of the parameters for the Lamb-Oseen vortex evolution.

Parameters	Value	Unit	Description
Γ_c	1	$\mathrm{m^2\,s^{-1}}$	Core strength
Ω	$[-1,1]^2$	m	Eulerian domain bounds
$\mathbf{u}_\infty$	$[0,0]$	$\mathrm{m\,s^{-1}}$	Free-stream velocity
ν	5×10^{-4}	$\mathrm{kg\,s^{-1}\,m^{-1}}$	Kinematic viscosity
(τ_0,τ_f)	2×10^{-3} to 2.5×10^{-3}	$\mathrm{m^2}$	Initial and final scaled viscous time
(t_0,t_f)	4 to 5	s	Initial and final simulation time
$h_{\min}$	$\frac{1}{100}\sqrt{2}$	m	Minimum mesh cell size
N_{cells}	200^2	-	Number of mesh cells
CFL	0.95	-	CFL number
$\|\mathbf{u}\|_{\max}$	1.5	$\mathrm{m\,s^{-1}}$	Maximum magnitude fo the velocity
Δt	0.001	s	Time step size
$N_{\text{t-steps}}$	1000	-	Number of time integration steps
ID_{fluid}	1	-	Fluid domain I.D
ID_{ext}	3	-	External Dirichlet velocity boundary I.D

We used CFL stability condition equation 3.29, to determine the time step size, $\Delta t = 0.001$. The Eulerian method time steps using a Forward Euler (FE) time marching and requires $N_{\text{t-steps}} = 1000$ time steps. During the evolution, we evaluated the growth of the error in velocity, and in vorticity between the numerical results and the analytical solution.

Results

We are interested in the evolution of error in vorticity, as this is the quantity which will be interpolated onto the Lagrangian domain. Figure 3.6 shows the initial and the final relative error in vorticity over the Eulerian domain. Opposed to the Lagrangian results, figure 2.19, we see that initial relative error in the vorticity field is larger. This is so because, the Lagrangian domain was initialized using the vorticity, where as the Eulerian domain was initialized using the velocity. To calculate the vorticity on the Eulerian domain, we had to project the curl the velocity onto the function space of vorticity W. This process of initialization in the finite element domain and project of the vorticity introduced additional numerical error. However, the pattern of the relative error in vorticity, is similar to the Lagrangian solution, with highest error at the core center, where we have the highest gradients in vorticity.

As the time progress, we see that the error of the problem is stable and does not increase as observed for the Lagrangian domain, figure 3.6b. The growth of the relative error in velocity and vorticity can be observed in figure 3.7. It shows that during the evolution of the Lamb-Oseen vortex, the relative error in velocity, and vorticity is stable. We see that due to the relation of vorticity to velocity, the error in vorticity is higher that velocity.

To determine the convergence of space, the simulation was run for $h \approx 0.25$ to $h \approx 5\times10^{-3}$. Figure 3.8a shows the convergence of the relative error in vorticity. This

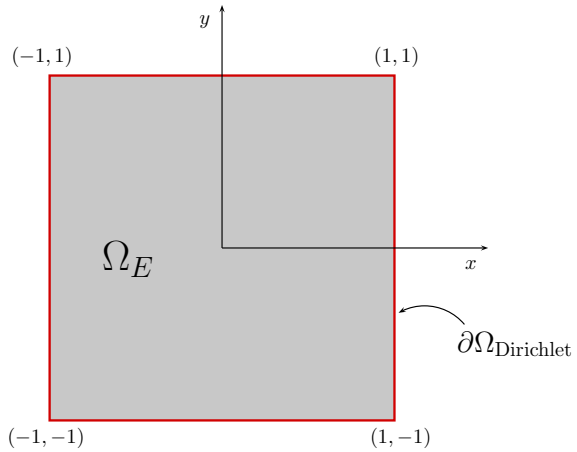


Figure 3.5: Eulerian domain for the Lamb-Oseen vortex problem. Figure shows the bound of the domain $\Omega = [-1, 1]^2$, identified as $\text{ID}_{\text{fluid}} = 1$; and the boundary domain $\partial\Omega$ [—, solid red], identified as $\text{ID}_{\text{ext}} = 3$, which is where the Dirichlet velocity boundary condition was applied.

validates that the scheme is 2^{nd}-order in space, due to second order function space CG_2 for the primitive variable, velocity.

To determine convergence in time, we ran the simulation with various time steps $\Delta t = 5 \times 10^{-3}$ to $\Delta t = 1 \times 10^{-4}$). As we performed the investigation, we saw that the error in primitive variable $\mathbf{u}$, converged at an order 1, figure 3.8. This is true to the theory, as we are employing a 1^{st}-order Forward Euler scheme for the time integration. Thus, we have verified with the analytical solution of the Lamb-Oseen vortex that our Eulerian method is implemented according to the theory, and perform in a robust manner.

3.4.2 Clercx-Bruneau dipole collison at $Re = 625$

The Eulerian method that we have developed here is to be used a wall-bounded Eulerian solver that can highly resolve the vorticity production of the boundary for the Hybrid method. Therefore it is vital that the vortex interaction with the no-slip boundary is handled properly.

To determine the proper handling of the no-slip boundary, it is common practice to use a simple test of dipole colliding with the wall. In this test cases, one could observe how the no-slip boundary handles the incoming vortex and can be used to determine if the system is formulated appropriately. Ould-salihi et al. [49] used this case to validate their Hybrid method that couples vortex particles with finite-difference method. Cottet et al. [20] used the collision to tool to validate the vortex method. Therefore, we decided to use the Clercx-Bruneau dipole collision is a test case from Clercx & Bruneau [15], where they performed a numerical study of a normal collision of a dipole with a no-slip boundary. This experiment provide extensive benchmark results for various Reynolds numbers with a Chevyshev pseudo-spectral numerical method.

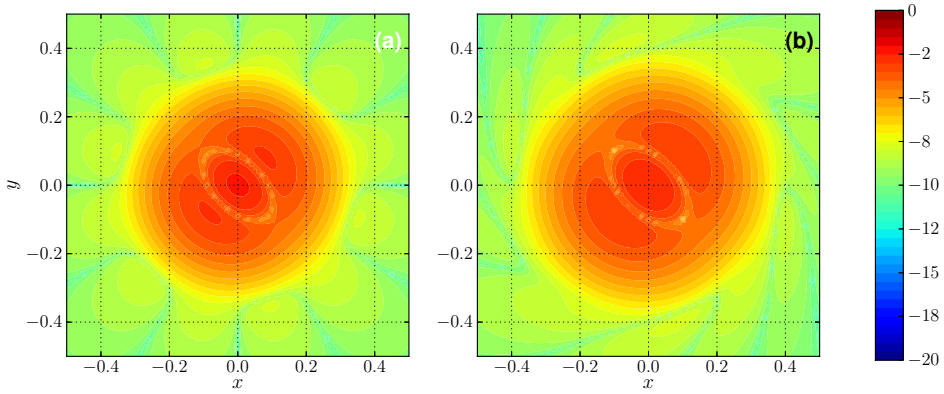


Figure 3.6: Relative error in vorticity field in logarithmic scale. Figure **(a)** shows the initial relative error in vorticity at $t = t_0$, and figure **(b)** shows the relative error in vorticity at the end of the time stepping $t = t_f$.

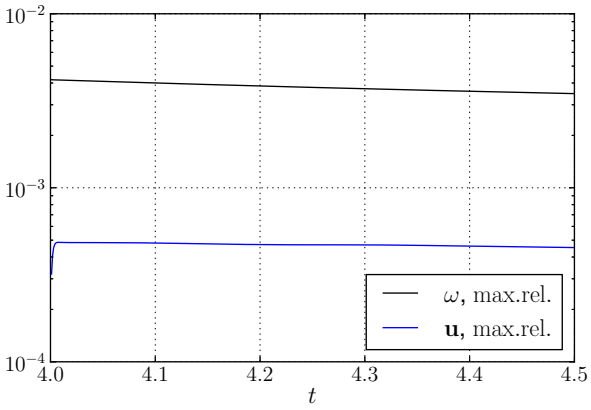
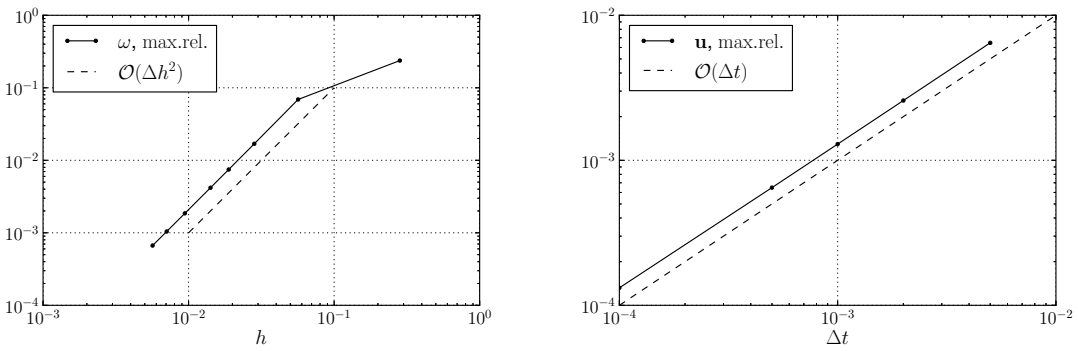
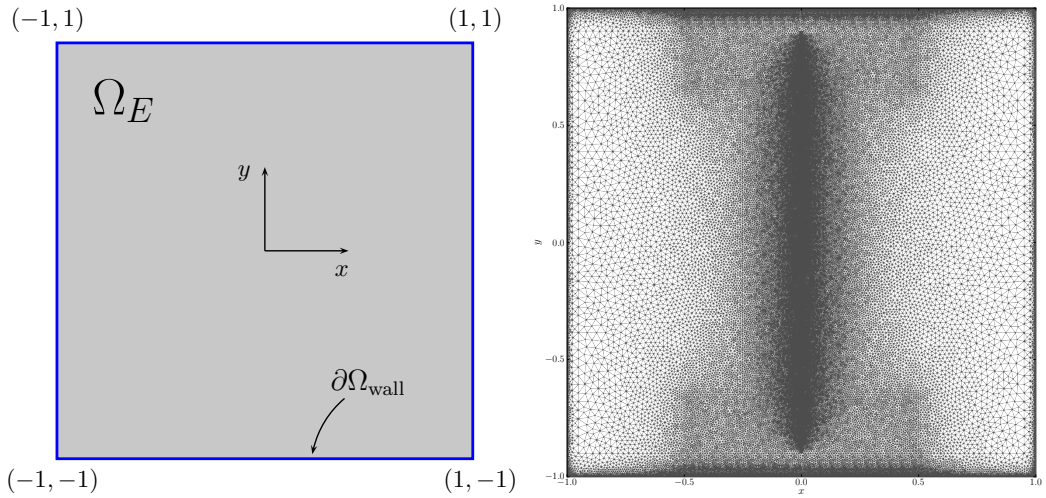


Figure 3.7: Evolution of the maximum relative errors from $t_0 = 4$ to $t_f = 4.5$. The figure depicts maximum relative error in velocity [—, solid blue] and the maximum relative error in vorticity [—, solid black].



(a) Spatial convergence **(b)** Temporal convergence

Figure 3.8: Convergence in space and time. The figure depicts **(a)** convergence in space of $\mathcal{O}(\Delta h^2)$ and **(b)** convergence in time of $\mathcal{O}(\Delta t)$.



(a) Domain definition **(b)** Domain mesh

Figure 3.9: Domain of the Clercx-Bruneau dipole collision problem. The figure depicts **(a)** the definition of the domain with the fluid domain [gray] and the no-slip boundary [blue]; and **(b)** the unstructured mesh of the domain with $N_{\text{vert}} = 48k$.

Problem Definition

Unlike other dipole test cases, Clercx & Bruneau provide well-defined initial and boundary conditions for the dipole vorticity field. Furthermore, they used a vorticity distribution that was continuous, which ensures a smooth velocity field for our Eulerian method using the $\mathbf{u} - p$ formulation. The literature provided results for the collision that we are interested: a normal collision with the dipole traveling perpendicular to the wall.

For this research, we decided to use the simpler case of normal collision at $Re = 625$, where Re is the integral-scale Reynolds number defined as,

$$Re = \frac{UW}{\nu}, \tag{3.35}$$

where U is the characteristic velocity of the flow, W is half width of domain, and ν is the kinematic viscosity. We require a low Reynolds number as the Eulerian method solves an incompressible laminar. The domain Ω of the problem is square with bounds $\Omega = [-1, 1]^2$, as shown in figure 3.9a. The problem is defined in a closed box, where the Eulerian domain is enclosed in a no-slip boundary $\partial\Omega_{\text{wall}}$ where dipole will collide and interact.

The initial conditions of the Clercx-Bruneau dipole is a smooth vorticity distribution which a positive monopole at $(x_1, y_1) = (0.1, 0)$ and the negative monopole at $(x_2, y_2) = (-0.1, 0)$, with each having a core radius $R = 0.1$. The vorticity distribution of the combined monopole is given as,

$$\begin{aligned} \omega(\mathbf{x}, 0) = \quad & \hat{\omega}_1 \left[1 - \left(\frac{r_1}{R}\right)^2\right] \exp\left\{-\left(\frac{r_1}{R}\right)^2\right\} \\ & + \hat{\omega}_2 \left[1 - \left(\frac{r_2}{R}\right)^2\right] \exp\left\{-\left(\frac{r_1}{R}\right)^2\right\}, \end{aligned} \tag{3.36}$$

where $\hat{\omega_1} = -\hat{\omega_2} \approx 299.528385375226$ is the extremum vorticity value of the monopole at $r_1 = r_2 = 0$. The radii r_1 and r_2 are the radial distance from the positive and the negative monopoles respectively. Figure 3.10a shows the vorticity contours of this initial vorticity distribution. The initial vorticity distribution decays at an exponential rate to zero at the no-slip boundary. This means the no-slip boundary condition is still guaranteed for the initial distribution. To initialize the problem in the Eulerian domain with $\mathbf{u} - p$, we used the velocity distribution,

$$u(\mathbf{x}, t) = -\frac{1}{2}|\hat{\omega}_1|(y - y_1)\exp\left\{-\left(\frac{r_1}{R}\right)^2\right\} + \frac{1}{2}|\hat{\omega}_2|(y - y_2)\exp\left\{-\left(\frac{r_2}{R}\right)^2\right\}, \tag{3.37a}$$

$$v(\mathbf{x}, t) = +\frac{1}{2}|\hat{\omega}_1|(x - x_1)\exp\left\{-\left(\frac{r_1}{R}\right)^2\right\} - \frac{1}{2}|\hat{\omega}_2|(x - x_2)\exp\left\{-\left(\frac{r_2}{R}\right)^2\right\}, \tag{3.37b}$$

where u and v are the velocity in the x and y direction, respectively. The fluid domain of the Eulerian domain, show in figure 3.9a was discretized using an controlled unstructured meshing method. From the velocity distribution, equation 3.37, we see that the maximum velocity in the fluid will be along the y-axis (i.e $x = 0$). Therefore, to satisfy the CFL condition, we need the minimum cell size at the location of the maximum velocity. Furthermore, to ensure the vorticity generation at the no-slip boundary is defined accurately, we increased resolution of the mesh at the boundary. The third region where we increased the resolution is where the dipole and the wall interacts (i.e $-0.5 \leq x \leq 0.5$ and $0.5 \leq |y| \leq 1$). In the region where there is no vorticity, we do not need high resolution (i.e $0.5 \leq |x| \leq 1$ and $-0.5 \leq y \leq 0.5$). With these parameterization, we obtained an unstructured grid with $N_{\text{vert}} = 48k$ vertices.

Results

After initializing the velocity field in the discretized domain, the problem was evolved from $t = 0$ to $t = 2.0$ where t was non-dimensionalized using W/U. Using the CFL condition, equation 3.29, we determine that the simulation required a time step size $\Delta = 1.25 \times 10^{-5}$, with a total of $160k$ time steps. Figure 3.10 shows the evolution of the vorticity field at various instances ($t = 0, 0.25, 0.5, 0.75, 1.25$). During the initial stages of the simulation, the initialized dipole travels along the y-axis towards the bottom no-slip boundary. The weaker outer regions of the core dipole travels in the opposite direction, and for this simulation, this evolution of the this dipole is ignored.

The main dipole approaches the bottom boundary, where the no-slip boundary generates vorticity to ensure no-through flow, figure 3.10b. As the primary dipole approaches closer, the vorticity filament at the wall rolls up and combines with the primary dipole forming two secondary dipoles, that is asymmetric across the y-axis. Figure 3.10c shows the state of the vorticity field at $t = 0.5$ after the secondary dipoles are generated. This secondary dipole initially travels away from the bottom wall and later on approaches the wall again, colliding for a second time and creating a tertiary vortex, figure 3.10d. The dipole stops convecting any further and diffuses as time progresses, as shown for the time instants $t = 1$ and $t = 1.25$.

Figure 3.11 compares the vorticity contours in a small part of the computation domain ($0 \leqslant x \leqslant 0.6$ and $-1 \leqslant y \leqslant -0.4$) at $t = 1$. The positive vortex (solid black) is surrounded

by the negative vortex (dashed black). The primary observation tells us that the overall shape of the vorticity contours is very similar to the reference data. However we see that in the present simulation, more iso-vorticity lines are present meaning that the diffusion of the core is slight different.

To determine the variation of the fluid properties as the time progresses, Clercx & Bruneau investigated the evolution of the total kinetic energy $E(t)$, the total enstrophy $\Omega(T)$, and the total Palinstrophy $P(t)$ of the flow field. The total kinetic energy $E(t)$ of the dipolar field can be determined as,

$$E(t) = \frac{1}{2} \int \int \mathbf{u}^2(\mathbf{x}, t) \, \mathrm{d}x\mathrm{d}y, \tag{3.38}$$

and at $t = 0$, $E(0) = 2$. The total enstrophy of the flow is determined as,

$$\Omega(t) = \frac{1}{2} \int \int \omega^2(\mathbf{x}, t) \, \mathrm{d}x\mathrm{d}y, \tag{3.39}$$

and can seen as the energy of the vorticity. The change in enstrophy of the field can give an insight to dissipation rate in the fluid. At $t = 0$, the enstrophy of the fluid is $\Omega(0) = 800$. The total Palinstrophy $P(t)$ of the flow measures the gradient of the vorticity,

$$P(t) = \frac{1}{2} \int \int [\nabla\omega(\mathbf{x}, t)]^2 \, \mathrm{d}x\mathrm{d}y, \tag{3.40}$$

and gives an insight to generation of vorticity at the no-slip boundary. Figure 3.12 compare the evolution of these time dependent parameters with the reference data provided by Clercx & Bruneau (dotted red) for $t = 0.25$, $t = 0.5$ and $t = 0.75$. The kinetic energy, figure 3.12a, reduced from $E(0) = 2$ to $E(2) \approx 0.3$. At $t = 0.4$, we have small kink representing the approach of the primary dipole at the wall. When plotting the reference data, we see that the variation in kinetic energy matches perfectly at $t = 0.25$, $t = 0.5$ and $t = 0.75$.

Figure 3.12b compares the variation in enstrophy (Ω). During the initial stages, the enstrophy decreases linearly, and at $t = 0.37$ there is sharp increase in the total enstrophy of the flow $\Omega(0.37) = 938.58$. This indicates the initial impact of dipole with the no-slip wall. After the impact, the enstrophy quickly drops and peaks again at $t = 0.65$ reaching $\Omega = 307.04$. In addition, to the 3 data points, Clercx & Bruneau determined the peak enstrophy of flow. Table 3.4 compares the difference between the present study and literature and we see that there is maximum of 0.3% error in time t and 0.6% error in enstrophy Ω. Therefore, the variation in enstrophy is well represented by our Eulerian method.

Figure 3.12c compares the variation in palinstrophy $P(t)$. Similar to enstrophy, we can observe to peak regions at $t = 0.36$ and $t = 0.65$, respesenting the two collision of the dipole. During the collision, vorticity is generated from the wall to ensure no-through boundary condition, which results in a sharp increase in the gradient of the vorticity. Table 3.4 compares the difference with the addition data provided and we see that there is a maximum error of 0.3% in time t and 1.8% in palinstrophy P. This is an acceptable error and tells us the generation of vorticity in Eulerian method performs according to theory.

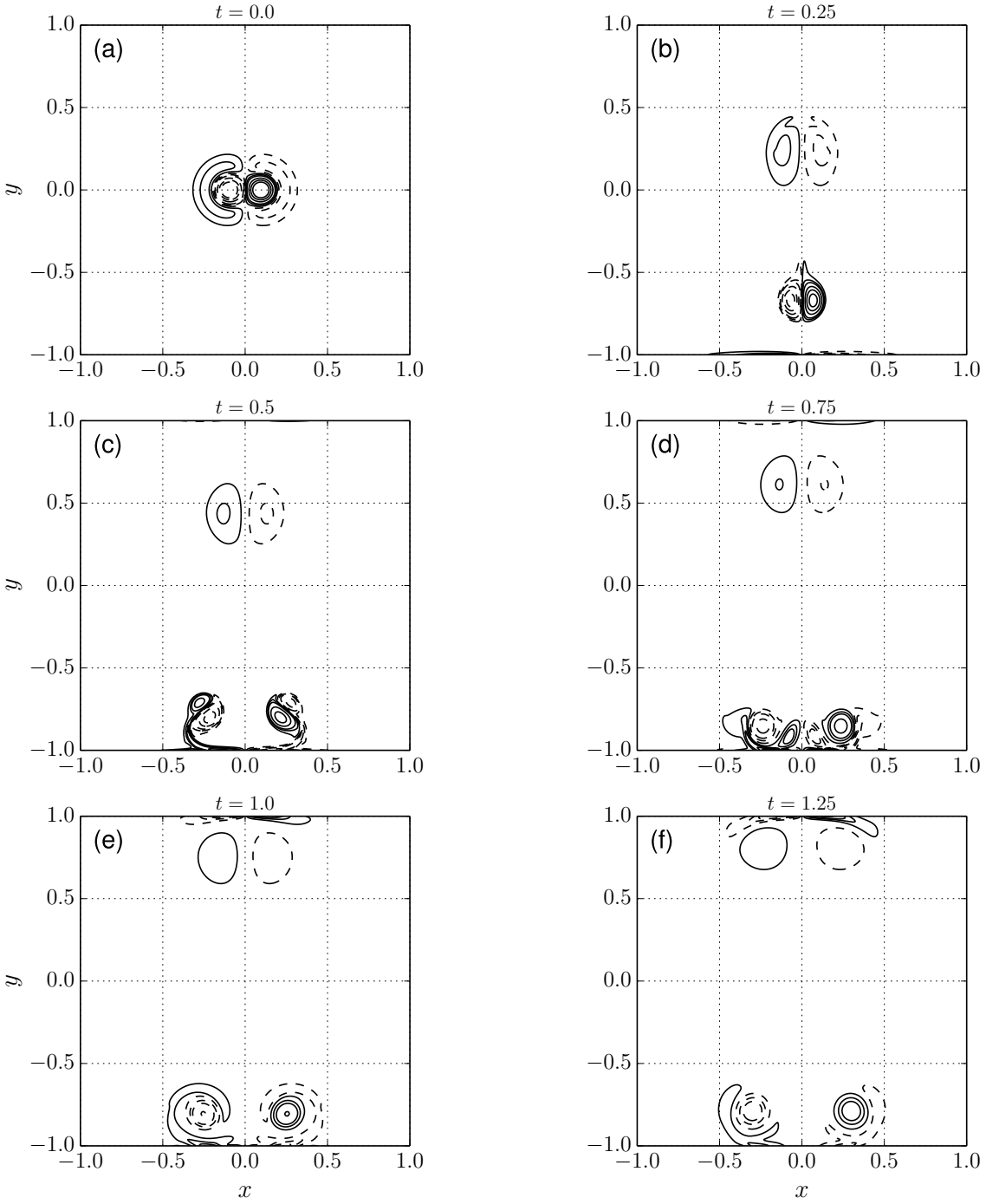
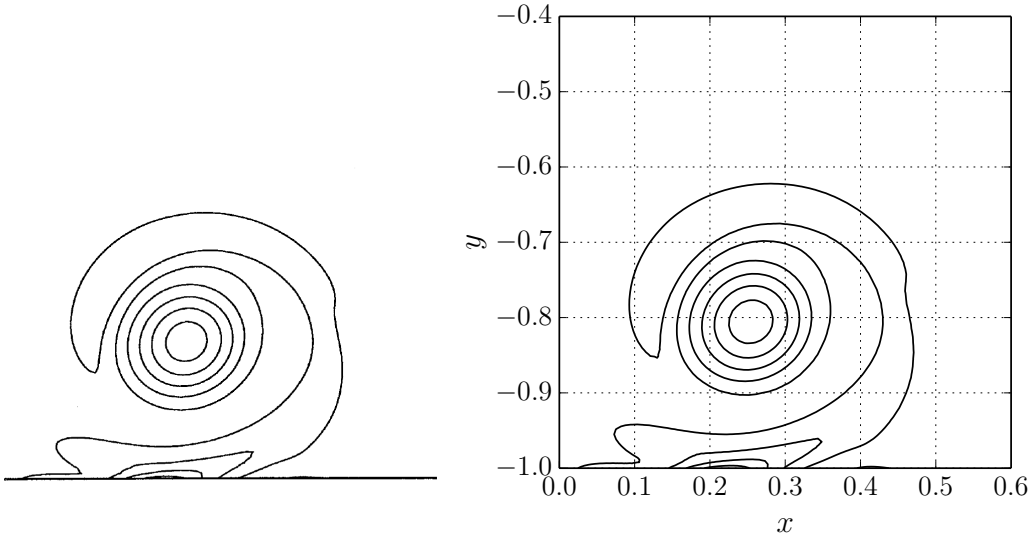


Figure 3.10: Vorticity contour plots of the normal Clercx-Bruneau dipole-wall collision experiment at $Re = 625$ at $t = 0, 0.25, 0.5, 0.75, 1.0, 1.25$ with vorticity contour levels at -320,-200,-100,-50,-10, 10, 50, 100, 200, 320. The figure depicts positive contours [—, solid black], and negative contours [- -, dashed black].

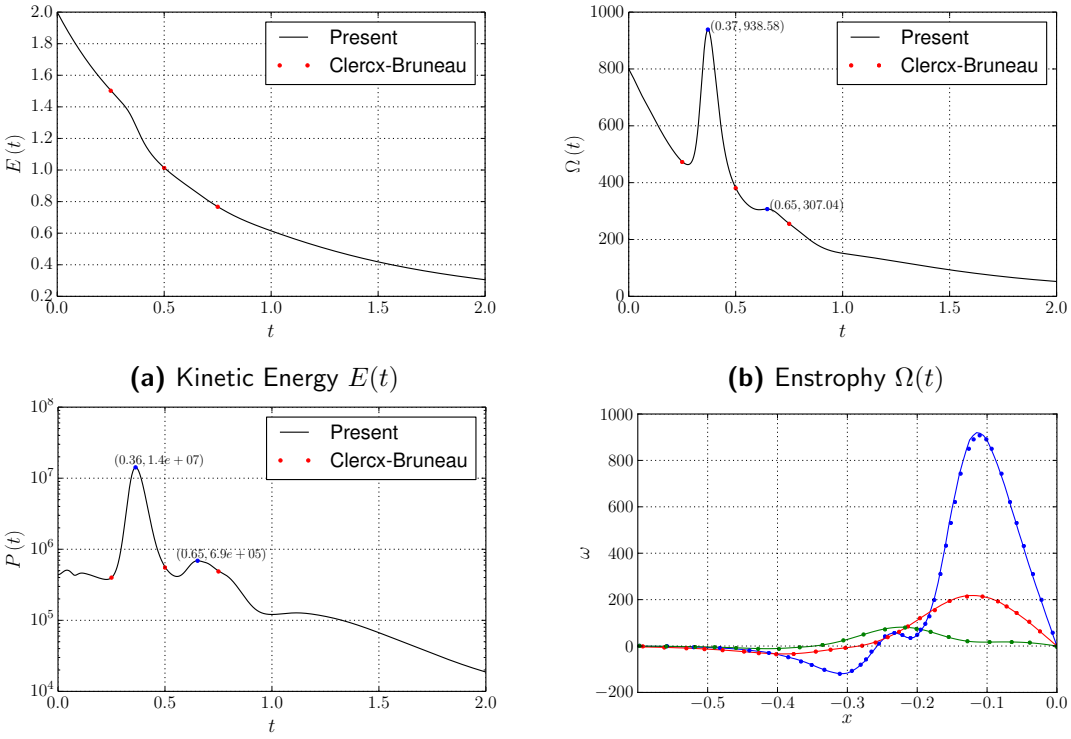
Table 3.3: Summary of the parameters for the Clercx-Bruneau normal collision of a dipole with a no-slip wall [15].

Parameters	Value	Unit	Description
Ω	$[-1,1]^2$	m	Eulerian domain bounds
Re	625	-	Reynolds number
U	1	$\mathrm{m\,s^{-1}}$	Characteristic velocity
W	1	m	Half width of the domain
ν	1.6×10^{-3}	$\mathrm{kg\,s^{-1}\,m^{-1}}$	Kinematic viscosity
$(x,y)_{1,2}$	$(\pm 0.1, 0)$	m	Initial location of the dipole
$\hat{\omega}_{1,2}$	± 299.5283853752226	-	Maximum vorticity of the monopole
(t_0, t_f)	$(0,2)$	s	Initial and final scaled viscous time
CFL	0.95	-	CFL number
$\|\mathbf{u}\|_{\max}$	12	$\mathrm{m\,s^{-1}}$	Maximum fluid velocity
Δt	1.25×10^{-5}	s	Time step size
N_{vert}	$\sim 48k$	-	Number of mesh vertices
$h_{\min}$	$\sim 3.6 \times 10^{-3}$	m	Minimum mesh cell size
N_{tsteps}	$160,000$	-	Number of time integration steps
$\mathrm{ID}_{\mathrm{fluid}}$	1	-	Fluid domain I.D
$\mathrm{ID}_{\mathrm{wall}}$	2	-	No-slip boundary I.D



(a) Literature **(b)** Present study

Figure 3.11: Comparison of the vorticity contours at $t = 1$. The figure compares the plot obtained by **(a)** literature and **(b)** the present study.



(a) Kinetic Energy $E(t)$

(b) Enstrophy $\Omega(t)$

(c) Palinstrophy $P(t)$

(d) Vorticity at the boundary y = -1.

Figure 3.12: Comparison of the fluid parameters. Figure **(a)**, **(b)**, **(c)** compares the evolution of the fluid properties from $t = 0$ to $t = 2$. Figure **(d)** compares the vorticity generated at the bottom-left wall ($y = -1$, $-0.6 \leqslant x \leqslant 0$) at $t = 0.4$ [—, solid blue], $t = 0.6$ [—, solid red] and $t = 1$ [—, solid green].

Table 3.4: A summary of the values of the first two maxima of the enstrophy E and palinstrophy P occurring at t_1 and t_2 respectively.

Instant	Case	Enstrophy Ω		Palinstrophy P	
		t	Ω	t	P
t_1	Reference[a]	0.371	933.6	0.361	1.39×10^7
	Present	0.370	938.6	0.360	1.40×10^7
t_2	Reference[a]	0.648	305.2	0.652	6.78×10^5
	Present	0.650	307.0	0.650	6.90×10^5

[a] Data obtained from Clercx & Bruneau [15]

Figure 3.12d compares the vorticity along the boundary of the domain at $y = -1$ for $-0.6 \leqslant x \leqslant 0$. The solid lines represents the present data, and is compared with the dotted data obtained from the reference. The comparison is done for various time instances $t = 0.4$, $t = 0.6$ and $t = 1.0$ and we can finally validate that the Eulerian method accurately represents vorticity generation from the wall.

3.4.3 Impulsively started cylinder at $Re = 550$

Finally, we investigated the problem of an impulsively started cylinder at $Re = 550$. This validation test ensured that at the end of the simulation, we are able to determine correct forces acting on the body.

Problem Definition

The Impulsively Started Cylinder (ISC) test case simulates the flow around a cylinder exposed to an impulsively started free-stream flow. The test cases focuses on the unsteady behavior of the separated flow past the cylinder. Various experimental and numerical investigation have been performed investigating the flow characterstics, and for this project we relied on the widely used and validated results of Koumoutsakos & Leonard [39]. They investigated the flow around the ISC using vortex methods, and provided extensive data on the vorticity profile behind the cylinder and the evolution of the Lift and Drag.

Figure 3.13 shows the domain definition of the ISC problem. Figure 3.13a shows the fluid domain Ω_E with an initial conditions $\mathbf{u} = 0$, and $p = 0$. The domain has the following boundary conditions: the no-slip wall boundary condition at $\partial\Omega_{\text{wall}}$ (solid blue) $\mathbf{u} = 0$, the free-stream Dirichlet velocity boundary condition at $\partial\Omega_{\text{dirichlet}}$ (solid red) $\mathbf{u}_\infty = [1, 0]$, and the pressure outlet $\partial\Omega_{\text{pressure}}$ (solid green). Unlike the previous test cases we now require a pressure outlet boundary condition $\partial p/\partial \mathbf{n} = 0$, as the velocity field behind the cylinder perturbed and therefore free-stream boundary condition cannot be applied there.

The unsteady simulation has Reynolds number Re of the flow dependent on the diameter of the cylinder D,

$$Re = \frac{UD}{\nu}, \tag{3.41}$$

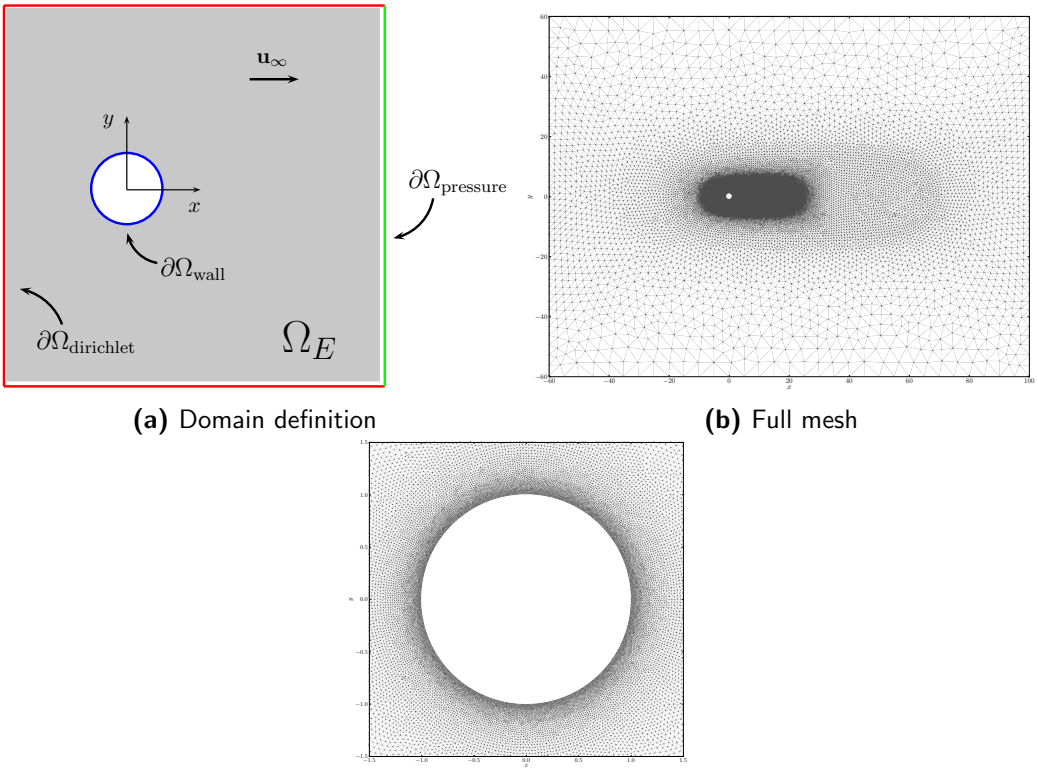
and the time t is non-dimensionalized with the radius R of the cylinder,

$$T = \frac{U}{R}t. \tag{3.42}$$

The domain was discretized with $N_{\text{vert}} = 48$, with the highest mesh resolutions at the surface of the body, and right behind the body, figure 3.13b and figure 3.13c. The simulation was time marched with $\Delta t = 1 \times 10^{-3}$ satisfying the CFL condition. The parameter of the simulation are tabulated in table 3.5.

Results

The simulation was started with an with an impulsively started free-stream boundary condition at dirichlet boundary $\partial\Omega_{\text{dirichlet}}$. The problem was evolved from $t = 0$ to $t = 100$



(a) Domain definition

(b) Full mesh

(c) Mesh near the surface

Figure 3.13: Domain of the ISC problem. The figure depicts **(a)** the definition, **(b)** the full domain mesh, and **(c)** the mesh near the surface.

Table 3.5: Summary of the parameters for the Impulsively started cylinder test case for $Re = 550$.

Parameters	Value	Unit	Description
Ω	$[-60, 100] \times [-60, 60]$	m	Eulerian domain bounds
Re	550	-	Reynolds number
$\mathbf{u}_\infty$	$[1, 0]$	$\mathrm{m\,s^{-1}}$	Free-stream velocity
R	1	m	Radius of cylinder
D	2	m	Diameter of cylinder
ν	3.6×10^{-3}	$\mathrm{kg\,s^{-1}\,m^{-1}}$	Kinematic viscosity
(t_0, t_f)	$(0, 100)$	s	Initial and final non-dimensional time
CFL	0.95	-	CFL number
$\|\mathbf{u}\|_{\max}$	2.5	$\mathrm{m\,s^{-1}}$	Maximum fluid velocity
Δt	1×10^{-3}	s	Time step size
N_{vert}	$\sim 47k$	-	Number of mesh vertices
$h_{\min}$	$\sim 9.7 \times 10^{-3}$	m	Minimum mesh cell size
N_{tsteps}	$100,000$	-	Number of time integration steps
ID_{fluid}	1	-	Fluid domain I.D (gray)
ID_{wall}	2	-	No-slip boundary I.D (blue)
$\text{ID}_{\text{dirichlet}}$	3	-	Dirichlet boundary I.D (red)
$\text{ID}_{\text{pressure}}$	4	-	Pressure boundary I.D (green)

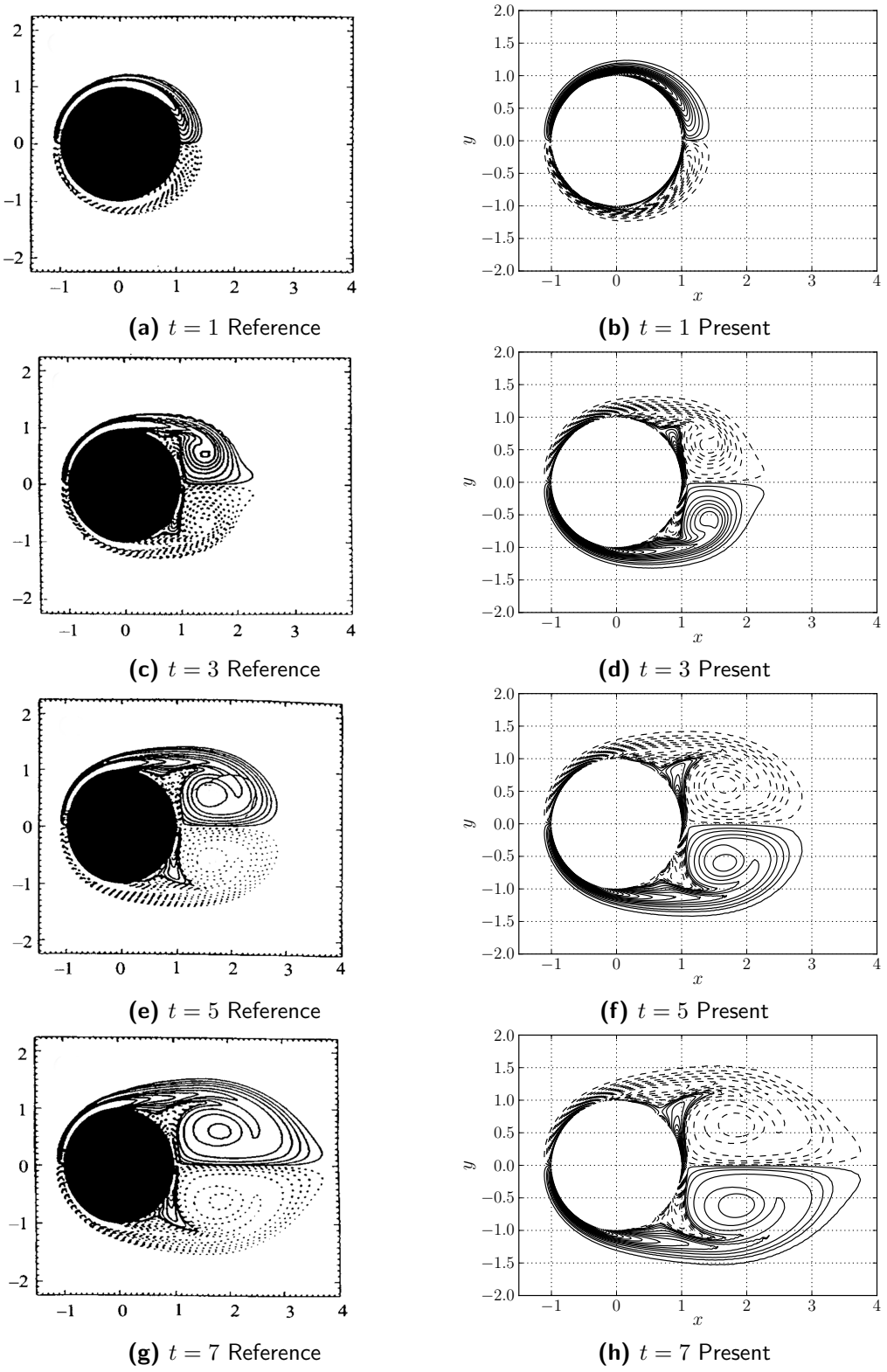


Figure 3.14: Comparison of the vorticity contours for $t = 1$, $t = 3$, $t = 5$ and $t = 7$ with contour levels $[-30, ..., -2, -1, -0.5, -0.1, 0.1, 0.5, 1, 2, ..., 30]$. The figures on left are obtained from the literature, Koumoutsakos and Leonard [39], and the figures on right are from the present study.

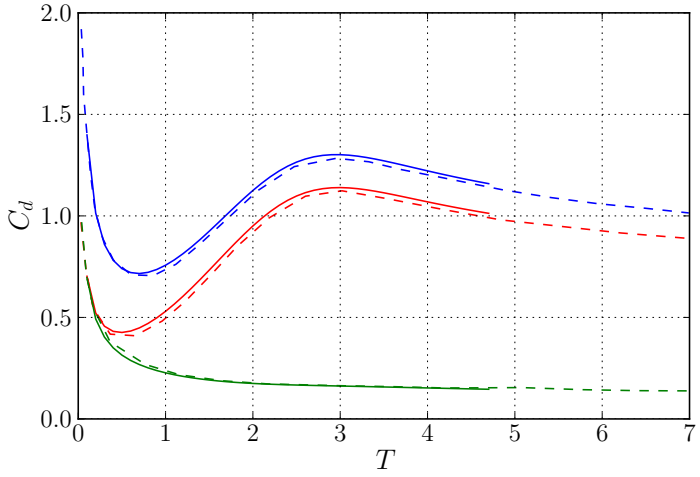


Figure 3.15: Evolution of drag force. The figure depicts the total drag coefficient C_d [—, solid blue], the pressure drag coefficient $C_{d_{\text{pres}}}$ [—, solid red] and the friction drag coefficient $C_{d_{\text{fric}}}$ [—, solid green]. The dotted lines indicated the data obtained from literature, Koumoutsakos and Leonard [39].

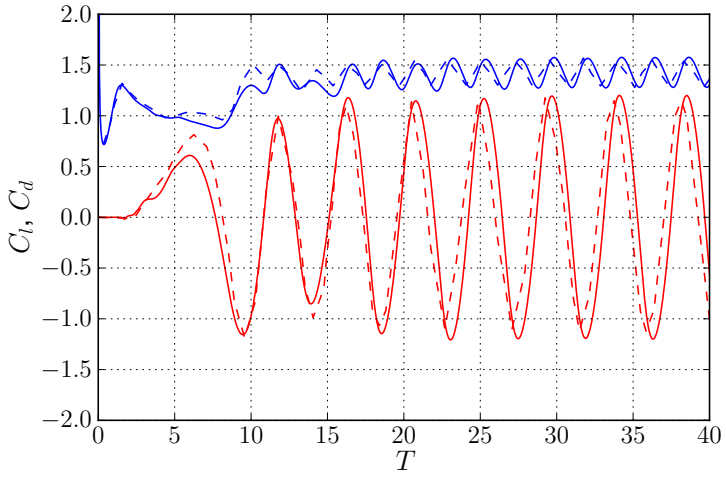


Figure 3.16: Evolution of the lift and drag coefficient from $t = 0$ to $t = 40$ with artificial perturbation [42]. The figure depicts the total drag coefficient C_d [—, solid blue] and the lift coefficient [—, solid red]. The dotted lines represent the data obtained from literature, RosenFeld et al. [52]

with number of time steps $N_{\text{tSteps}} = 100000$. To validate the scheme with the reference data of Koumoutsakos & Leonard [39], we investigated the evolution of the vorticity field and the evolution of the forces acting on the body.

Figure 3.14h depicts the evolution of the vorticity from $t = 1 \rightarrow 7$. The iso-vorticity contours of the present study is compared with the reference data from the literature [39]. At $t = 1$, negative and positive vorticity is generated at the top and bottom of the cylinder, respectively and is resulted from satisfying the no-slip boundary condition. As time progress, two primary vortices are formed right behind the cylinder, increasing in shape as time progress. Comparing the vorticity contours, we can say that the shape and the geometry of the contour lines match with the literature.

Using equations 3.31 to 3.33, we were able to calculate the Lift and the drag force acting on the cylinder as the time progress, which we used to validate against the literature. Figure 3.15 shows the components of the drag force (friction drag $C_{d_{\text{fric}}}$, pressure drag $C_{d_{\text{pres}}}$) acting on the surface of the body. At $t = 0$, we have a singularity in the total drag C_d acting on the body due to the impulsive start of the flow. It then plunges to $C_d = 0.75$ at $t = 0.8$ and peaks again near $t = 3$ at $C_d = 1.3$. The dotted line is the data obtained from literature [39] and we see that the results of the simulation matches well with the literature.

A final comparison was done for the evolution of the Lift and the Drag coefficient for larger period ($t = 0$ to $t = 40$), which was used to determine the oscillatory behavior of the Lift and Drag. For lower Reynolds number, the vorticity field is symmetric across the x-axis for a long time. This meant that the oscillatory behavior of the forces starts at a much later time. Therefore, we prescribed an artificial perturbation to the problem to create an asymmetry in the vorticity field. The perturbation was performed according to Leocointe & Piquet [42],

$$u_{\text{wall}} = \begin{cases} 0.15 & 3 \leqslant t \leqslant 3.5, \\ -0.25 & 3.5 \leqslant t \leqslant 5. \end{cases} \tag{3.43}$$

With this, we could ensure that we have a controlled behavior for the lift and drag, which we used to determine the amplitude and the frequency of the oscillation. Figure 3.16 compares the evolution of the lift and drag for $t = 0$ to $t = 40$. We see that our numerical scheme performs very similar to the literature [52]. However, there is a slight difference, which is due to the under-resolution of the Eulerian domain downstream of the cylinder.

3.5 Summary

In summary, we investigated, verified and validated the Eulerian method for the hybrid coupling scheme, and various observation of the method has been summarized as follows:

- The Eulerian method is used to highly resolve the near-body region of the fluid.
- We have used a Finite Element method to solve the incompressible laminar Navier-Stokes problem using the velocity-pressure $\mathbf{u} - p$ formulation.

- Incremental Pressure Correction Scheme (IPCS) was used to solve the Navier-Stokes problem, allowing us to decouple the velocity $\mathbf{u}$ and pressure p from the momentum equation.
- DOLFIN library from the FENICS project was used to perform automated finite element algorithms for solving the partial different equations.
- GMSH mesh generation tool was used to generate the unstructured mesh of the fluid domain.
- Once we have determined the velocity $\mathbf{u}$ and the pressure p fields, we can determine the vorticity associated to the fluid using an optimized calculation algorithm.
- A Lamb-Oseen Vortex test case was used to verify the implementation of the Eulerian method, and concluded that the method had a $1^{\mathbf{st}}$-order convergence in time and 2^{nd}-order convergence in space.
- To validate the vorticity handling and the vorticity production of the no-slip boundary, we used the high-fidelity numerical test case of Clercx & Bruneau investigation the collision of dipole with the wall at $Re = 625$. Investigating the change in kinetic energy E, enstrophy Ω, and Palinstrophy P, we validate that the results matched the literature. We evaluated the vorticity generated at boundary, which also showed that our numerical method handels according to theory.
- The final test case involved simulating an impulsively started cylinder at $Re = 550$. We investigated the shed of vorticity at time progress and validated that it matched the reference data provided by Koumoutsakos & ???. Finally, we investigated the evolution of the Lift and drag of the cylinder, and we saw that the frequency and the amplitude of the oscillation matched the theory. Therefore, our Eulerian method accurate determine the fluid behavior past an object such as the Strouhal number St.

Eulerian method algorithm

The algorithm for the Eulerian method can be summarized as follows:

1. **Mesh generation**: We generate the mesh of the fluid domain using GMSH before the iteration.
2. **Determine the boundary condition**: We need to determine the boundary conditions for the boundary domains: $\partial\Omega_{\text{wall}}$, $\partial\Omega_{\text{dirichlet}}$, $\partial\Omega_{\text{pressure}}$. If we have dirichlet velocity boundary conditions for all the exterior boundaries, we do not have to apply any pressure boundary conditions at $\partial\Omega_{\text{pressure}}$.
3. **Solve the IPCS**: Using IPCS, time march from t_n to t_{n+1} to solve for the new velocity $\mathbf{u}$ and pressure p field.
4. **Determine the vorticity**: Using the algorithm described in **??**, solve for the vorticity field ω at the time t_{n+1}.

Once we have the well-resolved vorticity ω of the near-body region, we use it to couple it with the Lagrangian method with our Hybrid coupling scheme.

3.6 Chapter Nomenclature

Latin Symbols

c	Reference length (chord)	m
C_d	Drag coefficient	-
C_l	Lift coefficient	-
CFL	CFL number	-
$\hat{\mathbf{e}}$	Cartesian unit vector	-
E	Kinetic Energy	??
f	Source terms	-
$\mathbf{I}$	Identity matrix	-
$\hat{\mathbf{n}}$	Unit normal vector	-
p	Pressure	Pa
	Trial function for pressure	-
$\mathcal{P}$	Lagrange polynomial	-
q	Degree of Lagrange polynomial $\mathcal{P}_q$	-
	Test function for pressure	-
Q	Scalar-valued function space for pressure p	-
t_n	Simulation time at n^{th} step	s
$\mathcal{T}_h$	Finite Element mesh	-
T	Cell of Finite Element mesh	-
$\mathbf{u}$	Velocity	$\mathrm{m\,s^{-1}}$
	Trial function for velocity	-
$\mathbf{u}^\star$	Tentative velocity	$\mathrm{m\,s^{-1}}$
v	Test function for velocity	-
V	Trial vector function space for velocity	-
$\hat{V}$	Test vector function space for velocity	-
w	Trial function for vorticity	-
x	Test function for vorticity	-
X	Scalar-valued function space for vorticity ω	-

Greek Symbols

Δh	Mesh cell size	m
Δt_E	Eulerian time step size	s
ϵ	Symmetric gradient	??
ν	Kinematic viscosity	??

ω	Vorticity	?
Ω	Fluid domain	-
Ω_E	Eulerian fluid domain	-
Ω_L	Lagrangian fluid domain	-
$\partial\Omega$	Boundary of the domain Ω	-
psi	Basis function	-
ρ	Fluid density	??
σ	Cauchy stress tensor	??

Chapter 4

Hybrid Eulerian-Lagrangian Vortex Particle Method

Chapter 1 introduces the Hybrid Eulerian-Lagrangian Vortex Particle Method (HELVPM), a domain decomposition method, where the Eulerian solver and the Lagrangian solver are used to solve different domains of the fluid. The algorithm that we use to couple the two solver is a modified version of approach used by Stock [59] and Daeninck [23]. The algorithm that we employ is summarized as follows:

1. **Correct Lagrangian:** Use the solutions of the Eulerian solver in the near-wall domain Ω_E to correct the solution of the Lagrangian solver, with key requirement that circulation is conserved.

2. **Evolve Lagrangian:** Evolve the newly adjusted Lagrangian solution from time t_n to t_{n+1}. The procedures of the Lagrangian solver is elaborated in Chapter 2.

3. **Determine Eulerian boundary conditions:** Use the Lagrangian solution at time t_{n+1} to determine the boundary conditions for time marching the Eulerian solver from t_n to t_{n+1}.

4. **Evolve Eulerian:** Evolve the Eulerian solver with the newly acquired boundary condition from t_n to t_{n+1}. The procedures of the Eulerian solver is elaborated in Chapter 3.

The coupling of the Eulerian and the Lagrangian solver is done at steps 1 and 3. In step 1, the Eulerian solution is transfered to the Lagrangian solver, whereas in step 3, we use the Lagrangian solution back to time-march the Eulerian solver. This chapter will be dedicated to elaborated the procedures of step 1 and step 3.

4.1 Decomposition of the domain

The hybrid solver decomposes the fluid domain into two subdomains: the near-body region, referred to as the Eulerian domain Ω_E where the Eulerian solution of the Eulerian solver is valid; and the wake region, referred to as the Lagrangian domain Ω_L where the Lagrangian solution of the Lagrangian solver is valid. Figure 4.1 shows this segregation of the fluid into this two regions. To ensure that the two solver are coupled, where steps 1 and 3 can be performed correctly, the Lagrangian domain Ω_L is overlap with the Eulerian domain Ω_E completely, such that $\Omega_E \subset \Omega_L$.

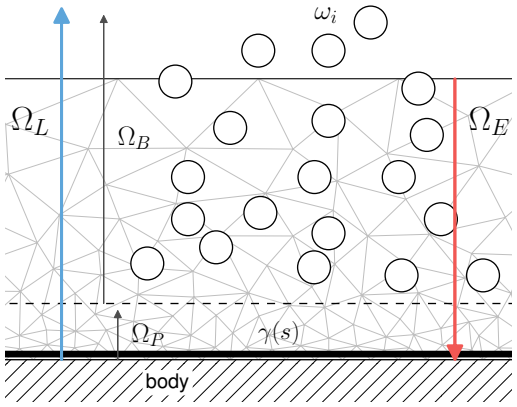


Figure 4.1: Schematic of the domain decomposition. The two subdomain are the Lagrangian domain $\Omega_L : \Omega_p \cup \Omega_b$ and the Eulerian domain Ω_E where $\Omega_E \subset \Omega_L$.

The Lagrangian domain Ω_L is further divided into two subdomain: the vortex blob domain Ω_b where the vortex blobs $\mathbf{x}_i \in \Omega_B$ resolve the vorticity ω; and the vortex panel domain Ω_p consisting of the wall-bounded vorticity resolved by the vortex panel at the surface $\mathbf{s} \in \partial\Omega_{body}$. This division of the Lagrangian domain Ω_L to Ω_b and Ω_p such that $\Omega_L = \Omega_p \cup \Omega_b$ is elaborated in section 2.5. The vortex panel was required to efficiently represent the singular vorticity distribution of the wall-bounded vortex sheet and further was necessary to enforce the wall boundary condition for the Lagrangian solver.

Therefore, the decomposition of the fluid domain is as follows:

$$\textit{fluid}: \begin{cases} \Omega_E & \{\textit{Eulerian domain}\}, \\ \Omega_L = \Omega_p \cup \Omega_b & \{\textit{Lagrangian domain}\}, \end{cases} \tag{4.1}$$

and should satisfy the following requirements:

- Eulerian domain belongs to the near-wall region of the Lagrangian domain, $\Omega_E : \Omega_E \subset \Omega_L$, bounded by the wall $\partial\Omega_{body}$ and the exterior Eulerian boundary $\partial\Omega_E$ such that $\Omega_E : \Omega_E \in [\partial\Omega_{body}, \partial\Omega_E]$.
- Vortex panel domain Ω_p belongs to the near-wall region of the Eulerian domain, $\Omega_p : \Omega_p \subset \Omega_E$, bounded by the wall $\partial\Omega_{body}$ and the boundary $\partial\Omega_p$ such that $\Omega_p : \Omega_p \in [\partial\Omega_{body}, \partial\Omega_p]$.

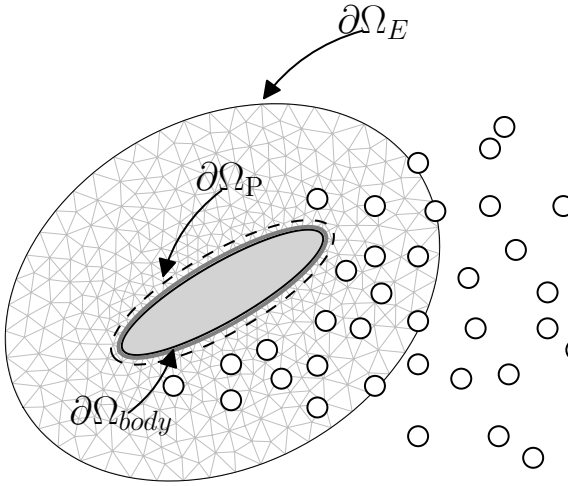


Figure 4.2: Boundaries of the decomposed domains. No-slip boundary $\partial\Omega_{body}$, exterior vortex panel boundary $\partial\Omega_p$, exterior Eulerian boundary $\partial\Omega_E$.

- Vortex blob domain Ω_b resolves the off-wall region of the Lagrangian domain $\Omega_b = \Omega_L \backslash \Omega_p$, overlaps with the Eulerian domain $\Omega_b \cap \Omega_E \neq \varnothing$. The vortex vortex blob domains starts from panel exterior boundary $\partial\Omega_p$ and continuous to full fluid domain, $\Omega_b : \Omega_b \in [\partial\Omega_p, \infty)$.

During the decomposition of the domain, we defined three boundaries, figure 4.2:

- $\partial\Omega_{body}$: No-slip wall boundary of the Eulerian domain Ω_E and the vortex panel domain Ω_p.
- $\partial\Omega_p$: External boundary of the vortex panel domain Ω_p.
- $\partial\Omega_E$: External boundary of the Eulerian domain $\partial\Omega_E$ where the Dirichlet velocity boundary condition will be prescribed.

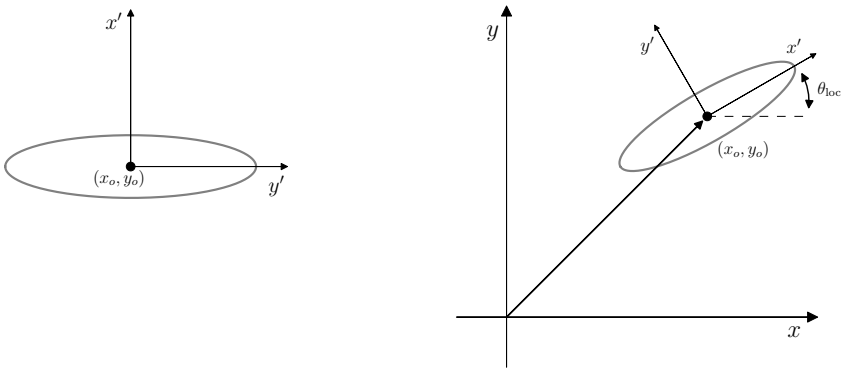
The solutions in the overlap region $\Omega_E \cap \Omega_L$ will be used to couple the Eulerian and the Lagrangian solver, based on the procedures of Stock [59] and the Daeninck [23].

4.1.1 Local to Global Transformation

The geometries in the simulation are defined in their respective local coordinate system $[x, y]'$. Figure 4.3a shows an elliptical geometry defined in its local coordinate system about its origin $[x_o, y_o]'$. The origin point is defined such that it is the center of rotation and any rotation will be prescribed about the origin point.

The body is then transformed to the global coordinate system $[x, y]$ by the displacement vector $[x_o, y_o]$ and a local rotation by θ_{loc} about the local origin $[x_o, y_o]$.

The Eulerian solver defines the body mesh in the local coordinate system is then transformed to global position using these parameters. Similarly, the panel geometry for the Lagrangian solver is defined in the same fashion. For a moving problems, the displacement vector and the local rotation angle can be updated to prescribe the motion.



(a) Local coordinate system $[x, y]'$ **(b)** Global coordinate system $[x, y]$

Figure 4.3: Elliptical geometry in **(a)** the local coordinate system and **(b)** the global coordinate system. The geometry is positioned using the displacement vector $[x_o, y_o]$ and rotated by θ_0 about the local origin point.

4.2 Correction of Lagrangian domain

The first step of the hybrid coupling scheme is to transfer the highly resolved Eulerian solution from the Eulerian solver to the Lagrangian solver. The vorticity in the domain Ω_E is transfered from the grid of the Eulerian solver onto the vortex blobs.

4.2.1 Approach from literature

This is the approach used by Stock [59] and is based on the assumption that the Eulerian solution is correct from the body up to 'somewhat inside of the outer Eulerian domain", and the Lagrangian solution is correct outside the outer Eulerian boundary. Figure 4.4 shows Daeninck's [23] result of hybrid coupling. It shows the vorticity field behind a cylinder and at the outer boundary $\partial\Omega_E$ one can observe a slight mismatch in the vorticity, and some artificial vorticity. Daeninck has observed this and stated that this is due to the slight difference in the solution of the two solvers.

Stock solution to this problem was to interpolate only part of the Eulerian domain of the Eulerian solver onto the Lagrangian solver ignoring the regions of incorrect vorticity field. This introduces the definition of the interpolation region Ω_{int}, as shown in figure 4.5a. In addition to the outer boundary region, Stock also proposed to ignore the boundary layer region during interpolation to the vortex blobs. His reasoning for ignoring this region during the correction process was that because the boundary layer has very strong vorticity gradient, they cannot be efficiently represented using the Gaussian vortex kernels. To resolve this singular vorticity distribution, we have to use boundary elements such as the vortex panel kernels, which can efficiently represent such distribution. Therefore, the interpolation region Ω_{int}, figure 4.5, has the following properties:

- The region is within the overlap region $\Omega_E \cap \Omega_b$ such that $\Omega_{int} : \Omega_{int} \subset \Omega_E \cap \Omega_b$.

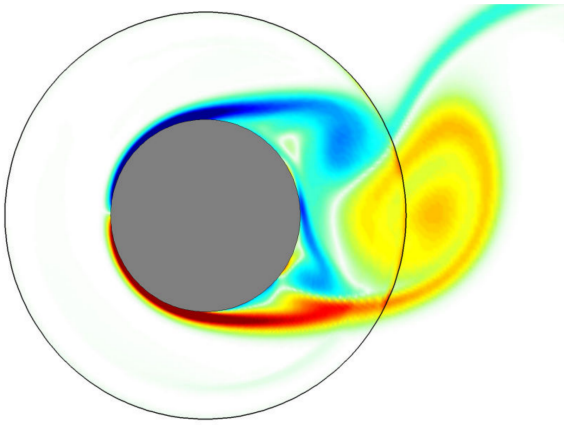
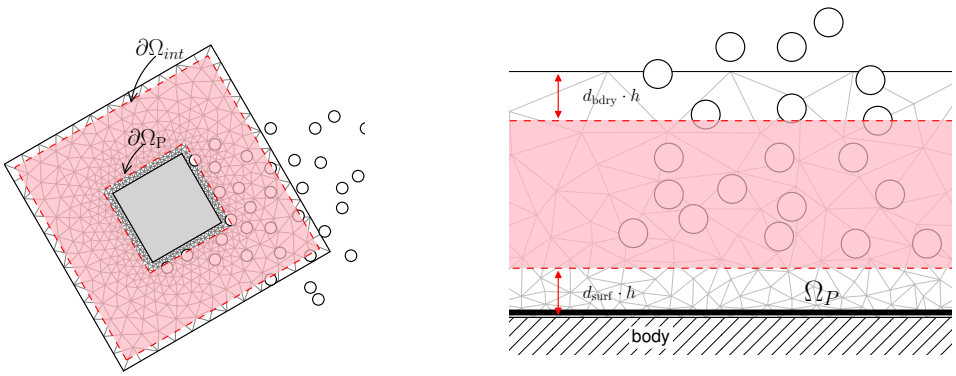


Figure 4.4: Result of hybrid coupling by Daeninck [23]. The figure shows artificial vorticity at the boundary of the Eulerian domain.

- The region is starts from the outer vortex panel boundary $\partial\Omega_p$. All the vorticity of the vortex panel boundary is represented using the vortex panels. The interpolation region ends at $\partial\Omega_{int}$, slight distance away from the outer Eulerian boundary $\partial\Omega_E$ ignoring the region of incorrect vorticity, as shown in figure 4.4.

- The offset of the interpolation region boundaries are in the order of the nominal vortex blobs spacing h. The boundary $\partial\Omega_p$ is offset by $d_{surf} \cdot h$ from the surface $\partial\Omega_{body}$, where Stock used $d_{surf} = 3$. The boundary $\partial\Omega_{int}$ is offset by $d_{bdry} \cdot h$ from the outer Eulerian boundary $\partial\Omega_E$, where Stock used $d_{bdry} = 2$.

- For an M_4' interpolation kernel and for high Re flows, Stock [59], stated that $d_{surf} = 3$ and $d_{bdry} = 2$ will ensure proper interpolation of the solutions.



(a) Boundaries of $\partial\Omega_{int}$ **(b)** Offset of the boundaries

Figure 4.5: Definition of the interpolation region $\partial\Omega_{int}$ with the boundaries: $\partial\Omega_p$ and $\partial\Omega_{int}$.

The summary of the interpolation algorithm used by Stock [59] based on the works of Daeninck [23] and Guermond and Lu [30] for interpolation the Eulerian solution onto the vortex blobs is as follows:

1. Interpolate the solution from Eulerian domain onto a temporary structured grid. The temporary structured grid has $\Delta x = \Delta y = h$, the nominal particle spacing and covers the entire Eulerian domain.
2. Identify the particles inside the interpolation region Ω_{int}. Fill gaps in the region with zero-strength particles, so that we can interpolate the Eulerian solution onto them.
3. Reset the strengths of the particles $\mathbf{x}_i$ inside the interpolation region Ω_{int}, figure 4.5a, using the local particle volume and the vorticity interpolated from the grid (i.e $\alpha_i = \omega_i \cdot h^2$).

However, during our research we have determined that this approach suffers from some issues and mainly does not guarantee that the circulation is conserved.

4.2.2 Issues with the correction algorithm

The two main issues with the above approach is as follows:

- The vorticity bounded to the solid wall was not interpolated to the vortex blobs.
- The particle strength initialization using the cell circulation equation, $\alpha_i = \omega \cdot h^2$, does not ensure the local conservation of circulation.

Vorticity at the solid wall

The first problem that we are concerned is that the vorticity bounded at the solid wall of the Eulerian solver was not transfered to the Lagrangian solver. Stock [59] use a Lagrangian solver with BEM that diffuses the vortex sheet to the vortex blobs. However, we implemented the approach of Daeninck [23], that requires the Eulerian solver to introduced the vorticity generated at the wall to the Lagrangian solver.

To ensure that all the vorticity in fluid is represented, we will use the vortex panels to represents the vorticity of the boundary layer region Ω_p. Furthermore, if the body is at motion, the body will contain circulation do the motion,

$$\Gamma_{body} = \iint_{body} \nabla \times \mathbf{u}_b \, \mathrm{d}A. \tag{4.2}$$

To ensure that all the vorticity is transfered from the Eulerian solver to the Lagrangian solver, we propose to modify Stock's algorithm to transfer the circulation of the domain Ω_p and the circulation in the body Γ_{body} to the vortex panels such that we do not violet the conservation of circulation.

Vorticity Field interpolation error

The second issue we must tackle is the interpolation error that arises due to the standard approach of initializing the particles using the local particle volume and the local vorticity,

$$\alpha_i = \omega_i \cdot h^2, \tag{4.3}$$

where i corresponds to the vortex blobs $\mathbf{x}_i \in \Omega_{int}$. We summarized this issue in the section 2.2.4 of the Lagrangian chapter which was extensively investigated by Barba and Rossi [1]. To solve the wake domain of the fluid, we used a vortex particle method that discretizes the vorticity field using N quadrature points,

$$\omega \approx \omega^h(\mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i \delta(\mathbf{x}_j - \mathbf{x_i}). \tag{4.4}$$

To remove the singularity of the kernel δ, we used a smooth Gaussian kernel ζ_σ. This approach of using vortex blobs is common in the research of vortex particle method and ensures continuous vorticity distribution. However, the downside to this approach is that on top of the discretization error, we now introduce the "smoothing error" or the "regularization error" due to the use of gaussian kernel. This is equivalent to blurring the vorticity field, as explained by Barba and Rossi [1], and the cumulative error in the initialization error is given as:

$$\text{Error} = \text{Smoothing Error} + \text{Discretization Error}$$

To perform accurate interpolation of the vorticity ω inside the interpolation domain Ω_{int} onto the particles $\mathbf{x}_j \in \Omega_{int}$, we must satisfy the following interpolation problem:

$$\omega(\mathbf{x}) \mid_{\text{Eulerian Solver}} = \hat{\omega}(\mathbf{x}) \mid_{\text{Lagrangian Solver}}, \tag{4.5}$$

where the $\omega(x)|_{\text{Eulerinan Solver}}$ is the Eulerian solution from the Eulerian solver, and the smoothed Lagrangian vorticity from the Lagrangian solver is given as,

$$\hat{\omega}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \zeta_\sigma(\mathbf{x} - \mathbf{x}_i). \tag{4.6}$$

The discrete vorticity field is represented by the linear combinations of the Gaussian basis function ζ_σ with the vortex blob strength α_i. Therefore taking that $\alpha_i = \omega(\mathbf{x}_i) \cdot h^2$ is mathematically incorrect and does not ensure the interpolated vorticity field matches the original vorticity distribution from the Eulerian solver.

We discussed the Beale's iterative method for retaining the original vorticity distribution in section 2.2.4, however this approach cannot be employed for decomposed domains. The Beale's method uses equation 4.6 to construct a linear system of equation to directly solve for the particle strengths α_i:

$$\mathbf{A}_{ij}\alpha_i = \omega_i, \tag{4.7}$$

where the coefficient matrix $\mathbf{A}$ is given as,

$$\mathbf{A}_{ij} = \zeta_\sigma(\mathbf{x}_j - \mathbf{x}_i). \tag{4.8}$$

However inverting the matrix $\mathbf{A}$ is still an open question, as stated by Koumoutsakos and Cottet [21], and was the primary investigation of Barba and Rossi [1]. The problem is that the matrix $\mathbf{A}$ is full and badly condition for direct inversion. For a global field interpolation (i.e for unbounded domain), one could use the Beale's iterative method which uses a successive over-relaxation (SOR) for solving the equation 4.8. This method relies on iterative correction of all the particles $\mathbf{x}_i \in \Omega_L$, in the full Lagrangian domain. However, in our case of initializing the strengths of the particles $\mathbf{x}_i$ in the sub-domain Ω_{int} of the Lagrangian domain Ω_L, it would require us to modify the strength of only the particles $\mathbf{x}_i$ in Ω_b. In such case, the Beale's iterative method is not valid and cannot be used. Therefore, the Beale's method cannot be used to solve the problem of the smoothing error.

In future, the key to solving this smoothing error might be in the research works of Barba and Rossi [1], where they try to reverse the blurring of the vorticity field by reversing the "diffusion" caused by the smoothing kernel. However, currently for our investigation the best possible way of ensure minimal interpolation error from Eulerian domain onto vortex blobs is to perform the following steps:

- Minimize the smoothing and discretization error by maximizing the particle resolution, achieved by setting $Ov = 1$ and reducing σ such that the relative error $\epsilon \leqslant 5\%$. The convergence of the overlap Ov and the core spreading σ was investigated in section 2.2.4.
- A vital requirement for vortex particle method is the conservation of circulation. Therefore, to ensure that the Hybrid method is valid, we ensure that the interpolation of the vorticity from the Eulerian solver to the Lagrangian solver satisfies the conservation of circulation.

Thus, we will modified the approach of Stock [59] to ensure that all the vorticity in transfered from the Eulerian solver to the Lagrangian solver and that we satisfy the conservation of circulation.

4.2.3 Modified correction strategy

The modified version of the correction can be divided into five steps

1. **Probe vorticity**: Interpolate the vorticity from the unstructured Eulerian mesh onto a uniform structured grid.
2. **Remove particles**: Remove particles that are inside the interpolation domain Ω_{int}.
3. **Generate particles**: Generate zero-strength particle inside the interpolation domain Ω_{int}.
4. **Assign strengths**: Use the standard particle initialization approach, $\alpha_i = \omega_i \cdot h^2$ to assign the particles $\mathbf{x}_i$.
5. **Conserve circulation**: Determine the mismatch in the total circulation of the Lagrangian field as determine the strength of the panels such that circulation is conserved.

Probe vorticity

The first sub-step of the correction step is to interpolate the vorticity from the unstructured Eulerian grid onto a uniform structured grid. The purpose of the structured grid is to perform fast and efficient interpolation of vorticity from the Eulerian domain onto the vortex blobs.

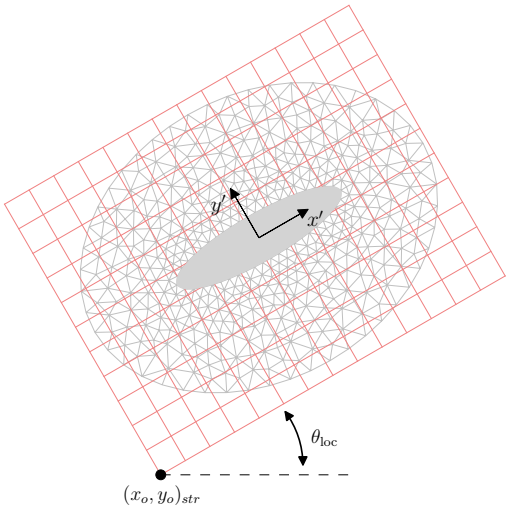


Figure 4.6: Structured interpolation grid $\mathbf{x}_{str}$ (pink) covering the entire Eulerian mesh (gray).

The structured grid $\mathbf{x}_{str}$ is defined in the local coordinates system of the geometry $[x, y]'$ where the grid covers the entire Eulerian domain Ω_E. Figure 4.6 shows the structured grid bounded to the Eulerian domain in the global coordinate system. The vorticity function ω of the function space X of the Eulerian solver is interpolated from the unstructured mesh $\mathbf{x}_{\text{unstr}}$ onto the structured uniform grid $\mathbf{x}_{\text{str}}$,

$$\hat{\omega}_i = \sum_k \omega_k W_{ki} \tag{4.9}$$

using the interpolation weight W, where $\hat{\omega}$ is the interpolated vorticity. Figure 4.7 shows a depiction of the transfer of the vorticity from the unstructured grid to the structured grid. As the structured grid $\mathbf{x}_{str}$ that does not move w.r.t to the unstructured mesh $\mathbf{x}_{unstr}$, the interpolation weight W only needs to be calculated once, ensuring fast interpolation. We used the `Probe` function, a C++ implementation developed by Mortensen [47], to the probe the vorticity function space X for the structured vorticity $\hat{\omega}$ at the nodes of the structured grid $\mathbf{x}_{str}$.

Once we have determine $\hat{\omega}$, we can assign the strengths of the particles using an efficient index search algorithm to find the location of the particle in the structured grid. If we had not used this approach and directly transfered the vorticity from the unstructured mesh $\mathbf{x}_{unstr}$ onto the vortex blobs $\mathbf{x}_i$, at each iteration we would require an expensive search algorithm to determine the position of the blob w.r.t to the nodes of the unstructured grid. This would mean that we would have to construct the interpolation matrix at each iteration, drastically reducing the efficiency of interpolation.

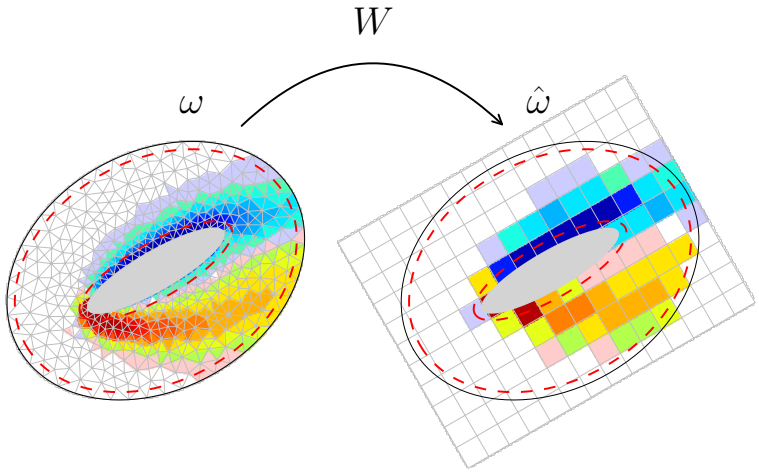
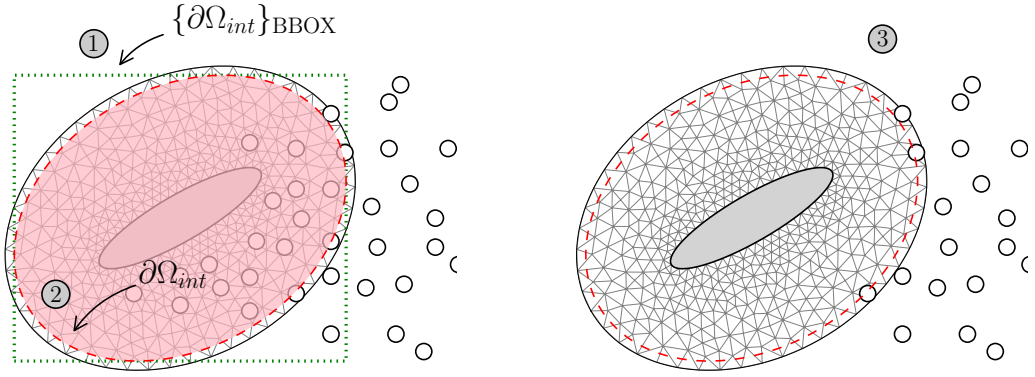


Figure 4.7: Interpolated vorticity $\hat{\omega}$ on the structured grid $\mathbf{x}_{str}$ from interpolating ω of the unstructured grid $\mathbf{x}_{unstr}$ with the interpolation weights W.

Remove particles

The second sub-step of the correction step is remove the particles that are inside the interpolation region Ω_{int} and the vortex panel domain Ω_p. The purpose of this step is that we want to ultimately correct the Lagrangian solution in these regions with the more refined Eulerian solution of the domain Ω_E. To perform the coupling, we first need to remove the particles in the region of correction Ω_{int} and Ω_p. Figure 4.8a shows the vortex blobs $\mathbf{x}_i$ inside the boundary $\partial\Omega_{int}$ that needs to be removed. To see which particles are inside, we need to perform a "Point inclusion in polygon" test to determine which



(a) Particles inside the interpolation region

(b) Total circulation $\Gamma_{removed}$ removed.

Figure 4.8: The interpolation region Ω_{int}, bounded by the boundary polygons: panel region boundary $\partial\Omega_P$ near the wall, and exterior boundary $\partial\Omega_{int}$ near the outer region.

particles $\mathbf{x}_i$ are within the boundary polygon $\partial\Omega_{int}$. However, this point-in-polygon search is computationally expensive but can be simplified by neglecting the particles outside the minimum bounding box of the polygon. Thus the steps to remove the vortex blobs inside the interpolation region is as follows:

1. Determine which particles inside the bounding box of the polygon $\partial\Omega_{int}$.
2. Perform a point-in-polygon test for only the particles $\mathbf{x}_i \in \text{BBOX}\{\partial\Omega_{int}\}$.
3. Remove the particles $\mathbf{x}_i$ from the total set of particles, resulting in a total circulation change of $\Gamma_{removed}$.

To perform the point-in-polygon test, we used the `pnpoly` function of `matplotlib`, the python 2D plotting library created by Hunter [33]. The function implemented the "point inclusion in polygon" test algorithm developed by Franklin [27]. The algorithm is based on the crossings test, which determines whether the point is inside the polygon by determining the number of the times a semi-infinite ray originating from the point intersects with the polygon.

Generate particles

The third sub-step of the correction step is generate zero-strength particles inside interpolation region Ω_{int}, figure 4.9. These zero-strength particles will later be corrected with the strengths obtained from the structured grid $\mathbf{x}_{str}$.

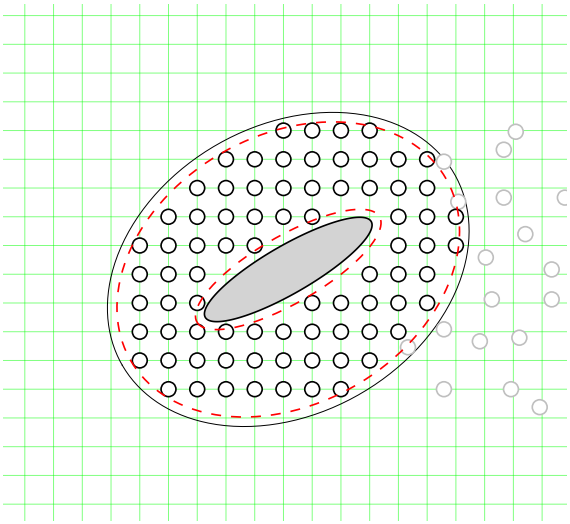


Figure 4.9: Particles inside the interpolation domain Ω_{int} located at $\mathbf{x}_i$ coinciding the Lagrangian remeshing grid.

The procedures of generating zero-strength particles are as follows:

1. Generate zero-strength particles $\mathbf{x}_i$ inside the bounding box of the boundary polygon $\partial\Omega_{int}$, $\mathbf{x}_i \in \text{BBOX}\{\partial\Omega_{int}\}$. The position of the particles $\mathbf{x}_i$ coincides with the global Lagrangian remeshing grid (shown in green), such that particles are equally spaced.

2. Perform a point-in-polygon test for the particles $\mathbf{x}_i$, so that we can neglect the particles outside of the interpolation boundary $\partial\Omega_{int}$, the particles inside the body Ω_{body}, and the particles inside the vortex panel domain Ω_p, leaving us only the particles $\mathbf{x}_i \in \Omega_{int}$.

Figure 4.9 shows the newly generated particles (in black) and the pre-existing set of particles (in gray). Onces we have uniformly distributed particles covering all the regions of the interpolation region Ω_{int}, we can transfer the solution from the Eulerian solver to the Lagrangian solver.

Assign strengths

The fourth sub-step of the correction is to assign the strengths to the newly generated particles in the interpolation domain Ω_{int}. The strengths of the particles α_i is determined using the standard method,

$$\alpha(\mathbf{x}_i) = \hat{\omega}(\mathbf{x}_i) \cdot h^2, \tag{4.10}$$

where the local circulation inside the area h^2 is assigned to the particle. We have minimized the h^2 interpolation area such that the smoothing error caused the Gaussian kernel is acceptable, see section 2.2.4. Therefore, to determine the strength of the particles inside the interpolation region, we simply require the vorticity $\hat{\omega}(\mathbf{x}_i)$ at the local $\mathbf{x}_i$. We have interpolated the vorticity from the unstructured finite element grid nodes onto a structure grid $\mathbf{x}_{str}$. We can transfer the vorticity from the structured grid onto to vortex blobs using an efficient Bilinear interpolation algorithm.

Figure 4.10 shows the schematic representation of the Bilinear interpolation of vorticity. The vortex blob (in blue) is located inside the one of the cells of the structured grid (in

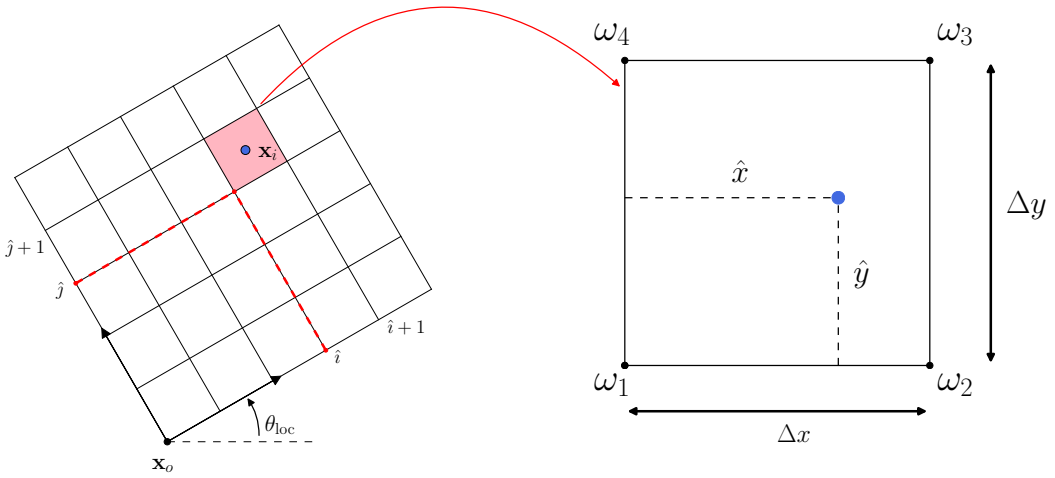


Figure 4.10: Interpolating the strengths from the structured grid $\mathbf{x}_{str}$ onto the vortex blobs $\mathbf{x}_i$ using a bilinear interpolation

pink), bounded by 4 grid nodes:

$$\begin{aligned} p_1 &= \mathbf{x}_{i,j}, \\ p_2 &= \mathbf{x}_{i+1,j}, \\ p_3 &= \mathbf{x}_{i+1,j+1}, \\ p_4 &= \mathbf{x}_{i,j+1}. \end{aligned} \tag{4.11}$$

The four nodes $p_1, ..., p_4$ are defined in the anti-clockwise direction. The bilinear interpolation of the vorticity becomes,

$$\hat{\omega}(\mathbf{x}_i) = \sum_{k=1}^{4} W_k \cdot \omega_k \tag{4.12}$$

where $\{\omega_k, W_k\} \mapsto p_k$. The interpolation weights W_k are defined as

$$\begin{aligned} W_1 &= \frac{(\hat{x} - \Delta x)(\hat{y} - \Delta y)}{\Delta x \Delta y} \\ W_2 &= \frac{-\hat{x}(\hat{y} - \Delta y)}{\Delta x \Delta y} \\ W_3 &= \frac{\hat{x}\hat{y}}{\Delta x \Delta y} \\ W_4 &= \frac{-\hat{y}(\hat{x} - \Delta x)}{\Delta x \Delta y} \end{aligned} \tag{4.13}$$

where the cell of the structured grid has dimension $[\Delta x, \Delta y]$, with $\Delta x = \Delta y$. The coordinates of the blobs are normalized such that, the vortex blobs is located at $0 \leqslant \hat{x} \leqslant \Delta x$ and $0 \leqslant \hat{y} \leqslant \Delta y$. Figure 4.11 shows the results of the assigning the strengths of the particles from the structured grid.

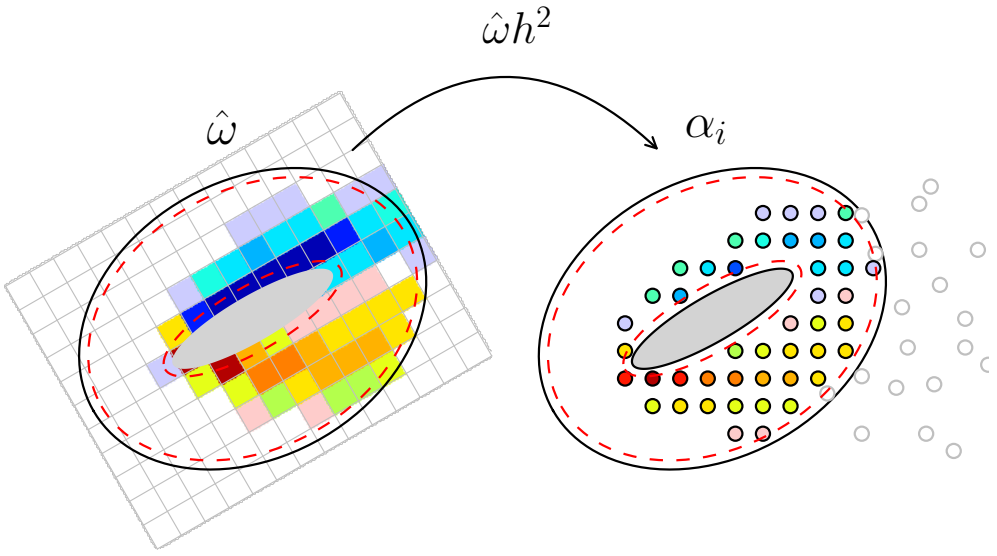


Figure 4.11: Interpolated strengths α_i from the structured grid $\mathbf{x}_{str}$ using bilinear interpolation.

Conserve circulation

The fifth and the final step of correcting the Lagrangian filed is to ensure that the we have to ensure that circulation is conserved. Two main source of errors for the conservation of circulation is the vorticity at the solid wall, and the vorticity field interpolation error.

The vorticity at the solid was not transfered to particles because the Gaussian kernels cannot efficiently represent the singular distribution. However, we cannot simply neglect this distribution and therefore we use the vortex panels to represent these. To ensure that the conservation of circulation is satisfied in the Lagrangian solver, we will prescribe the integral strengths of the panels (i.e the total circulation).

The second error is the vorticity field interpolation error. The correction algorithm so far does satisfy the conservation of circulation, so we will employ Kelvin's circulation theorem to ensure circulation is conserved. If we are dealing with fluid flow with initial total circulation $\Gamma_0 = 0$, then according to Kelvin's circulation theorem, we require that at all times t,

$$\Gamma_{panels} + \Gamma_{blobs} = 0. \tag{4.14}$$

Thus, to ensure that the circulation is conserved, we have to solve for the no-slip panels such that the total circulation is zero. However, in a general case (especially when we are dealing with multiple bodies), we have to formulate the equality in a different manner. We have to define two regions of the flow, domain where Eulerian solution is valid,

$$\Omega_{inside} = \Omega_{body} \cup \Omega_p \cup \Omega_{int}, \tag{4.15}$$

and domain where Lagrangian solution is valid,

$$\Omega_{outside} = \Omega_L \backslash \Omega_{inside}. \tag{4.16}$$

The total circulation of the Lagrangian solver now becomes,

$$\Gamma_b^{outside} + \Gamma_b^{inside} + \Gamma_p = 0 \tag{4.17}$$

where $\Gamma_b^{outside}$ is the total circulation in the domain $\Omega_{outside}$, and $\Gamma_b^{inside} + \Gamma_p$ is the total circulation in the domain Ω_{inside} from the Lagrangian solver. Due to the correction algorithm, we require that the Lagrangian solutions in the domain Ω_{inside} matches the Eulerian solution, therefore we have equality:

$$\Gamma^{inside}\Big|_{Eulerian\ Solver} = \Gamma_b^{inside} + \Gamma_p\Big|_{Lagrangian\ Solver}, \tag{4.18}$$

where Γ^{inside} total circulation from the Eulerian solution in the domain Ω_{inside}. From the equality, we can derived the net circulation of the vortex panels,

$$\Gamma_p = \Gamma^{inside}\Big|_{Eulerian} - \Gamma_b^{inside}. \tag{4.19}$$

Section 2.5 summarized the methodology for solving the no-slip boundary condition using the vortex panel using this prescribed net strengths Γ_p. Do to the slight error in coupling, we have an error in the total circulation ϵ_Γ,

$$\Gamma_b^{outside} + \Gamma_b^{inside} + \Gamma_p = \epsilon_\Gamma. \tag{4.20}$$

To remove this mismatch in total circulation, we will have to modify the strengths of the newly generated vortex blobs such that the total circulation is conserved. The mismatch in total circulation ϵ_Γ is correctly uniformly with all the vortex blobs such that:

$$\hat{\alpha}_i^{inside} = \alpha_i^{inside} - \frac{\epsilon_\Gamma}{N^{inside}}, \tag{4.21}$$

where $\hat{\alpha}_i^{inside}$ is the corrected vortex blob strengths, α_i^{inside} is the previous strengths of the vortex blobs, and N^{inside} is the number of vortex blobs $\mathbf{x}_i \in \Omega_{int}$.

4.3 Evolution of the Lagrangian solution

We have corrected the near-region solution of the Lagrangian domain $\Omega_L \cap \Omega_E$ with solution obtained from the Eulerian solver. Furthermore, we have solved the vortex panels such that: (a) it conserves the total circulation, and (b) it ensure the no-through/no-slip boundary condition. With the initial conditions provided at t_n, we can use the algorithms described in chapter 2, to evolve the Lagrangian solution from t_n to t_{n+1}. We can summarize the several features of the Lagrangian solver as follows:

- A 4^{th}-order Runge-Kutta method is used to time march the vorticity field from t_n to $t_n + 1$.
- The Lagrangian solver has a convection time step size $\Delta t_c = t_{n+1} - t_n$.
- We use Tutty's diffusion scheme such that the diffusion time step size $\Delta t_d = \Delta t_c$. Tutty's diffusion scheme is used in the majority of the cases as it is more versatile as it enables us to diffuse with convection ensure well represented vorticity field at every time t.
- The strengths of the vortex panels γ_i remains constant during t_n and t_{n+1}. This is derived from the assumption that the change in the total circulation in domain Ω_p is small during t_n and t_{n+1}.

Chapter 2 gives a detailed analysis on procedures of the Lagrangian solver.

4.4 Evolution of the Eulerian solution

Once we have evolved the Lagrangian solution from t_n to t_{n+1}, we can determine the boundary conditions for the Eulerian domain for t_{n+1}. In chapter 3, we have determined that to evolve the Eulerian solution, we require: (a) the initial velocity $\mathbf{u}$ distribution at t_n, and (b) the Dirichlet velocity boundary condition $\mathbf{u}$ at the boundary $\partial\Omega_E$ at the final step t_{n+1}.

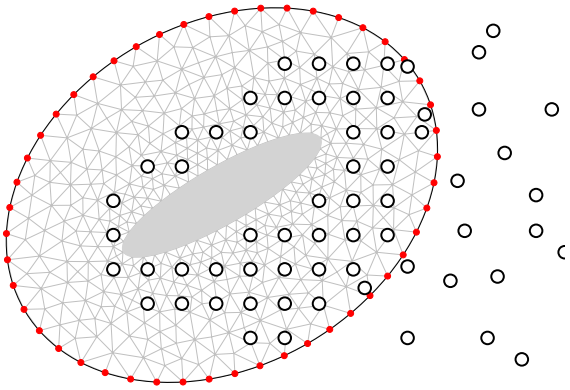


Figure 4.12: Dirichlet boundary conditions at boundary of Eulerian domain $\mathbf{u} \in \partial\Omega_E$. We evaluate the induced velocities from the Lagrangian solution at the nodes of the boundary [•, red dot].

4.4.1 Dirichlet boundary conditions

We can determine the Dirichlet velocity boundary condition at the Eulerian boundary $\partial\Omega_E$ from the Lagrangian vorticity field ω at t_{n+1}. In section 2.2.1, we derived that the discrete mollified velocity field of the vortex blobs. So, the velocity at the Eulerian boundary $\mathbf{x}_{bdry}$ is given an,

$$\mathbf{u}(\mathbf{x}_{bdry}, t_{n+1}) = \sum_p \mathbf{K}_\sigma[\mathbf{x}_{bdry} - \mathbf{x}_p(t_{n+1})]\alpha_p(t_{n+1}), \tag{4.22}$$

where $\mathbf{x}_{bdrt}$ are the nodal coordinates of the Eulerian dirichlet boundary $\partial\Omega_E$, as shown in figure 4.12.

4.4.2 Multi-step evolution

When coupling the Eulerian solver with the Lagrangian solver, we will see that the Eulerian time step size $\Delta t_E \leqslant \Delta t_L$. This is also the main benefit of the domain decomposition such that the wake region can be evolved with much larger step size. So we will have to perform k_E Eulerian sub-steps to reach the Lagrangian step t_{n+1},

$$t_k = t_n + k\Delta t_E, \tag{4.23}$$

where $k = 0, ..., k_E$ and k_E is given as

$$k_E = \frac{t_{n+1} - t_n}{\Delta t_E} = \frac{\Delta t_L}{\Delta t_E}. \tag{4.24}$$

When $k = 0$, we have $t_k = t_n$ and for $k = k_E$, we have $t_k = t_{n+1}$. We have a criterion that Δt_L must be multiple of Δt_E for an integer k_E. Figure 4.13 depicts the multi-stepping of the Eulerian solution from t_n to t_{n+1} to match the Lagrangian time. As the Eulerian solver requires boundary condition at each sub-step, we have to perform a interpolation of the boundary conditions for each sub-step t_k. We can perform a linear interpolation of the boundary condition for determines the boundary conditions at each sub-step,

$$\mathbf{u}(t_k) = \mathbf{u}(t_n) + k\Delta\mathbf{u}, \tag{4.25}$$

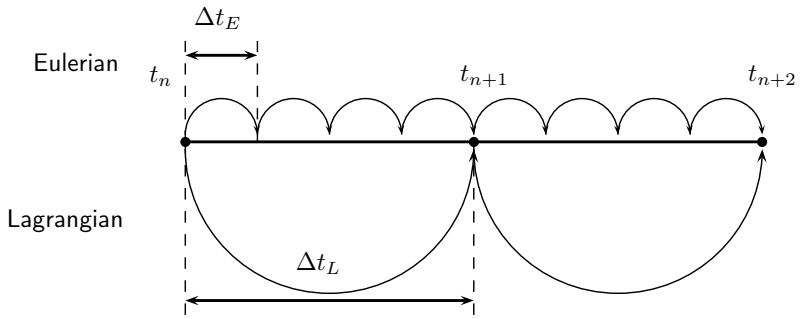


Figure 4.13: Eulerian multi-stepping to match the Lagrangian Δt_L. The figures shows $\Delta t_L = 4\Delta t_E$ and required $k_E = 4$ iterations to time march from t_n to t_{n+1}.

where $\Delta\mathbf{u}$ is given as

$$\Delta\mathbf{u} = \frac{\mathbf{u}(t_{n+1}) - \mathbf{u}(t_n)}{k_E}, \tag{4.26}$$

and is the gradient in velocity between each sub-step. We can summarized the feature of the evolution of the Eulerian solution as follows:

- The Eulerian solver uses a 1^{st} order Forward Euler time-marching scheme to evolution the solution from t_n to t_k.
- The solution is evolved k_E steps to reach t_{n+1}.
- We use velocity-pressure $\mathbf{u} - p$ formulation for the solution in the Eulerian solver.
- At the end of the time-step t_{n+1}, the Eulerian solver will have a higher resolved solution of the wall-region in comparison to the Lagrangian solver.

The modified correction strategy is iterated until we reach the desired time t.

Chapter 3 gives a detailed analysis on procedures of the Eulerian solver.

4.5 Introduction to pHyFlow: Hybrid solver

We have implemented the algorithms described in chapter 2, 3, and 4 into `pHyFlow`, an acronym for **p**ython **Hy**brid **Flow** solver. `pHyFlow` functions a fluid dynamics computational library in python, that has implemented the Eulerian solver, the Lagrangian solver (without vorticity diffusion of panels). These solver can used as a standalone solver (for test purposes), or can be coupled together to make the Hybrid solver.

The features of `pHyFlow` can be summarized as follows:

- `pHyFlow` is a hybrid flow solver that uses Hybrid Eulerian-Lagrangian Vortex Particle Method to couple the Navier-Stokes grid solver and a vortex blob solver.
- The algorithms are written in Python , Cython [4], C, C++, and CUDA C/C++ for efficiency. All the high-level algorithms such as definition of the problem, coupling of the solver, convection and diffusion of the problem is implemented in

PYTHON . The low-level algorithms such as remeshing kernel and saving routine calculations are written in the computationally efficient languages: CYTHON, C, and C++. The parallelizable routine such as calculation of the induced velocity of the vortex blobs is written in CUDA C/C++ for the NVIDIA GPU hardware.

- `pHyFlow` uses several open-source libraries: FEniCS [44], Fenicstools [47], Scipy [35], Numpy [63], mpi4py [24], pyUblas [37], for performing the calculations; and PyVTK [37], H5py [16], Matplotlib [33] for plotting and efficient data storage.
- `pHyFlow` is maintained, and is available at the bitbucket online repositiory `https://bitbucket.org/apalha/phyflow2.0`.

4.5.1 Program structure

The `pHyFlow` library serves as a computing environment for PYTHON programming language, where one could solve the hybrid flow problems. To achieve this, we have implemented an Eulerian solver, and a Lagrangian solver (without panel diffusion scheme), which can be used as a standalone solver for verification and validation. The `pHyFlow` library is structured into several modules, categorized by their purposes. In each `module`, we defined a `class` that handles the functions in the module. To add flexibility in computation, we added an `option` file where the user change the solver options. Figure 4.14 shows the structure of the `pHyFlow` library, classified using a color code. The structure of the `pHyFlow` is as follows:

- `IO`: This module contains all the input/output function for saving and plotting data. The `File` class handles the functions of the `IO` module.
- `aux`: This module contains all the auxiliary function of the library that does not belong to the fluid dynamics computation.
- `cpp`: The module that contains all the low-level compiled function that has been wrapped using binding generator for the use in python. The module contains the two main low-level algorithms for performing the induced velocity calculations for vortex blobs and vortex panels, and the remeshing algorithm for the vortex blobs.
- `blobs`: This module contains all the vortex blob operations. The module contains the class `Blobs`, an the vortex blob solver object handling the all the vortex blobs operations. Algorithms of the vortex blobs defined in chapter 2 is implemented in this module.
- `panels`: This module contains all the vortex panel operations and is wrapped in the class `Panels`, an the Panel method solver object. Algorithms of the vortex panel defined in chapter 2 is implemented in this module
- `lagrangian`: This module contains all the vortex blob and vortex panel coupling function and wrapped in the `LagrangianSolver` class and containing all the high-level function for managing the Lagrangian solver. THe vortex panel, vortex blob coupling algorithm described in chapter 2 is implemented in this module.

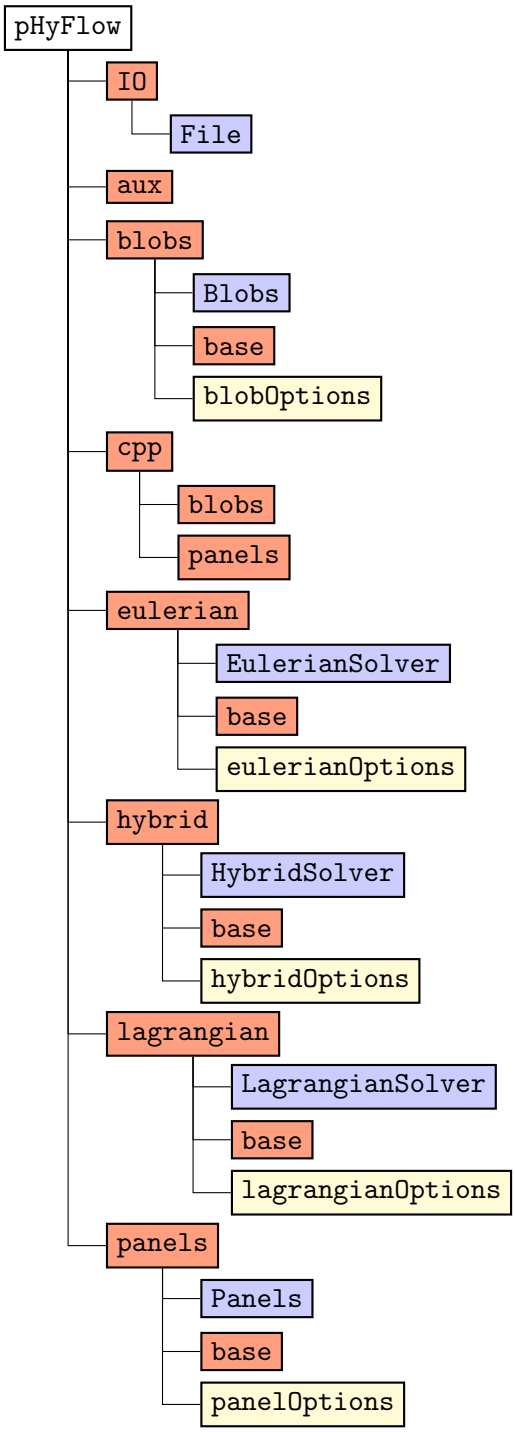


Figure 4.14: Flowchart of the pHyFlow library structured into modules, option script files, and classes.

- `eulerian`: This module contains all the Navier-Stokes grid operations and wrapped in the `EulerianSolver` class, that containing all the high-level function for defining and managing the Eulerian solver. Algorithms explained in chapter 3 is implemented in this module.
- `hybrid`: This module contains all the functions related to coupling of the Lagrangian and the Eulerian solver, summarized in section 4.2, 4.3, and 4.4. The functions are wrapped in the `HybridSolver` class and manages the global coupling process.

Figure 4.14 shows the structure of the `pHyFlow` library and is categorized into several modules of different purposes. It is structured in this manner such that one could employ the library for any general simulation purposes such as for hybrid case, or for non-hybrid cases (e.g. potential flow using vortex panels, full Eulerian grid simulation using Eulerian solver). This means that one could use a single module of `pHyFlow` library for the desired test case.

Hybrid class Hierarchy

However, the hybrid module relies on the functions of the Lagrangian module and the Eulerian module. Moreover, the Lagrangian module requires the function of vortex blob module and the vortex panel module. Therefore, the hierarchy of the hybrid class is defined in a different manner, as shown in figure 4.15.

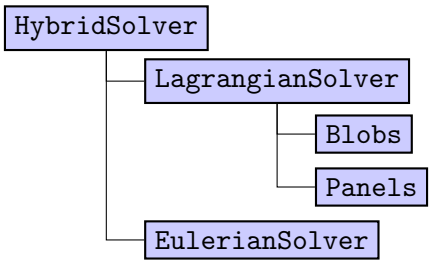


Figure 4.15: Flowchart of the `HybridSolver` hierarchy. The `HybridSolver` couples the `LagrangianSolver` class and the `EulerianSolver` class using the hybrid coupling schemes.

We use a bottom-up approach to construct the `HybridSolver` object, starting the lower-level objects: `Blobs`, `Panels` and then constructing the mid-level object: `LagrangianSolver`, and `EulerianSolver`, and finally constructing the highest-level object: `HybridSolver`. The procedure of constructing the hybrid class is as follows:

1. Construct the lowest-level objects:
   (a) Construct the `Blobs` object using the vorticity field parameters, the vortex blob parameters, time step parameters, and population control parameters.
   (b) Construct the `Panels` object using panel geometry parameters.
2. Construct the mid-level solvers:
   (a) Construct `LagrangianSolver` object using the vortex blob object `Blobs` and the vortex panel object `Panels`.

   (b) Construct `EulerianSolver` object using the geometry mesh file, interpolation probe grid parameters, and the fluid parameters.

3. Construct the hybrid solver:

   (a) Construct `HybridSolver` object using the Lagrangian solver object `LagrangianSolver`, the Eulerian solver object `EulerianSolver`, and the interpolation parameters.

A detailed description of the parameters required for the construction of the objects, and the schematic of these objects are given in appendix A.

Chapter 5

Implementation of Hybrid Eulerian-Lagrangian Vortex Particle Method

Chapter 6

Verification and Validation of Hybrid Method

This chapter focuses on the verification and the validation of the hybrid method. To perform this feat, we investigated several test-cases: Lamb-Oseen Vortex at $Re = 1000$, Clercx-Bruneau Dipole collision at $Re = 625$, Impulsively Started Cylinder at $Re = 1000$ and the flow around an elliptical airfoil at $Re = 5000$.

The verification of `pHyFlow` was perform as a start where we used the analytical solution of the Lamb-Oseen vortex to verify the velocity and the vorticity field. The Lamb-Oseen vortex problem was also essential for investigating the influences of the solver parameters that impact the accuracy of the coupling.

The validation of the accuracy of the hybrid solver was performed, once we verified the proper implementation of the hybrid solver. Clercx-Bruneau dipole collision test case was used to investigate the generation of the vorticity in the hybrid scheme and the transfer of this vorticity. This was the first test-cases, where we could confirm the implementation of the vortex panel for the hybrid scheme.

6.1 Lamb-Oseen Vortex Evolution

The Lamb-Oseen Vortex test case simulates the evolution of a laminar vortex core in an unbounded domain. In section 2.6.2, we used this test case to verify and validate the implementation of the vortex blobs of the Lagrangian solver and in section 3.4.1, we used it to verify the implementation of the Eulerian solver. Therefore, in a similar fashion we will employ this test case to verify the coupling of the hybrid solver.

The unbounded nature of the problem helps us to neglect the influence of the solid boundary (i.e the wall). Therefore, this test case does not require the panel solver in the Lagrangian solver as we are only concerned with the coupling of the vortex blobs to the Eulerian solver. Thus, we can primarily focus of the vorticity field interpolation error

Table 6.1: Summary of the parameters for the Lamb-Oseen vortex evolution. Parameters tabulated below are used for benchmark case.

Parameters	Value	Unit	Description
Γ_c	1	$\mathrm{m^2\,s^{-1}}$	Core strength
Ω	$[-0.5, 0.5] \times [-0.5, 0.5]$	m	Eulerian domain bounds
ν	0.001	$\mathrm{kg\,s^{-1}\,m^{-1}}$	Kinematic viscosity
τ	100	s	Initial time
Ov	1	-	Overlap ratio
h	0.01	m	Nominal blob spacing
Γ_{thres}	$(1 \times 10^{-14}, 1 \times 10^{-14})$	-	Population Control threshold
h_{grid}	0.007 to 0.016	m	FE cell diameter span
N_{cells}	26448	-	Number of mesh cells
Δt_L	0.001	s	Lagrangian time step size
Δt_E	0.001	s	Eulerian time step size
k_E	1	-	Eulerian sub-steps
$N_{\text{t-steps}}$	1000	-	Number of time integration steps
t	0 to 1	s	Simulation time span
d_{bdry}	$2 \cdot h$	m	Interpolation boundary offset

discussed in section 4.2.2, and quantitatively present the importance of ensuring conservation of circulation. This is the primary purpose of employing the Lamb-Oseen Vortex test case.

The secondary purpose is to quantify the influences of the discretization on the accuracy of the coupling. A parameter sensitivity analysis was therefore performed to determine their effects on the coupling error. The parameters that determine the spatial discretization of the vortex blobs is nominal particle spacing h, and the overlap ratio Ov (see figure 2.3). The spatial discretization of the Eulerian solver is regarded as a control variable for this test case as its impact was concluded in section 3.4.1. The parameters that determine the temporal discretization of the hybrid method is the time step size of the Eulerian solver Δt_E and the time step size of the Lagrangian solver Δt_L which are depended according to equation 4.24 where k_E is the number of Eulerian sub-steps.

The coupling error was quantified my determining the growth of maximum relative error in vorticity ϵ given by equation 2.72, approach used in section 2.6.2 and section 3.4.1.

6.1.1 Problem Definition

The Lamb-Oseen Vortex problem is defined by the vorticity field, equation 2.68, and the velocity field, equation 2.70. The hybrid solver is initialized by first assigning the strengths of the vortex blobs using equation 2.71. The Eulerian domain Ω_E is then initialized using the solution of the Lagrangian solver. Daeninck [23] used this approach to enhance the coupling between the methods ensuring minimum interpolation error.

Figure 6.1 shows the Hybrid domain configuration for the Lamb-Oseen Vortex problem with the Lagrangian domain Ω_L spanning the full fluid domain. The Eulerian domain

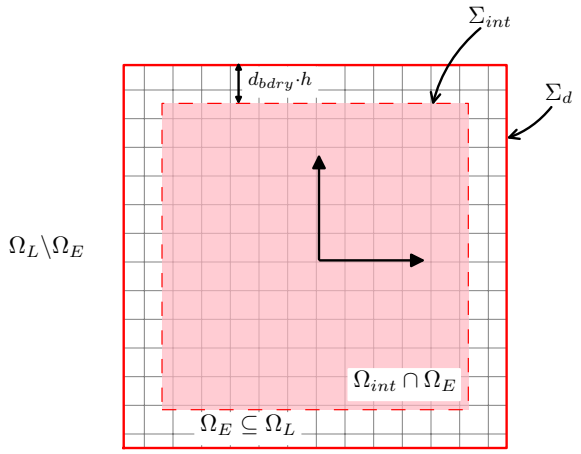


Figure 6.1: The domain decomposition for the Lamb-Oseen vortex problem, $\Omega_E \subseteq \Omega_L$. The Eulerian domain bounds $\Omega_E = [-1, 1] \times [-1, 1]$ with Dirichlet boundary $\partial\Omega_{dirichlet}$ [—, solid red] (*not to scale*).

Ω_E only resolves the center of the Lamb-Oseen core, $\Omega_E \subseteq \Omega_L$. The domain bounds $[-0.5, -0.5] \times [-0.5, -0.5]$ with a Dirichlet velocity boundary Σ_d where the velocity boundary condition is applied as described in section 4.4.1. The correction of the Lagrangian domain is performed in the interpolation domain Ω_{int} according to the procedures described in section 4.2.

The spatial discretization of the Eulerian domain Ω_E is regarded as the control variable. Therefore, the parameter sensitivity analysis is performed by varying the spatial discretization of the Lagrangian method. The Eulerian domain is discretized with an unstructured mesh formulation using GMSH (see section 3.2.2) having $N_{cells} = 226448$ unstructured cells and grid size h_{grid} spanning 0.007 to 0.0016.

The Lamb-Oseen Vortex problem is defined according to the parameters tabulated in table 6.1. The core is located at $(0, 0)$, where the Eulerian domain Ω_E is centered. The parameters are chosen such that vorticity ω and velocity $\mathbf{u}$ is non-zero at the boundary of the Eulerian domain Σ_d, figure 6.1.

The evolution of the Lagrangian solver and the Eulerian solver is performed according to section 4.3 and 4.4 respectively. The Lagrangian solver performs TRS for diffusion of the vortex blobs, see section 2.4.1. The scheme requires vortex blob redistribution at every step, $f_{redis} = 1$. In conjunction with the redistribution, the population control is performed at every step, $f_{pc} = 1$ with Γ_{thre} in table 6.1.

6.1.2 Results and Discussion

The investigation of the Lamb-Oseen vortex problem is divided into three parts. The first part of the investigation concerns with comparing several stages of the hybrid coupling, section 6.1.3, where we compare the uncoupled scheme with the one-way coupled scheme and fully coupled scheme. These successive coupling investigate will help determine the source and quantify the error of the coupling. The second part of the investigation, section 6.1.3 focuses on importance of conservation of circulation that was discussed in

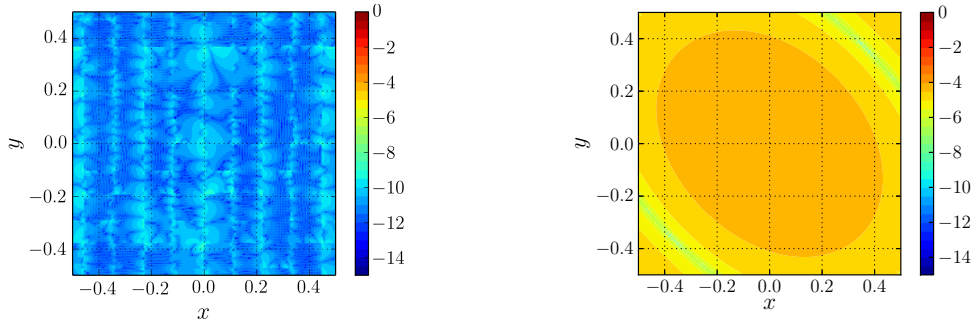
section 4.2.3. The results of the non-conserved and conserved scheme are compared to conclude the importance of conservation of circulation. During these two investigations, the parameters tabulated in table 6.1 are used.

The third and final investigation is dedicated to the parameter sensitivity analysis, section 6.1.3. Parameters that determine the spatial and temporal discretization of the scheme is investigated to verify the convergence of scheme.

6.1.3 Uncoupled vs. One-way Coupled vs. Fully Coupled

To verify the implementation of the hybrid algorithm, we compared several stages of the hybrid coupling with the standard, fully Eulerian test case. The three types of the coupling are as given:

- **Uncoupled**: The uncoupled test case involves only Eulerian solver and serves as a benchmark to quantify the error in coupling. The boundary conditions are determined directly from the analytical formulation, equation 3.34.
- **One-way coupled**: The one-way coupled test case is a partially coupled hybrid test case where the Eulerian method is evolved using the Lagrangian solution. The correction of the Lagrangian solution is not performed in this scenario. Thus, this case will help us determine the error in evolution Eulerian method using the Lagrangian solution.
- **Fully coupled**: The fully coupled test case performs the full coupling strategy described in section 4.2.3. The Eulerian method is evolved using the Lagrangian solution and the Lagrangian solution in the interpolation domain Ω_{int}, figure 6.1 is corrected at the end of each time step. This test case will help us quantify the error in transferring the Eulerian solution to the Lagrangian method.



(a) Velocity **(b)** Vorticity

Figure 6.2: Initial relative error at $t = 0$ inside the Eulerian domain. The figure depicts **(a)** the relative error in velocity $\mathbf{u}$ and **(b)** the relative error in vorticity ω.

Figure 6.2 depicts the initial relative error in velocity and vorticity inside the Eulerian domain Ω_E. The relative error in velocity is near machine epsilon $\epsilon \leq 10 \times 10^{-8}$, but the error in vorticity is in the order 10×10^{-5}. Similar observation was made in section 3.4.1

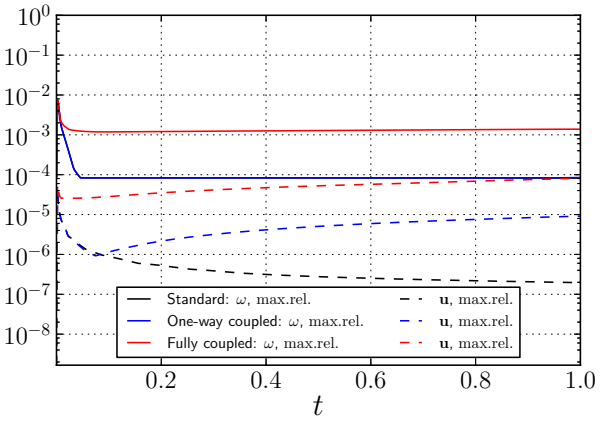


Figure 6.3: Comparison of the evolution of the maximum relative error from $t = 0$ to $t = 1$. The figure compares standard case (**black**) vs. the one-way coupled case (**blue**) vs. the fully coupled case (**red**). The plot depicts maximum relative error in velocity (dashed), and the maximum relative error in vorticity (solid).

and arises from the projection error when determining the vorticity from velocity. The error is dependent on the type of discretization and we have second-order velocity space and first-order vorticity space.

The simulation is evolved from $t = 0$ to $t = 1$ with $N_{t-steps} = 1000$ Lagrangian and Eulerian time steps using the time step parameters tabulated in table 6.1. Figure 6.3 shows the evolution of maximum relative error in vorticity ω and velocity $\mathbf{u}$ of the uncoupled, one-way coupled and the fully coupled cases in the Eulerian domain Ω_E w.r.t. the analytical solution, equation 2.68. The initial observation shows the error in velocity is two to three orders of magnitude less than the error in vorticity and occurs due to the projection error. The figure shows that the uncoupled scheme has the lowest error in vorticity and velocity. As the boundary condition is directly obtained from the analytical solution, the error only arises from FE discretization of the Eulerian method. As time progresses, the error in velocity converges around 10×10^{-7} and the error in vorticity converges around 10×10^{-4}.

The one-way coupled case shows an increase in the velocity field inside the Eulerian domain Ω_E. The increase in error is negligible at the initial stages of the coupling. This states that the discretization error of the analytical solution is well represented using the vortex blobs. At $t = 1$, the error in velocity increases by two orders of magnitude from 10×10^{-7} to 10×10^{-5}. This implies that the error is due to the growth in error of the Lagrangian field. In chapter 2, we observed there is an increase in the error due to the circulation processing techniques (population control, remeshing) and the time-marching of the vortex blobs.

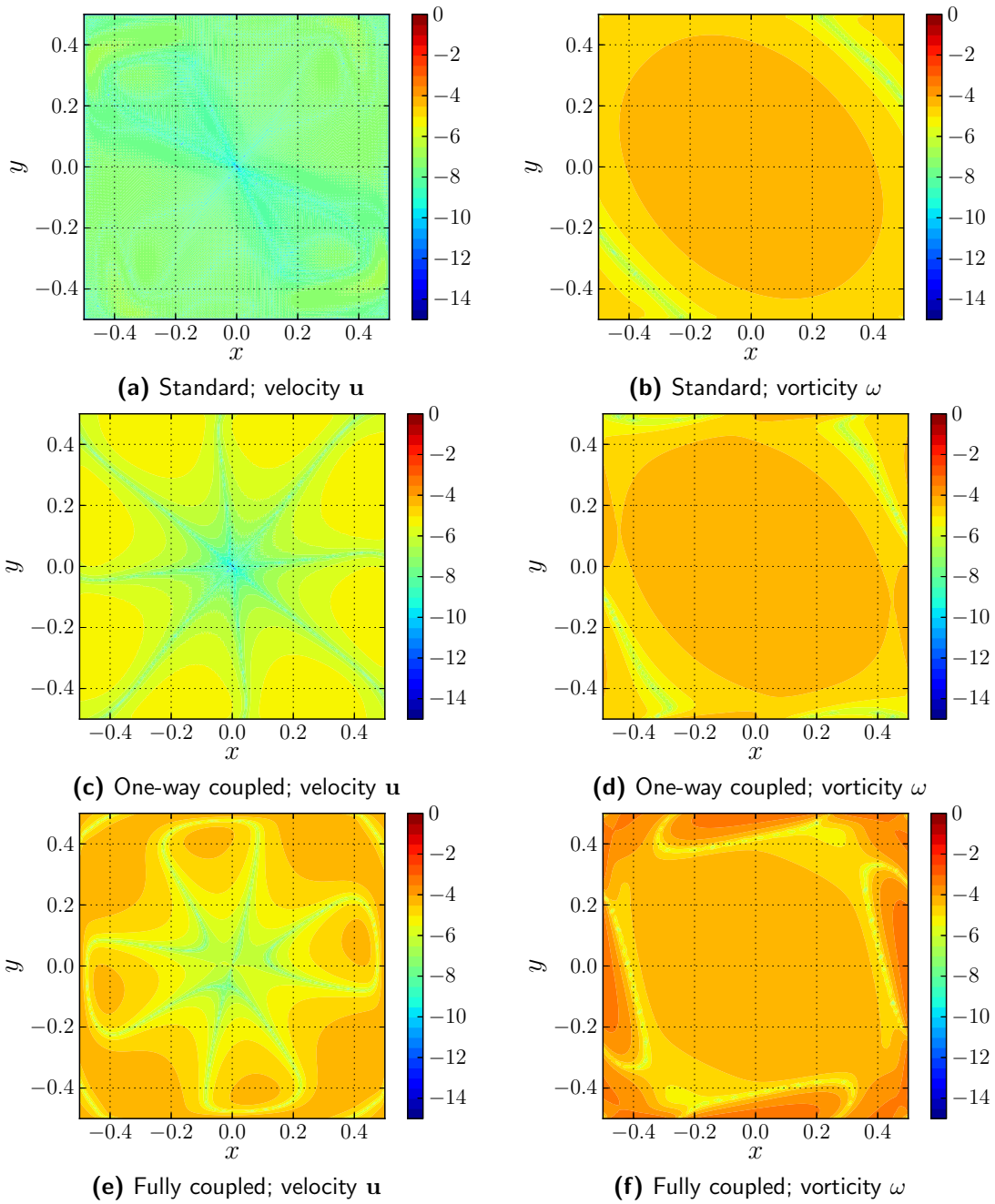
The fully coupled case demonstrates that there is further increase in the error in velocity. Moreover, we observe that the there is now an increase in error in vorticity as well. As we are transfer the discrete vorticity field from the Eulerian method to the Lagrangian method it causes an increase in error of the Lagrangian solution. The consequence of this is that now the velocity field resolved by the Lagrangian method has an addition error. The figure depicts this effect, showing an increase in error in velocity. After the first few

iteration (around $t = 0.2$), there is a measurable difference between the fully coupled and the one-way coupled case.

Conservation of circulation

Parameter sensitiviy analysis

6.1.4 Conclusion



(a) Standard; velocity $\mathbf{u}$

(b) Standard; vorticity ω

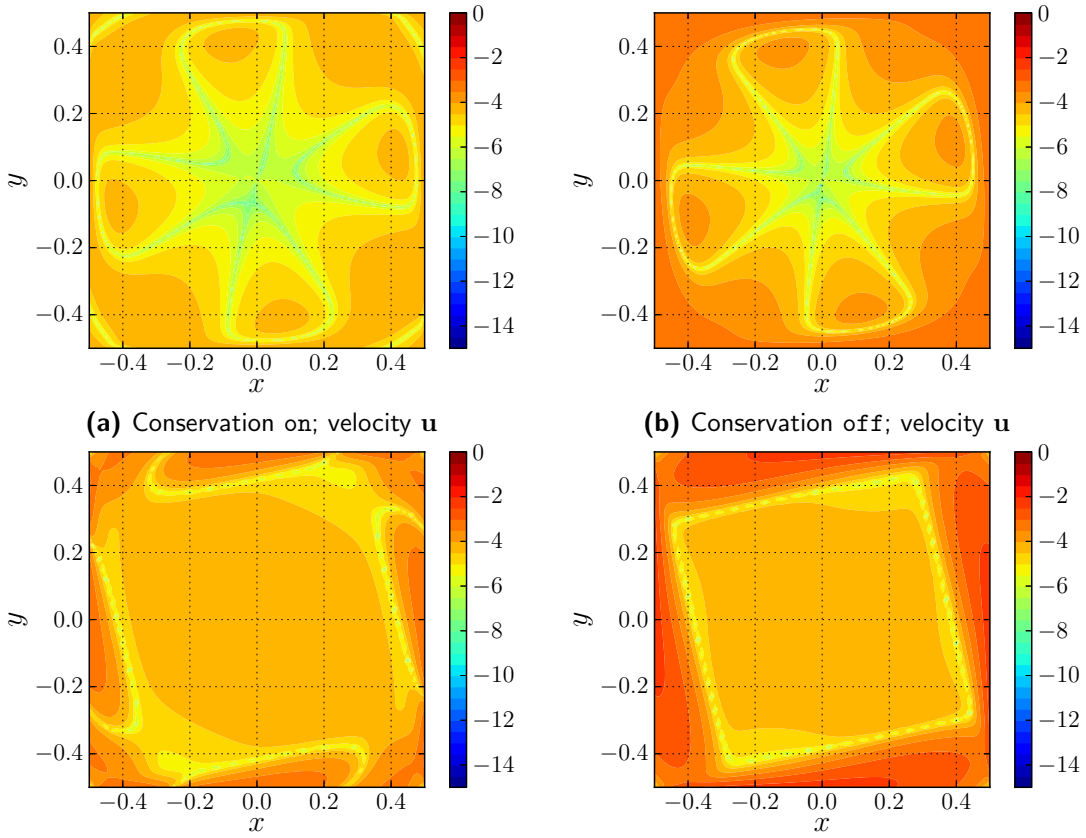
(c) One-way coupled; velocity $\mathbf{u}$

(d) One-way coupled; vorticity ω

(e) Fully coupled; velocity $\mathbf{u}$

(f) Fully coupled; vorticity ω

Figure 6.4: Initial relative error at $t = 0$ inside the Eulerian domain. The figure depicts **(a)** the relative error in velocity $\mathbf{u}$ and **(b)** the relative error in vorticity ω.



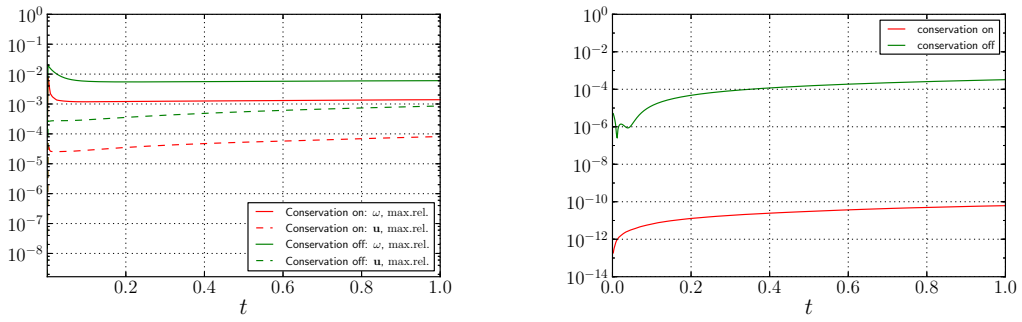
(a) Conservation `on`; velocity $\mathbf{u}$

(b) Conservation `off`; velocity $\mathbf{u}$

(c) Conservation `on`; vorticity ω

(d) Conservation `off`; vorticity ω

Figure 6.5: Initial relative error at $t = 0$ inside the Eulerian domain. The figure depicts **(a)** the relative error in velocity $\mathbf{u}$ and **(b)** the relative error in vorticity ω.



(a) Relative Error in velocity and vorticity

(b) Error in total circulation Γ

Figure 6.6: Initial relative error at $t = 0$ inside the Eulerian domain. The figure depicts **(a)** the relative error in velocity $\mathbf{u}$ and **(b)** the relative error in vorticity ω.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

7.1.1 Lagrangian domain

7.1.2 Eulerian domain

7.1.3 Hybrid method

7.2 Recommendations

7.2.1 Lagrangian domain

7.2.2 Eulerian domain

7.2.3 Hybrid method

References

[1] Barba, L. A., and Rossi, L. F. Global field interpolation for particle methods. *Journal of Computational Physics 229*, 4 (2010), 1292–1310.

[2] Barba, L. A. L. Vortex method for computing high-Reynolds number flows: Increased accuracy with a fully mesh-less formulation.

[3] Beale, J. On the Accuracy of Vortex Methods at Large Times. In *Computational Fluid Dynamics and Reacting Gas Flows SE - 2*, B. Engquist, A. Majda, and M. Luskin, Eds., vol. 12 of *The IMA Volumes in Mathematics and Its Applications*. Springer New York, 1988, pp. 19–32.

[4] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. Cython: The best of both worlds. *Computing in Science and Engineering 13*, 2 (2011), 31–39.

[5] Boffi, D. Three-Dimensional Finite Element Methods for the Stokes Problem. *SIAM Journal on Numerical Analysis 34*, 2 (Apr. 1997), 664–670.

[6] Braza, M., Chassaing, P., and Ha Minh, H. Numerical Study and Physical Analysis of the Pressure and Velocity Fields in the Near Wake of a Circular Cylinder. *Journal of Fluid Mechanics 165* (1986), 79–130.

[7] Brenner, S. C., and Scott, L. R. *The Mathematical Theory of Finite Element Methods*. Texts in applied mathematics. Springer-Verlag, 2002.

[8] Brezzi, F., and Fortin, M. *Mixed and hybrid finite elements methods*. Springer series in computational mathematics. Springer-Verlag, 1991.

[9] Carey, G. F. *Computational Grids: Generations, Adaptation & Solution Strategies*. CRC Press, 1997.

[10] Chang, C. C., and Chern, R. L. Numerical study of flow around an impulsively started circular cylinder by a deterministic vortex method. *Journal of Fluid Mechanics 233* (1991), 243–263.

[11] Chen, Z. *The Finite Element Method: Its Fundamentals and Applications in Engineering*. World Scientific, 2011.

[12] Chorin, A. J. Numerical solution of the Navier-Stokes equations. *Mathematics of computation 22*, 104 (1968), 745–762.

[13] Chorin, A. J. Numerical Study of Slightly Viscous Flow. *Journal of Fluid Mechanics 57*, Part 4 (1973), 785–796.

[14] Ciarlet, P. G., and Raviart, P. A. General Lagrange and Hermite interpolation in $\mathbf{R}^n$ with applications to finite element methods. *Archive for Rational Mechanics and Analysis 46*, 3 (1972), 177–199.

[15] Clercx, H., and Bruneau, C.-H. The normal and oblique collision of a dipole with a no-slip boundary. *Computers & Fluids 35*, 3 (Mar. 2006), 245–279.

[16] Collette, A. *Python and HDF5*. O'Reilly Media, 2013.

[17] Commons, W. Darrieus windmill, 2007.

[18] Commons, W. Windmills d1-d4 (thornton bank), 2008.

[19] Cooper, C. D., and Barba, L. A. Panel-free boundary conditions for viscous vortex methods. In *19th AIAA Computational Fluid Dynamics Conference* (Dept. of Mechanical Engineering, Universidad Técnica F. Santa María, Casilla 110-V, Valparaíso, Chile, 2009).

[20] Cottet, G.-H., Koumoutsakos, P., and Salihi, M. L. O. Vortex Methods with Spatially Varying Cores. *Journal of Computational Physics 162*, 1 (July 2000), 164–185.

[21] Cottet, G. H., and Koumoutsakos, P. D. *Vortex Methods: Theory and Practice*, vol. 12. Cambridge University Press, 2000.

[22] Courant, R. Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc 49*, 1 (1943), 1–23.

[23] Daeninck, G. *Developments in Hybrid Approaches: Vortex Method with Known Separation Location; Vortex Method with Near-Wall Eulerian Solver; RANS-LES Coupling*. PhD thesis, Université Catholique de Louvain, Belgium, 2006.

[24] Dalcín, L., Paz, R., Storti, M., and D'Elía, J. MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing 68*, 5 (2008), 655–662.

[25] Degond, P., and Mas-Gallic, S. The Weighted Particle Method for Convection-Diffusion Equations. Part 1: The Case of an Isotropic Viscosity. *Mathematics of Computation 53*, 188 (Oct. 1989), 485–507.

[26] Dixon, K., Simão Ferreira, C. J., Hofemann, C., Van Bussel, G., and Van Kuik, G. A 3D unsteady panel method for vertical axis wind turbines. In *European Wind Energy Conference and Exhibition 2008* (2008), vol. 6, pp. 2981–2990.

[27] Franklin, W. R. Pnpoly-point inclusion in polygon test. http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html, 2006.

[28] Geuzaine, C., and Remacle, J.-F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering 79*, 11 (2009), 1309–1331.

[29] Goda, K. A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows. *Journal of Computational Physics 30*, 1 (1979), 76–95.

[30] Guermond, J. L., and Lu, H. A domain decomposition method for simulating advection dominated, external incompressible viscous flows. *Computers & Fluids 29*, 5 (June 2000), 525–546.

[31] Guermond, J. L., Minev, P., and Shen, J. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering 195*, 44-47 (2006), 6011–6045.

[32] Guermond, J. L., and Shen, J. A new class of truly consistent splitting schemes for incompressible flows. *Journal of Computational Physics 192*, 1 (2003), 262–276.

[33] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (2007), 90–95.

[34] Johnston, H., and Liu, J.-G. Accurate, stable and efficient Navier-Stokes solvers based on explicit treatment of the pressure term. *Journal of Computational Physics 199*, 1 (2004), 221–259.

[35] Jones, E., Oliphant, T., Peterson, P., et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2014-07-16].

[36] Katz, J., and Plotkin, A. *Low-speed aerodynamics*. Cambridge Aerospace Series. Cambridge University Press, 2001.

[37] Kloeckne, A. PyUblas: Hybrid numerical codes in Python and C++ integrate numpy and Boost.Ublas, 2008–. [Online; accessed 2014-07-16].

[38] Koumoutsakos, P. *Direct numerical simulations of unsteady separated flows using vortex methods*. PhD thesis, California Institute of Technology, USA, 1993.

[39] Koumoutsakos, P., and Leonard, A. High-resolution simulations of the flow around an impulsively started cylinder using vortex methods. *Journal of Fluid Mechanics 296* (1995), 1–38.

[40] Koumoutsakos, P., Leonard, A., and Pépin, F. Boundary Conditions for Viscous Vortex Methods. *Journal of Computational Physics 113*, 1 (July 1994), 52–61.

[41] Lamb, H. *Hydrodynamics*. Cambridge university press, 1993.

[42] Lecointe, Y., and Piquet, J. On the use of several compact methods for the study of unsteady incompressible viscous flow round a circular cylinder. *Computers and Fluids 12*, 4 (1984), 255–280.

[43] Logg, A. NSBench in Launchpad. https://launchpad.net/nsbench, 2014.

[44] Logg, A., Mardal, K.-A., Wells, G. N., et al. *Automated Solution of Differential Equations by the Finite Element Method*, vol. 84. Springer, 2012.

[45] Logg, A., and Wells, G. N. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software 37*, 2 (2010).

[46] Monaghan, J. Extrapolating B-splines for interpolation. *Journal of Computational Physics 60*, 2 (Sept. 1985), 253–262.

[47] Mortensen, M. Fenicstools. https://github.com/mikaem/fenicstools, 2014.

[48] Nair, M. T., and Sengupta, T. K. Unsteady flow past elliptic cylinders. *Journal of Fluids and Structures 11*, 6 (1997), 555–595.

[49] Ould-Salihi, M. L., Cottet, G. H., and El Hamraoui, M. Blending Finite-Difference and Vortex Methods for Incompressible Flow Computations. *SIAM Journal on Scientific Computing 22*, 5 (Jan. 2001), 1655–1674.

[50] RAO, S. S. *The Finite Element Method in Engineering*. Elsevier Science, 2011.

[51] Reed, W., and Hill, T. R. Triangular Mesh Method for the Neutron Transport Equation. 10–31.

[52] Rosenfeld, M., Kwak, D., and Vinokur, M. A fractional step solution method for the unsteady incompressible Navier-Stokes equations in generalized coordinate systems. *Journal of Computational Physics 137* (1991), 102–137.

[53] Shankar, S., and Dommelen, L. A New Diffusion Procedure for Vortex Methods. *Journal of Computational Physics 127*, 1 (Aug. 1996), 88–109.

[54] Shiels, D. *Simulation of controlled bluff body flow with a viscous vortex method*. PhD thesis, California Institute of Technology, USA, 1998.

[55] Simão Ferreira, C. J. *The near wake of the VAWT: 2D and 3D views of the VAWT aerodynamics*. PhD thesis, Delft University of Technology, Netherlands, 2009.

[56] Simão Ferreira, C. J., Bijl, H., van Bussel, G., and van Kuik, G. Simulating Dynamic Stall in a 2D VAWT: Modeling strategy, verification and validation with Particle Image Velocimetry data. *Journal of Physics: Conference Series 75*, 1 (July 2007), 012023.

[57] Simão Ferreira, C. J., Kuik, G., van Bussel, G., and Scarano, F. Visualization by PIV of dynamic stall on a vertical axis wind turbine. *Experiments in Fluids 46*, 1 (Aug. 2008), 97–108.

[58] Speck, R. *Generalized Algebraic Kernels and Multipole Expansions for massively parallel Vortex Particle Methods.* PhD thesis, University of Wuppertal, Germany, 2011.

[59] Stock, M. J., Gharakhani, A., and Stone, C. P. Modeling rotor wakes with a hybrid OVERFLOW-vortex method on a GPU cluster. In *28th AIAA Applied Aerodynamics Conference* (2010).

[60] Taylor, C., and Hood, P. A numerical solution of the Navier-Stokes equations using the finite element technique. *Computers & Fluids 1*, 1 (Jan. 1973), 73–100.

[61] Tryggeson, H. *Analytical Vortex Solutions to the Navier-Stokes Equation.* PhD thesis, Växjö University, Sweden, 2007.

[62] Tutty, O. R. A Simple Redistribution Vortex Method (with Accurate Body Forces). *ArXiv e-prints* (Sept. 2010).

[63] Van Der Walt, S., Colbert, S. C., and Varoquaux, G. The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering 13*, 2 (2011), 22–30.

[64] Vermeer, L., Sø rensen, J., and Crespo, A. Wind turbine wake aerodynamics. *Progress in Aerospace Sciences 39*, 6-7 (Aug. 2003), 467–510.

[65] Wee, D., and Ghoniem, A. F. Modified interpolation kernels for treating diffusion and remeshing in vortex methods. *Journal of Computational Physics 213*, 1 (2006), 239–263.

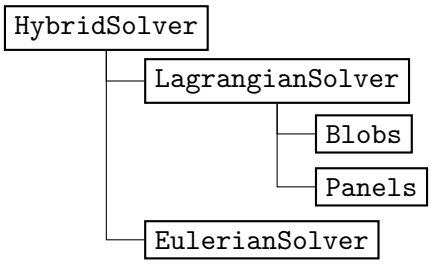
[66] Winckelmans, G., Cocle, R., Dufresne, L., and Capart, R. Vortex methods and their application to trailing wake vortex simulations. *Comptes Rendus Physique 6*, 4-5 (May 2005), 467–486.

Appendix A

pHyFlow Code Structure

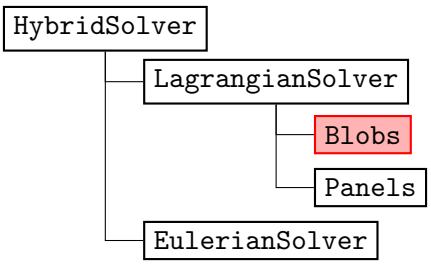
The document outlines the pHyFlow code structure. The pHyFlow functions are organized into several classes. The functions related to the vortex particles are placed inside the Blobs class. The functions related to the panel problem are inside Panels class. The LagrangianSolver class is made to couple the functions of the vortex blobs and the vortex panel together. The functions of the Eulerian domain are placed inside the EulerianSolver class, where the Navier-stokes grid problem is solved. Finally, coupling of all the problems are done with the HybridSolver class. Note, all the classes are capable of handling multi-body / multi-domain problem within them and LagrangianSolver class and the HybridSolver class only couples methods together.

pHyFlow Structure:

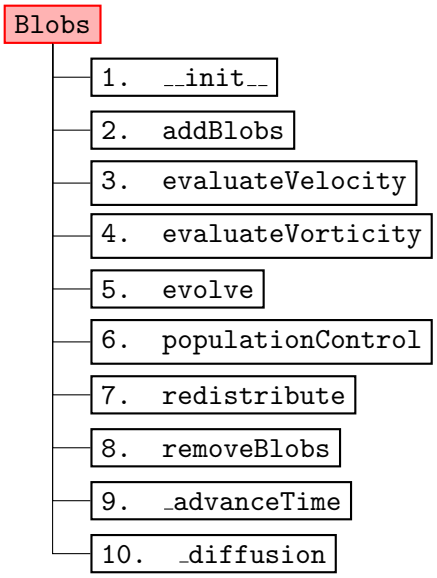


Blobs Class

The main structure of the Blobs class. This class contains all the function related to the calculation of the vortex blobs.



Class structure:



Attributes:

Attributes	Description
blobControlParams	The diffusion parameters. It is a dictionary containing all the parameters of the diffusion method used for the simulation. Contains: stepRedistribution, stepPopulationControl, gThresholdLocal, gThresholdGlobal.
computationMethod	computationMethod (tuple) with the type of Biot-Savart solver (direct, fmm) and the type of hardware to use (cpu, gpu).
deltaTc	The size of the convective time step Δt_c
deltaTd	The size of the convective time step Δt_d
diffusionParams	A dictionary containing all the parameters related to the computation of the diffusion step. Specifies the diffusion scheme and other specific parameters. Contains: method, c2.
g	The strength of the vortex blobs α.

`gThresholdGlobal`	Maximum value of variation of total vorticity due to the removal of blobs during population control.
`gThresholdLocal`	Minimum value of circulation to consider for each vortex blob when selecting blobs to remove during population control.
`h`	The size of the cell associated to the vortex blobs. Corresponds to the minimum spacing between the core of two neighboring cells. It is related to the core size of the blob, σ, and to the spacing h by the expression $Ov = h/\sigma$.
`integrationMethod`	`integrationMethod` (`fe`, `rk4`) the type of time integrator used: `fe` forward Euler, `rk4` Runge-Kutta 4^{th} order.
`nu`	The fluid kinematic viscosity, used to calculate the diffusion coefficient: `c2` and diffusion time step `deltaTd`, Δt_d.
`numBlobs`	The number of blobs.
`overlap`	The overlap ratio between neighboring blobs.
`plotVelocity`	A flag that defines if velocity is to be plotted or not.
`sigma`	The core size of the vortex blobs.
`stepDiffusion`	The frequency of diffusion steps.
`stepPopulationControl`	The frequency of population control.
`stepRedistribution`	The frequency of redistribution of blobs.
`timeIntegrationParams`	A dictionary containing all time integration parameters of the simulation. Contains the definition of the time integration scheme possibly additional parameters specific to the scheme.
`t`	The current time of the simulation.
`tStep`	The current time step of the simulation.
`velocityComputationParams`	A dictionary containing all the parameters related to the computation of induced velocities. Specifies computation scheme (direct or fmm) and hardware to use (cpu or gpu).
`vInf`	The free stream velocity.
`x`	The x coordinates of the vortex blobs.
`y`	The y coordinates of the vortex blobs.

Table A.1: Attributes of `Blobs` class and their description.

`__init__`

Description: Initialize the `Blobs` class with either the given input parameters or by a reading a `file` containing all the necessary parameters.

Input Parameters

File Name	Containing all the parameters to re-initalize the class.	
	— or —	
Parameters	Vorticity Field	: {`xBlob, yBlob, gBlob`} or {`wFunction, xBounds, yBounds`}
	Blob parameters	: `overlap, h`
	Time Step parameters	: `deltaTc, nu, stepRedistribution, integrationMethod, computationMethod`
	Population control parameters	: `stepPopulationControl, gThreshold`

Descriptions of the parameters:

Vorticity field		*Default*
`xBlob,yBlob`	: the x, y blob coordinates.	-
`gBlob`	: the circulation Γ_i associated to each of the vortex blobs.	-
	— *or* —	
`wExactFunction`	: the function that returns the exact value of vorticity ω at any given x, y coordinates.	
	Input parameters : `xEval,yEval`	
	Assigns : `-`	-
	Returns : `wEval`	
`xBounds, yBounds`	: the x, y bounds of the domain where the particles was originally distributed.	-

Blob parameters		*Default*
`overlap`	: the overlap ratio h/σ.	1.0
`h`	: the size of the cell h associated to the blobs. *Note:* Cells are square.	-

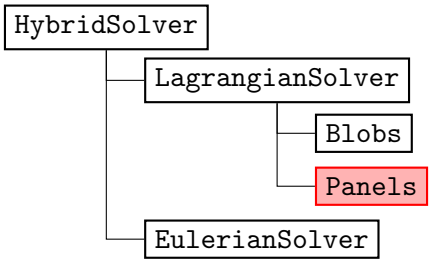
Time step parameters		*Default*
`deltaTc`	: the size of the convective time step Δt_c.	-
`nu`	: the fluid kinematic viscosity ν, used to calculate the diffusion coefficient c^2 and diffusion time step size ΔT_d.	-
`stepRedistribution`	: the redistribution step frequency.	1
`integrationMethod`	: the time integration method (`FE`: Forward euler , `RK4`: 4^{th} order Runge-Kutta).	RK4
`computationMethod`	: the calculation method to evolve the blobs, (`Direct`: Direct Method, `FMM`: Fast-Multipole Method) using (`CPU`, `GPU`).	{FMM, GPU}.

Population control parameters		*Default*
`stepPopulationControl`	population control step frequency	1.
`gThreshold`	: the tuple with minimum **and** maximum value of the circulation Γ_{min}.	-

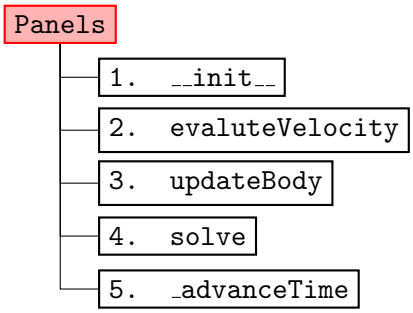
Free stream velocity		*Default*
`vInf`	: The free-stream velocity function, returning the velocity action on the vortex blobs.	-
	Input parameters : `t`	
	Assigns : `-`	-
	Returns : `vx,vy`	

`Panels` class

The main structure of the panel method class `Panels`. This class contains all the functions related to the calculation of panels.



Class structure:



Attributes:

Attributes	Description
`A`	The inter-induction matrix $\mathbf{A}$, the LHS of the problem.
`cmGlobal`	The global position vector for each of the $\mathbf{N}$ body, refining the position of the local panel $(0, 0)$ in the global coordinate system.
`deltaT`	The simulation time step size ΔT
`geometryKeys`	The dictionary containing all the parameters of the geometry. Contains: `xPanel` (the x coordinate of the $\mathbf{M}$ panel corners.), `yPanel` (The y coordinate of the $\mathbf{M}$ panel corners), `cmGlobal`, `thetaLocal`, `dPanel` (The off-set of the panel collocation point from the panel mid-point).
`nBodies`	The number of panel bodies.
`norm`	The x, y normal vector of each panel.
`normCat`	The global concatenated x, y component of the panel normal vector at each collocation points.
`nPanels`	The number of panels in each body/geometry.
`nPanelsTotal`	The total number of panels.
`panelKernel`	A string defining panel kernel type.
`problemType`	A string defining the panel problem is of a `moving` type or of a `fixed` type.
`solverCompParams`	The dictionary containing solver computation parameters.
`sPanel`	The vortex sheet strengths γ of $\mathbf{M}$ panels.
`t`	The current time t of the simulation.
`tang`	The x, y tangent vector of each panel.

tangCat	The global concatenated x, y component of the panel normal vector at each collocation points.
thetaLocal	The local rotation angle θ w.r.t to the local coordinate system. The rotational will be performed around the local reference point $(0, 0)$, i.e around the global center of rotation point cmGlobal.
tStep	The current step of the simulation.
velCompParams	A dictionary containing the velocity computation parameters, method and hardware.
xyCPGlobal	The global x, y coordinate of the panel collocation points.
xyCPGlobalCat	The global concatenated x, y coordinate of the panel collocation points.
xyPanelGlobal	The global x, y coordinate of the panel bodies.
xyPanelGlobalCat	The global concatenated x, y coordinate of the panel bodies.
xyPanelLocal	The local x, y coordinate of the panel bodies.

Table A.2: Attributes of Panels class and their description.

__init__

Input Parameters

File Name	Containing all the parameters to re-initalize the class.
Parameters	Panel coordinates : {xCP, yCP, xPanel, yPanel, cmGlobal, thetaLocal}
	External velocity : externVel

Description: Initialize the panels class with the given input parameters. In the case of a multibody problem, a list of panel coordinates can be given and internally it takes care of the inter-coupling.

Panel coordinates

xCP,yCP	: the local x, y-coordinates of the panel collocation points.
xPanel,yPanel	: the local coordinate of the panel edges. *Note*: Should have a closed loop (end with initial point coordinates).
cmGlobal	: the position of reference points of a given panel body.
thetaLocal	: the rotational angles of the panel body axes w.r.t to the global x-axis.

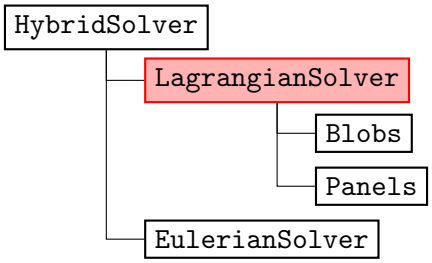
External velocity

externVel	: Reference to an external velocity **function** acting of the panels. For the panel case, the external velocity will the induced velocity of the blobs + freestream vortexBlob.evaluateVelocity.

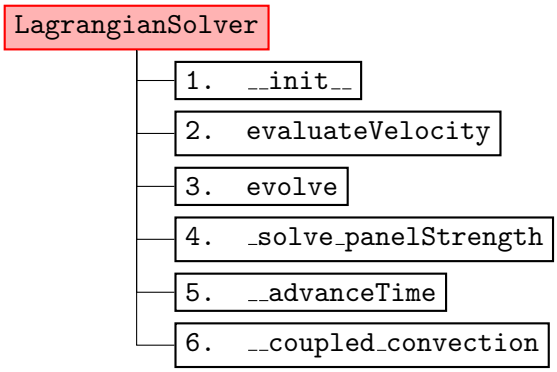
Input parameters : xCP,yCP
Assigns : -
Returns : vxCP,vyCP

LagrangianSolver Class

The main structure of the Blobs + Panels (LagrangianSolver) class. This class contains all the function related to the calculations of panel with vortex blobs.



Class structure:



Attributes:

Attributes	Description
deltaT	The inter-induction matrix **A**, the LHS of the problem.
gTotal	The total circulation of the Lagrangian domain.
t	The current time t of the simulation.
tStep	The current step of the simulation.
vInf	The x, y component of the free-stream velocity.
Blobs	The vortex blobs class Blobs.
Panels	The vortex panels class Panels.

Table A.3: Attributes of LagrangianSolver class and their description.

__init__

Input Parameters

File Name	Containing all the parameters to re-initalize the class.
Parameters	vortexBlobs : {vortexBlobs} class.
	panels : panels class.

Description: Initialize the `vortexMethod` class using **vortexBlob+panelMethod** classes.

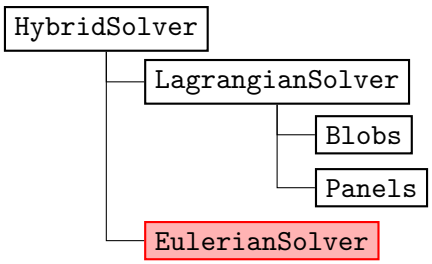
Input parameters:

`Blobs`: vortex particle class

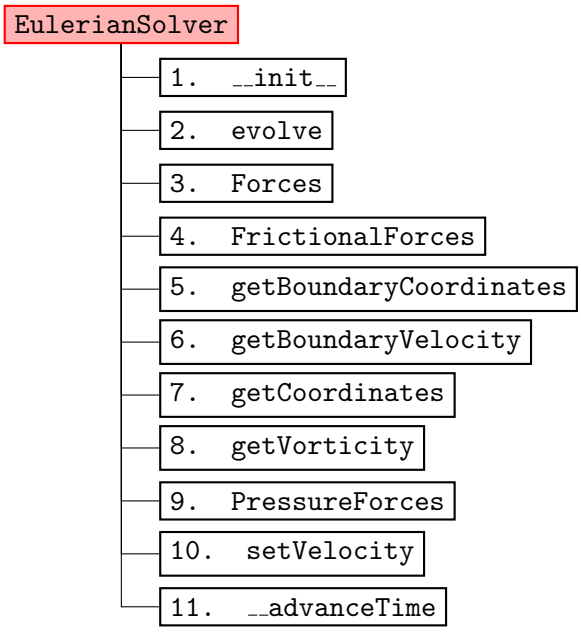
`Panels`: panel method class

EulerianSolver

The main structure for the Navier-stokes class `EulerianSolver`. This class contains all the functions related to computation of the Navier-stokes problem. Below is set of functions that acts as the interface to the class.



Class structure:



Attributes:

Attributes	Description
`deltaT`	The time step size Δt.
`deltaTMax`	The maximum allowable time step size $\max\{\Delta t\}$.
`cfl`	The CourantFriedrichsLewy condition stability number CFL.
`cmGlobal`	The x, y position of the mesh local reference point $(0, 0)$ in the global coordinates.
`hMin`	The minimum mesh cell size.
`nu`	The fluid kinematic viscosity ν.
`probeGridMesh`	The *local* x, y coordinates of the probe grid mesh.
`probeGridParams`	The dictionary containing all the parameters of the probe grid for extracting the vorticity data.

`solverParams`	The dictionary file containing all the solver parameters.
`t`	The current time of the simulation.
`thetaLocal`	The local rotational angle θ of the mesh domain. Therefore, the rotation will be done about local reference point $(0, 0)$, i.e `cmGlobal` in the global coordinate system.
`tStep`	The current step of the simulation.
`uMax`	The maximum fluid velocity $\max\{\mathbf{u}\}$.

Table A.4: Attributes of `EulerianSolver` class and their description.

`__init__`

Description: Initialize the `navierStokes` class either using a `fileName` containing all the necessary parameter for initialization or by explicitly inputing the parameters.

Input Parameters

File Name	Containing all the parameters to re-initalize the class.	
	– or –	
Parameters	Mesh data	: `mesh, boundaryDomains`
	Geometry position	: `cmGlobal, thetaLocal`
	Fluid parameters	: `uMax, nu`
	Solver options	: `cfl`
	Probe grid parameters	: `x0, y0, Lx, Ly, hx, hy`

Description of the parameters:

Mesh data

`mesh`	: the mesh data file.
`boundaryDomains`	: the boundary mesh domain data file.

Geometry position

`cmGlobal`	: the x, y position of the geometry in global coordinates.
`thetaGlobal`	: the rotation angle (in rad) of the geometry in global coordinate system.

Fluid parameters

`uMax`	: the maximum fluid velocity U_{max}. Used to determine the maximum time step size Δt_{max}.
`nu`	: the fluid kinematic viscosity ν, for incompressible navier-stokes problem.

Solver options

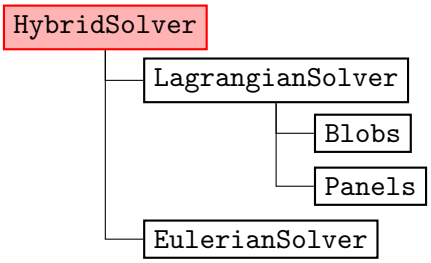
`cfl`	: the CFL stability parameter. If explicit time marching scheme, $CFL < 1$.

Probe grid parameters

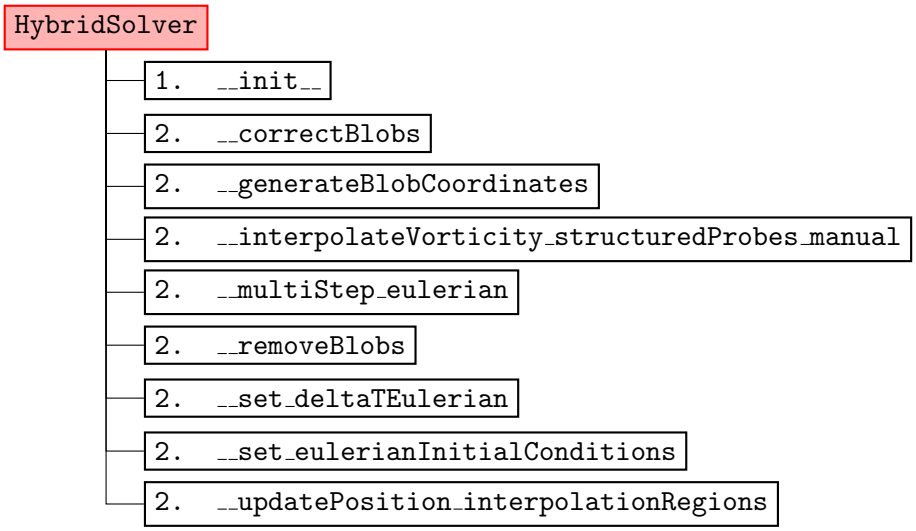
`x0,y0`	: the x, y coordinate of the origin of the probe grid.
`Lx,Ly`	: the x, y size (width and height) of the probing grid.
`hx,hy`	: the x, y spacing of the probe grid cell.

HybridSolver Class

The main structure for the hybrid class `HybridSolver`. This class contains all the functions related to computation of the hybrid problem.



Class structure:



Attributes:

Attributes	Description
`deltaTEulerian`	The time step size of the Eulerian sub-domain Δt_E.
`deltaTLagrangian`	The time step size of the Lagrangian sub-domain Δt_L.
`nu`	The fluid kinematic viscosity ν.
`t`	The current time t of the simulation.
`tStep`	The current step of the simulation.
`vInf`	The x, y component of the free-stream velocity.
`interpolationRegion`	The dictionary containing the `surfacePolygon` and `boundaryPolygon` defining the boundaries of the interpolation region for each Eulerian sub-domains. The geometry is identified by the keys of the Eulerian sub-domain found in `multiEulerian`. The coordinates are defined in local coordinate system of the Eulerian grid and will be transformed (rotated + moved) during the evolution step.
`lagrangian`	The Lagrangian solver class contains all the parameters related to simulation the flow in lagrangian sub-domain.

`multiEulerian`	The `multiEulerian` is solver class containing all the Eulerian sub-domains of the hybrid problem.

Table A.5: Attributes of `HybridSolver` class and their description.

`__init__`

Input Parameters

File Name	Containing all the parameters to re-initalize the class.	
Parameters	`vortexMethod`	: {`vortexMethod`} class.
	`navierStokes`	: `navierStokes` class.
	Interpolation region	: `xPolygon, yPolygon`
	Motion functions	: `T, cmGlobal, thetaGlobal, cmDotGlobal, thetaDotGlobal`

Description: Initialize the `hybrid` class using `LagrangianSolver` + `EulerianSolver` classes.

Input parameters:

`LagrangianSolver`: The vortex method containing `Blobs` and **Panels** classes which can already handle the multi-body problem.

`EulerianSolver`: The Navier-Stokes grid solver class (if multiple: list of `EulerianSolver` classes). The number of navier-stokes class has to be same as the number of vortex panels.

Interpolation Region: the Navier-Stokes class (if multiple: list of `EulerianSolver` classes). Should be equal to number of Navier-Stokes classes. The interpolation region should be defined as list of x, y coordinates of the polygon of the interpolation region.

Motion function: the function describing the motion of all the geometries in the hybrid class.

Interpolation Regions	
`xPolygon,yPolygon`:	the new x, y coordinate of the polygons description the interpolation region. The polygon should have a closed loop (end with starting coordinates) before continuing to the next polygon. In the case of multiple polygons, a list of `xPolygon,yPolygon` should be given and should be as many as the number of navier-stokes domain.

Motion function

`T`	: the current time.
`cmGlobal`	: a list of new positions of the geometries in the hybrid problem.
`thetaGlobal`	: a list of new rotational angle of the geometries in the hybrid problem.
`cmDotGlobal`	: a list of current displacement velocity of the geometries in the hybrid problem.
`thetaDotGlobal`	: a list of current rotational velocity of the geometries in the hybrid problem.

Input parameters : `T`
Assigns : `-`
Returns : `cmGlobal,thetaGlobal,cmDotGlobal,thetaDotGlobal`