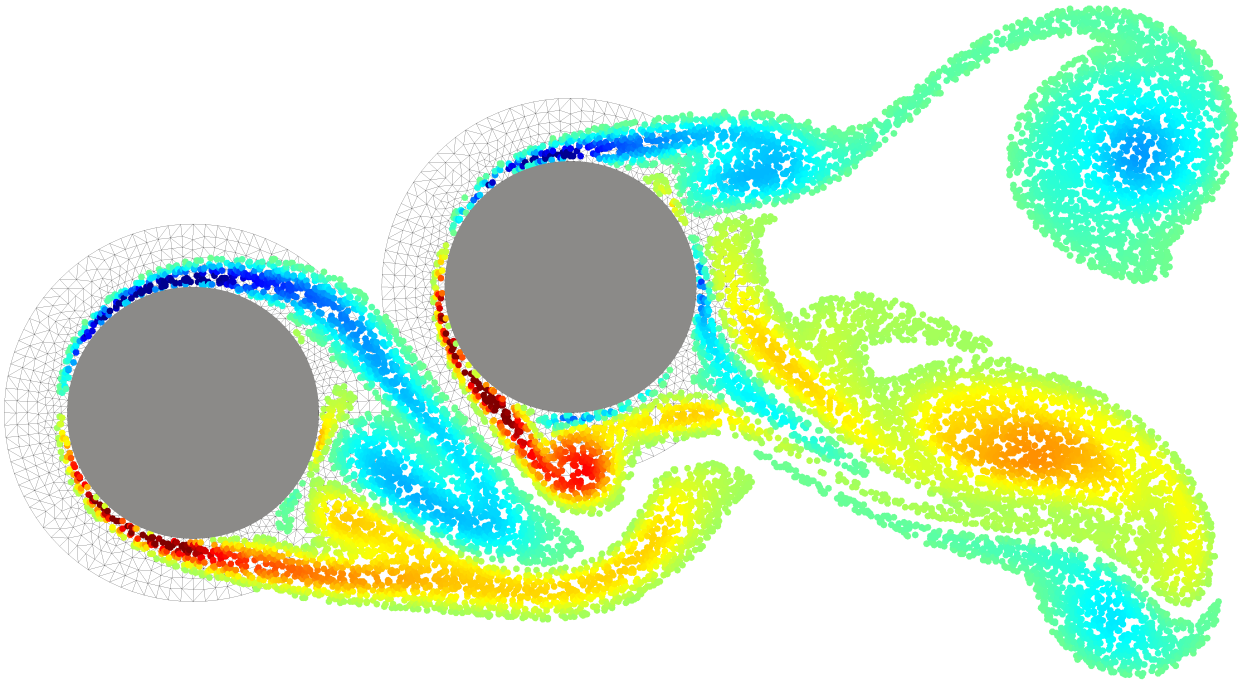
Master of Science Thesis



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

The wake geometry of a Vertical Axis Wind Turbine (VAWT) is unlike the standard Horizontal Axis Wind Turbine (HAWT). The blades of the turbine continuously passes through its own wake, creating complex wake-body interactions such as flow separation and dynamic stall, and convectional grid-based numerical method which is capable of describing such near-body phenomena fails at efficiently resolving the wake geometry. However, as these phenomena have a direct impact on the performance of the VAWT, it is paramount that there exists a numerical method that is not only capable of accurately resolving the small-scale near-body phenomena but also excels at efficiently resolving the unsteady large-scale wake geometry.

This was the goal of the research and the numerical method that satisfied these requirements was the domain decomposition method known as the Hybrid Eulerian-Lagrangian Vortex Particle Method (HELVPM), based on the doctoral thesis of Daeninck [6] and the additional study performed by Stock [17]. In the present study, we coupled an Eulerian Finite Element Method, which only resolves the near-body domain, with a Lagrangian Vortex Particle Method, which resolves the entire wake. The advantage of such fluid domain segregation was that the Eulerian method could focus on accurately describing the near-body features whereas the Lagrangian method could focus on efficiently evolving the wake using simulation acceleration methods such as Fast Multipole Method (FMM) and parallel computation in Graphical Processing Units (GPU).

The present study initially developed, verified and validated the Finite Element method and the Vortex Particle method separately ensuring it performs according to the theory. These methods were then coupled using the algorithm of Daeninck [6] and Lagrangian correction strategy developed by Stock [17]. However, during the study we determined that additional modifications to the coupling strategy is required to ensure conservation of circulation. Furthermore, it was determined that the spatial resolution of numerical method at the overlap region, where coupling takes place, plays a crucial role in the accuracy of the coupling.

Even though the hybrid method of the present study sacrificed some efficiency to ensure an accurately coupled scheme, we must not that it is still at its infancy. With the help of

advanced techniques such as varying particle core size, higher order time marching scheme in the Eulerian method, and boundary element method acceleration techniques such as FMM and/or GPU calculation, the hybrid method has the potential to substantially outperform standard grid based methods. In conclusion, the hybrid method that has been developed here has the potential to accurately describing the near-wake phenomena and efficiently evolving the wake of a VAWT.

Acknowledgements

The work I present here would not have been possible without the support of my supervisors dr.ir. C. (Carlos) J. Simao Ferreira and dr.ir. A. (Artur) Palha da Silva Clerigo. I want to thank Carlos for the constant encouragement and ensuring that I didn't loose myself in the detail and to have a global picture. Thank you for also giving me fresh perspective when it seems impossible and making me to think outside the box. I want to thank Artur for his daily supervision and guidance to teach me all the intricate details for developing this numerical method. I also want to thank him for putting aside time to work hand-in-hand to find solutions to the problems I faced and supporting me throughout the thesis.

I want to thank my friends: Chid, Thor, Oliver, Mark, Rob, Alberto, Hector and Dieter for spending time together at the university, tackling the challenge of finishing the master thesis together. You guys gave me joy during the work and it was fun working beside you guys. I want to thank my other friends you made by my last few years in Netherlands an enjoyable experience.

Finally, I want to thank my family: Daddy, Mummy, Chechi, and Denis Chettan. Thank you for your patience and your unending love throughout my life.

Delft, The Netherlands
Date TBD

L. Manickathan B.Sc.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

AD	Actuator Disk
BEM	Blade Element Momentum
CFD	Computional Fluid Dynamics
CG	Continuous Galerkin
CSVP	Constant-Strength Vortex Panel
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
PC	Population Control
PIV	Particle Image Velocimetry

PSE	Particle Strength Exchange
RWM	Random Walk Method
VAWT	Vertical-Axis Wind Turbine
VPM	Vortex Particle Method
WRS	Wee Remeshing Scheme

Chapter 1

Coupling Eulerian and Lagrangian Method

Chapter **??** provided a detailed summary on Hybrid Eulerian-Lagrangian Vortex Particle Method, where we introduced the concept of coupling the Lagrangian and the Eulerian method. The chapter investigated the coupling algorithm without Schwartz iterative method used by Daeninck [6], and the Lagrangian correction algorithm used by Stock [17]. However, during the investigation and implementation of the algorithms, we observed some issues that had to be dealt with.

1.1 Modifications to the Lagrangian Correction Strategy

The Lagrangian correction step by Stock [17], based on the coupling strategy of Daeninck [6] had to be modified in the present work. During the investigation of the algorithm, it became apparent that without additional steps, the total circulation in the Lagrangian method will be not be conserved.

The two issues that causes an error in the total circulation (and subsequently introduces additional error in the coupling) are the following:

- **Vortex particle re-initialization**: In section **??**, we observed that initializing the particles using the local particle volume and local vorticity causes a diffusive effect on the vorticity distribution due to the *smoothing error* of Gaussian kernels. Section 1.1.1 elaborates the cause and correction required for this problem.
- **Circulation of Vortex sheet**: In section **??**, we saw that as the vortex panel problem is singular, we require an additional constraint on total circulation of the vortex sheet. It is an unknown and has to be determined from the solution of the Eulerian method. Section 1.1.2 elaborates the methodology for determining the strength.

- **Conservation of total circulation**: The Lagrangian correction step, consisting of re-initializing the vortex blobs inside the interpolation domain Ω_I (Figure **??**), and the transfer of circulation to the vortex panels must ensure that the total circulation in the Lagrangian method is conserved. Section 1.1.3 elaborates the methodology used to explicitly ensure the conservation of circulation in the Lagrangian domain during the correction step.

1.1.1 Vortex Particle Re-initialization

The Lagrangian correction step requires the correction (or adjustment/modification) of the strength of the vortex particles inside the interpolation domain Ω_I, shown in Figure **??**. However, when investigating this approach, it becomes apparent that the standard initializing approach of local particle volume and local vorticity is erroneous.

Concern with re-initialization method

The Lagrangian method discretizes the exact vorticity field ω by linear combination of N Gaussian basis functions, i.e vortex blobs. Let $\mathcal{P} = \{1, ..., N\} \subset \mathbb{N}$ be the set of particle indices of N particles p with position $\mathbf{x}_p$. The discrete continuous vorticity field $\hat{\omega}$ is given as,

$$\omega \approx \hat{\omega}(\mathbf{x}) = \sum_{p \in \mathcal{P}} \alpha_p \zeta_\sigma(\mathbf{x} - \mathbf{x}_p). \tag{1.1}$$

where α_p is the strength of particle $p \in \mathcal{P}$. In vortex method, it is typically a standard approach to initialize the particle strengths α_p using the local particle area h^2 and the local vorticity ω,

$$\alpha_p = \omega_p \cdot h^2. \tag{1.2}$$

where ω_p is the vorticity at particle position $\mathbf{x}_p$. This approach was also employed by Stock [17], to modify the strengths of the vortex blobs inside the correction region Ω_I. However, in section **??**, we observed that this type of initialization introduces a *smoothing error*. Barba and Rossi [**?**] noticed that the standard initialization corresponds to a Gaussian blurring of the original vorticity field and is equivalent to the "diffusion" of the vorticity.

The accurate re-initialization of the Lagrangian vorticity field $\hat{\omega}^L$ in Ω_I using the Eulerian vorticity field ω^E must satisfy the following equality:

$$\omega^E = \hat{\omega}^L \qquad \text{in } \Omega_I. \tag{1.3}$$

Satisfying equation 1.3 ensures that the vorticity solution of the Eulerian method ω^E matches the blurred (mollified) Lagrangian vorticity field $\hat{\omega}^L$. Substituting equation 1.1 into Equation 1.3 gives:

$$\omega^E(\mathbf{x}_j) = \sum_{i=p \in \mathcal{P}} \alpha_i \zeta_\sigma(\mathbf{x}_j - \mathbf{x}_i) \qquad \text{in } \Omega_I, \tag{1.4}$$

simplifying to a system of linear equations,

$$\omega^E = \mathbf{A}_{ij}\alpha_i \qquad \text{in } \Omega_I \tag{1.5}$$

where $\mathbf{A}_{ij} = \zeta_\sigma(\mathbf{x}_j - \mathbf{x}_i)$ is a coefficient matrix. Therefore the strengths of the particles α_p must be obtained from equation 1.5 and the initialization of the particles using equation 1.2 is mathematically incorrect.

Equation 1.5 is a linear system of equations equating the strengths of each particle to the desired Eulerian vorticity distribution. However, inverting the matrix $\mathbf{A}$ is still an open equation in vortex method, as stated by Koumoutsakos and Cottet [**?**], and was investigated by Barba and Rossi [**?**]. 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 Beale's iterative method which uses a successive over-relaxation (SOR) for solving the equation 1.5. This method relies on iterative correction of all the particles in the full Lagrangian domain. However, in our case for initializing the strengths of the particles only in the sub-domain Ω_I of the Lagrangian domain Ω_L, it would require us to modify the strength of only those particles. In such case, Beale's iterative method is not valid and cannot be used. Therefore, Beale's method cannot be used to solve the problem of smoothing error.

In future, the key to solving this smoothing error might be in the research works of Barba and Rossi [**?**], 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 solution is to minimize the error in equation 1.2.

Modification

The error ϵ due to the mismatch in the Eulerian vorticity field ω^E and the Lagrangian vorticity $\hat{\omega}^L$ in the interpolation domain Ω_I is defined as,

$$\epsilon = |\omega^E - \hat{\omega}^L| \qquad \text{in } \Omega_I \tag{1.6}$$

where the error is divided into,

$$\epsilon = \epsilon_\sigma + \epsilon_h, \tag{1.7}$$

the sum of the smoothing error ϵ_σ, and the discretization error ϵ_h. In section **??**, we investigated the minimization of this error and observed that the error ϵ scales with particle resolution. With an overlap ratio of $\lambda = 1$ and a small particle core spreading σ, we can ensure that the initialization error inside the interpolation domain Ω_I is minimal. To determined an appropriate core spreading σ, we should target an initialization error of $\epsilon \leqslant 5\%$ ensuring a small error in coupling.

1.1.2 Circulation of Vortex sheet

The second concern with the implementation of Daeninck's coupling strategy and Stock's Lagrangian correction step is the uncertainty of the vortex sheet strengths. In section **??**, we described that to time march the Lagrangian solution from t_n to $t_n + \Delta t_L$, we have to enforce a *no-slip* boundary condition at the wall by computing the satisfactory vortex sheet distribution γ.

To solve for the vortex sheet distribution γ that satisfy the no-slip boundary conditions, we discretized the boundary integral equation using vortex panels, as described in section **??**. Koumoutsakos [**?**], related the vortex sheet strengths to the no-slip boundary

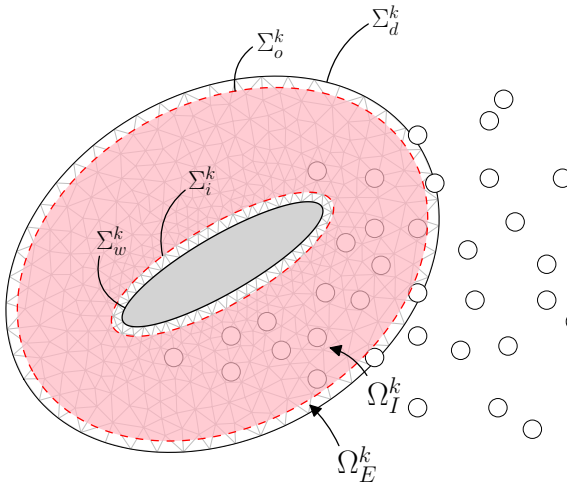


Figure 1.1: A domain decomposition with $k = 1, ..., N_E$ Eulerian subdomains. The figure shows the definition of the k^{th} interpolation domain Ω_I^k belonging to the k^{th} Eulerian domain Ω_E^k.

conditions with Fredholm integral equation of the second kind, equation **??**. However, as this equation is singular, we require an additional constraint to obtain a unique solution. Kelvin's circulation theorem imposes a direct constraint on the integral strengths of the vortex sheet, shown in equation **??**, and can be used to find the unique solution.

Let us investigate a hybrid problem with N_E number of Eulerian subdomain and the k^{th} Eulerian domain $\Omega_E^k \subset \Omega_L$ is defined as shown in Figure 1.1. Stock [17] said that the Eulerian solution is assumed to be correct from the body surface Σ_w^k to only somewhat inside of the outer Eulerian domain Σ_d^k. More precisely, all the Eulerian solution within the domain Ω_{in}^k with $\partial\Omega_{in}^k = \Sigma_o$ as shown in Figure 1.2. During the Lagrangian correction step, Stock corrected the particles p with $\mathbf{x}_p \in \Omega_I^k$ using the more accurate Eulerian solution. So, any remaining Eulerian solution within Σ_o^k that was not transfered during the Lagrangian correction step should be resolved by the vortex sheet to ensure conservation of circulation.

In terms of circulation, we require that the Eulerian circulation $\Gamma_{in}^{E,k}$ and the corrected Lagrangian circulation $\hat{\Gamma}_{in}^{L,k}$ of the k^{th} correction region Ω_{in}^k must match to ensure conservation of circulation during the correction step, given by,

$$\Gamma_{in}^{E,k} = \hat{\Gamma}_{in}^{L,k} \qquad \text{in } \Omega_{in}^k, \tag{1.8}$$

The Lagrangian circulation $\hat{\Gamma}_{in}^{L,k}$ is decomposed as follows,

$$\hat{\Gamma}_{in}^{L,k} = \Gamma_{\gamma}^{L,k} + \hat{\Gamma}_{I}^{L,k}, \tag{1.9}$$

where $\Gamma_{\gamma}^{L,k}$ is the circulation of the vortex sheet and $\hat{\Gamma}_{I}^{L,k}$ is the circulation of the corrected vortex blobs in the interpolation domain Ω_I^k. The circulation of the particles $\hat{\Gamma}_{I}^{L,k}$ is determined directly from the sum of particles particle strengths $\hat{\alpha}_p^k$,

$$\hat{\Gamma}_{I}^{L,k} = \sum_p \hat{\alpha}_p^k \qquad \text{for } \mathbf{x}_p \in \Omega_I^k, \tag{1.10}$$

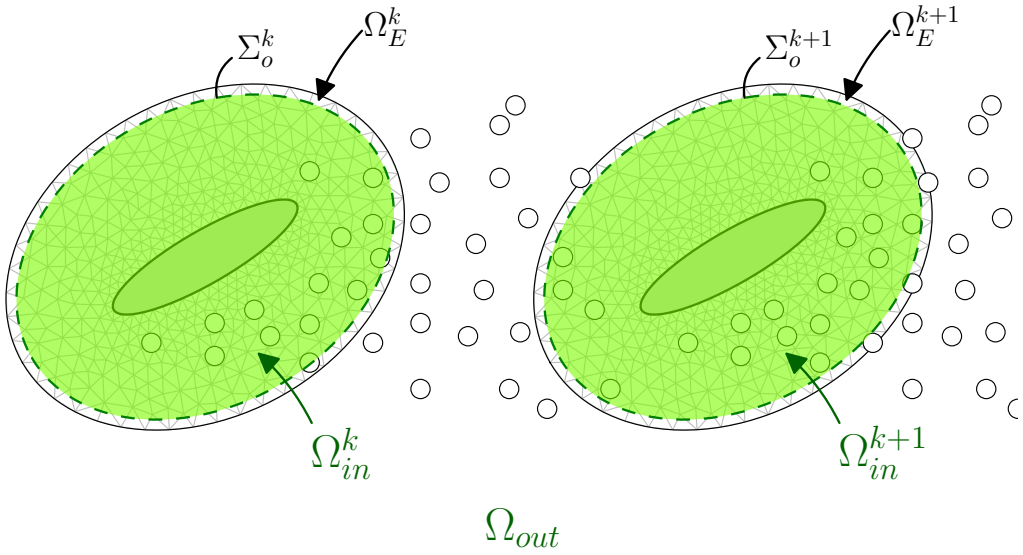


Figure 1.2: The segregation of the Lagrangian domain $\Omega_L = \Omega_{in} \cup \Omega_{out}$ into the region which is uncorrected Ω_{out} and the region which is corrected $\Omega_{in} = \bigcup_{k=1}^{N_E} \Omega_{in}^k$.

where $\hat{\alpha}_p$ is the strength of the corrected particles determined using equation 1.2. Substituting equation 1.9 into equation 1.8 helps us determine the circulation of the vortex sheet,

$$\Gamma_\gamma^{L,k} = \Gamma_{in}^{E,k} - \hat{\Gamma}_I^{L,k}. \tag{1.11}$$

Using this additional constraint on the circulation of the vortex sheet, we can solve the distribution of the vortex sheet that satisfies the no-slip boundary condition.

1.1.3 Conservation of Total Circulation

In section 1.1.1 we observed that the method used to determine the strength of particles inside the interpolation domain Ω_I (Figure 1.2) is not accurate for transferring the Eulerian solution to the Lagrangian domain. This approach is erroneous and has a detrimental effect on the total circulation of the Lagrangian domain Ω_L. To ensure that circulation is conserved during the correction step, we to perform an additional correction to the transfer.

The Lagrangian domain Ω_L can be divided into two section (as shown in Figure 1.2):

- Correction region Ω_{in}: The Lagrangian region where the correction takes place: $\Omega_{in} = \bigcup_{k=1}^{N_E} \Omega_{in}^k$, where the k^{th} correction region is defined by $\partial\Omega_{in}^k = \Sigma_o^k$, the outer boundary of the interpolation region Ω_I^k.
- Unmodified region Ω_{out}: The Lagrangian region that is outside the correction region and is therefore unmodified during the correction step: $\Omega_{out} = \Omega_L \backslash \Omega_{in}$.

The total circulation in the Lagrangian domain before the correction is given as,

$$\Gamma^L = \Gamma_{in}^L + \Gamma_{out}^L, \tag{1.12}$$

where $\Gamma_{in}^{L} = \sum_k \Gamma_{in}^{L,k}$ is circulation inside Ω_{in} before they are corrected, and Γ_{out}^{L} is the circulation in Ω_{out}. During the Lagrangian correction step, the circulation Γ_{in}^{L} is corrected and replaced with the corrected circulation $\tilde{\Gamma}_{in}^{L}$ determine from the Eulerian method, equation 1.8, resulting in a new total Lagrangian circulation $\tilde{\Gamma}^{L}$ given by,

$$\tilde{\Gamma}^{L} = \tilde{\Gamma}_{in}^{L} + \Gamma_{out}^{L}, \tag{1.13}$$

To ensure that the circulation is conserved during the correction step, we required $\Delta\Gamma = 0$. This gives us the requirement that,

$$\Gamma^{L} = \tilde{\Gamma}^{L}. \tag{1.14}$$

or, substituting equation 1.12 and equation 1.13 into equation 1.14 gives us the requirement that,

$$\Gamma_{in}^{L} = \tilde{\Gamma}_{in}^{L}, \tag{1.15}$$

and equivalently,

$$\Gamma_{in}^{L,k} = \tilde{\Gamma}_{in}^{L,k}. \tag{1.16}$$

This means that the circulation inside the correction region Ω_{in}^{k} before the correction should match the circulation after the correction. However, as there exists a slight error in the particle initialization, we introduce a small error in circulation ϵ^{k} during the transfer, resulting in,

$$\hat{\Gamma}_{in}^{L,k} = \tilde{\Gamma}_{in}^{L,k} + \epsilon_{in}^{L,k}, \tag{1.17}$$

where $\hat{\Gamma}_{in}^{L,k}$ is the erroneous circulation that was actually transfered and $\tilde{\Gamma}_{in}^{L,k}$ is the correct circulation that was supposed to be transferred. Substituting equation 1.17 into equation 1.15, we see that the error $\epsilon_{in}^{L,k}$ is written as,

$$\epsilon_{in}^{L,k} = \hat{\Gamma}_{in}^{L,k} - \Gamma_{in}^{L,k}. \tag{1.18}$$

and is simply the mismatch in the circulation during the correction step. This error can by modifying the new set of initialized particles in the interpolation domain Ω_I. Using equation 1.10, the final strengths of the particles $\tilde{\alpha}_p^k$ for particles $\mathbf{x}_p^k \in \Omega_{in}^{k}$ is given as,

$$\tilde{\alpha}_p^k = \alpha_p^k - \frac{\epsilon_{in}^{L,k}}{N}, \tag{1.19}$$

where α_p^k is the uncorrected strengths determined using the local particle area and local vorticity, equation 1.2, and N is number of particles inside the correction region Ω_{in}^{k}. Following this procedure in addition to Stocks Lagrangian correction strategy described in section **??**, we will ensure that our hybrid scheme conserves circulation.

1.2 Modified Lagrangian Correction Algorithm

In this section we will investigate the algorithm used for the *Correct Lagrangian* step of our hybrid evolution, Figure 1.3. We first summarized the algorithm used by Stock [17] in section **??**. However in section 1.1, we determine that the algorithm required some

modification to ensure minimum interpolation error and conservation of total circulation. Therefore, we will also describe the algorithm required to implement these modifications.

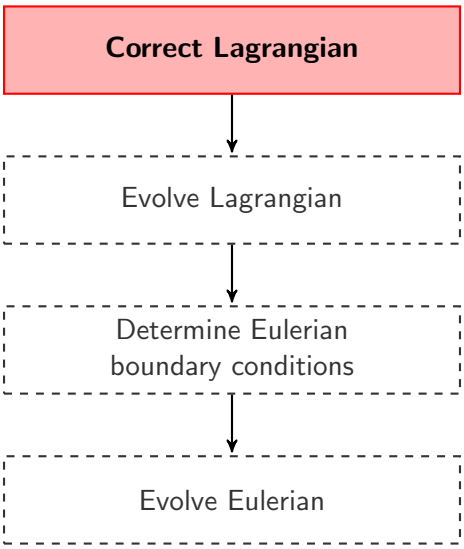


Figure 1.3: Flowchart of the hybrid evolution, focusing on the 1st step: Correct the Lagrangian solution.

The general approach we used to perform transfer of the solution from the Eulerian to Lagrangian domain is by substituting particles inside the zone of correction (i.e the correction region Ω_{in} as shown in Figure 1.1) with a correct set of particles that represent the more accurate Eulerian solution. We need to determine the vorticity at their positions so that we can use equation 1.2 to initialize the strengths of these new particles. The vorticity at these positions can be determined by interpolating the Eulerian solution onto their positions.

However, determining the location of the particle w.r.t the unstructured grid of the Eulerian method is computationally expensive. Therefore, we will first interpolate the Eulerian solution onto a regular grid. Performing a search algorithm on such grid is computationally less expensive. Furthermore, to ensure that the interpolation of vorticity from unstructured grid onto a regular structured grid is also efficient, the regular structured grid will be bounded to the Eulerian domain. In such case, the interpolation problem only needs to be setup once as the transfer weights remains unchanged throughout the simulation.

The final stages of the Lagrangian correction step is to ensure that the total circulation is conserved. We will therefore determine the strength of the vortex sheet and the total strength of the newly generated vortex blobs such that circulation is conserved.

The algorithm describing these procedure is organized and summarized as follows:

1. **Interpolate vorticity**: Interpolate the vorticity from the unstructured Eulerian mesh onto a structured grid that is bounded to the Eulerian domain Ω_E.

2. **Remove particles**: Remove particles that are inside the interpolation domain Ω_I.

3. **Generate particles**: Generate zero-strength particle inside the interpolation domain Ω_I.

4. **Assign strengths to particles**: Interpolate the vorticity from the structured grid onto the position of the newly generated particles. Using the standard particle initialization approach described in section 1.1.1, assign the strengths to the newly generated particles.

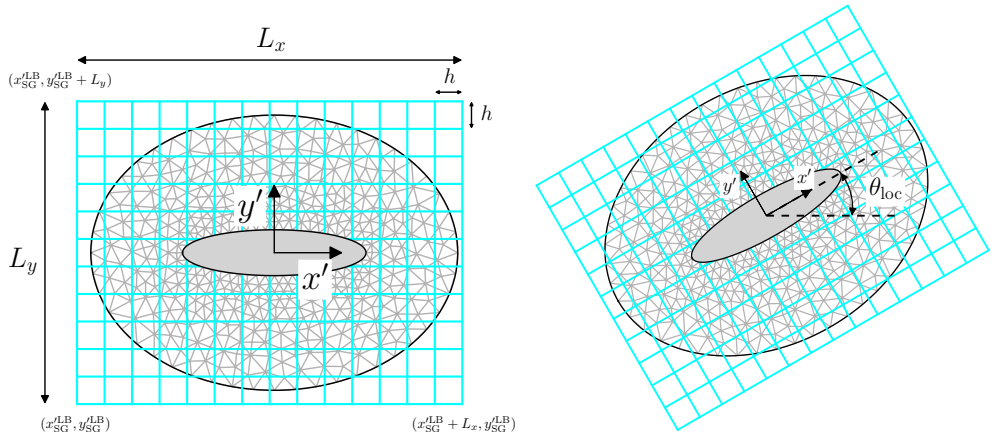
5. **Correct total circulation**: Using the approach described in section 1.1.2 and section 1.1.3, determine the total strength of the vortex sheet and total strength of the newly generated vortex blobs such that the total circulation is conserved.

The steps 1 to 4 will be described for the k^{th} Eulerian domain Ω_E^k. These steps simply needed to be iterated for all N_E number of Eulerian domains (i.e bodies) in the case of a multi-body problem. The last step of correcting the total circulation is performed for all the Eulerian domains simultaneously as we require the knowledge of total circulation. A detailed description of the each steps of the Lagrangian correction algorithm is given below.

1.2.1 Interpolate Vorticity

In this step, we will interpolate the vorticity from the unstructured grid of the Finite Element method onto a regular structured grid, that is bounded to the Eulerian domain Ω_E^k. The algorithm for interpolating the vorticity from the unstructured grid onto a structured grid is as follows:

1.a) **Setup a structured grid** (*only once*): A structured grid (SG) is created in the local coordinate system of the k^{th} Eulerian domain Ω_E^k. The grid is constructed such that it covers the Eulerian grid (SG) of the Eulerian method, as shown in Figure 1.4a.



(a) Local coordinate system **(b)** Global coordinate system

Figure 1.4: Structured grid (cyan) covering the entire Eulerian grid (**black**).

The parameters that define the structured grid in the local coordinate system are the starting point of the grid (the left-bottom corner) $(x'^{\mathrm{LB}}_{\mathrm{SG}}, y'^{\mathrm{LB}}_{\mathrm{SG}})$, the horizontal length of the grid L_x, the vertical length of the grid L_y, and the grid cell spacing h (which is equal to the nominal blob spacing). The lengths L_x and L_y are defined such that it is a multiple of the grid spacing h,

$$L_x = n_x \cdot h \tag{1.20a}$$

$$L_y = n_y \cdot h, \tag{1.20b}$$

where n_x and n_y are the number of cells in the horizontal and the vertical direction, respectively. The transformation of the structured grid follows that of the body and therefore the structure grid does not move w.r.t to the Eulerian grid. For more details on the transformation from the local coordinate system to the global, see appendix A. Figure 1.4b shows the structured grid bounded to the body in the global coordinate system.

1.b) **Determine the weights** (*only once*): The interpolation of the Eulerian vorticity ω from the Eulerian grid onto the structured grid is represented as,

$$\hat{\omega}_j = \sum_i \omega_i W_{ij}, \tag{1.21}$$

where $\hat{\omega}_j$ is the interpolated vorticity on the structured grid and W is the interpolation weights. As the structured grid is bounded to the Eulerian grid, the weights of the interpolation does not change throughout the simulation and only needs to be calculated once. This makes the interpolation problem computationally very efficient.

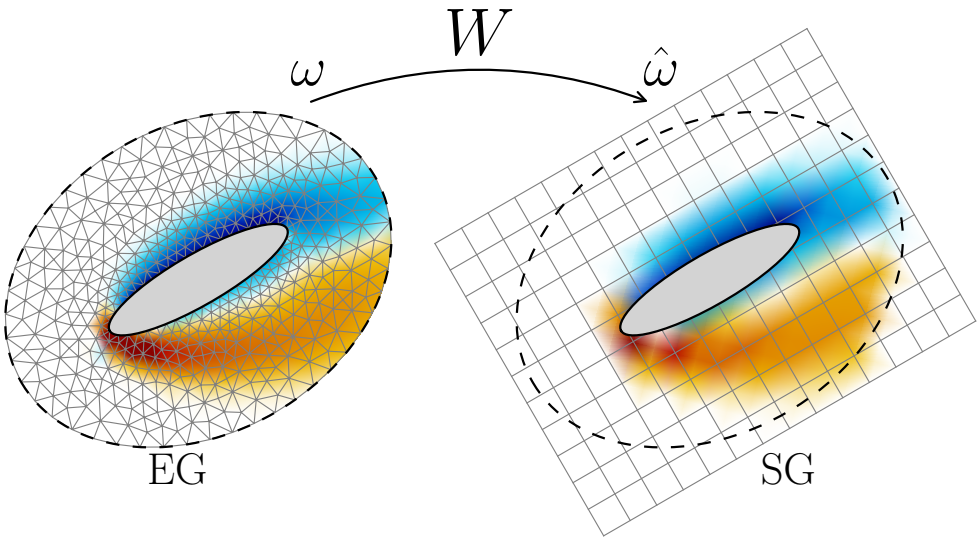


Figure 1.5: Interpolation of the vorticity ω from the Eulerian grid onto the structured grid using equation 1.21.

1.c) **Interpolate the vorticity**: This step needs to be perform at every Lagrangian correction step. The vorticity $\hat{\omega}$ on the structured grid is calculated by simply solving the pre-assembled problem, defined by equation 1.21.

To construct the interpolation problem, we used a tree search algorithm from the CGAL library [**?**], included in FEniCS and adapted for fast repeated evaluation by Mortenson (Fenicstools [**?**]). The algorithm probes the vorticity function ω of the unstructured Eulerian grid at the nodes of the structured grid $(x_{\text{SG}}, y_{\text{SG}})$. Figure 1.5 shows a depicts the result of the transferring the vorticity from the unstructured grid to the structured grid. The vorticity on the structured grid can then be efficiently interpolated onto the positions of the particles.

1.2.2 Remove Particles

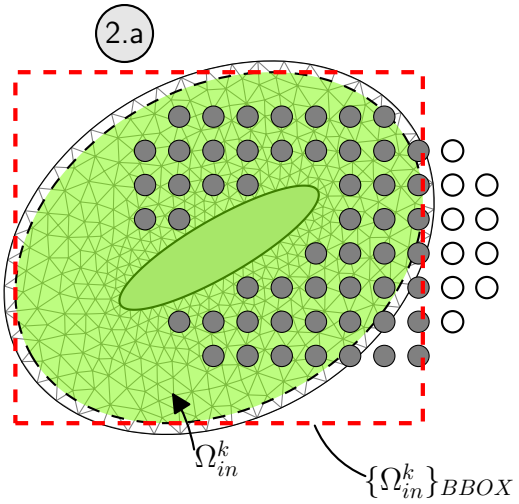
In this step, the uncorrected vortex particles inside the correction region Ω_{in} (as shown in Figure 1.2) will be removed. Let $\mathcal{P}$ be the set of particle indices with position $\mathbf{x}_p \in \Omega_L$. The algorithm for removing the particles p is as follows:

2.a) **Find the particles inside the BBOX of Ω^k_{in}**: Find the particles p that lie inside the minimum bounding box of the correction domain Ω^k_{in}. Let $\mathcal{P}^k_{\text{BBOX}}$ be the set of particles indices that are within the bounding box of the correction domain Ω^k_{in}, the gray particles shown in Figure 1.6a. The minimum bounding box (shown in red) is constructed using the outer boundary polygon of the correction region Σ_o. Performing this search algorithm is computationally efficient and reduces the number of particles of interest for the more expensive *point-in-polygon* search algorithm of the next step.

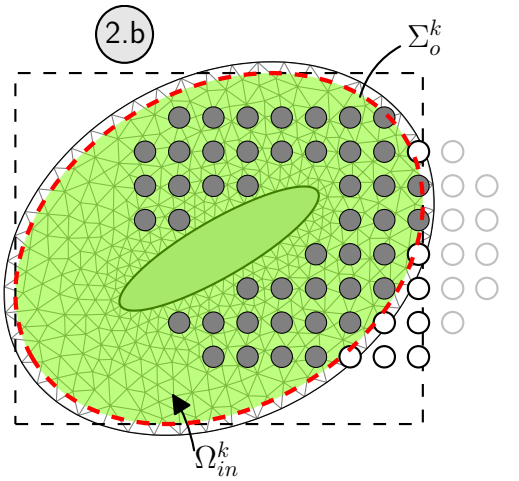
2.b) **Find the particles inside the correction region Ω^k_{in}**: Determine which of the particles in the set $\mathcal{P}^k_{\text{BBOX}}$ also lie within the correction domain Ω^k_{in}. Let $\mathcal{P}^k_{in} \subset \mathcal{P}^k_{\text{BBOX}}$ be the set of particles that also lie inside the correct region Ω^k_{in}, the gray particles in Figure 1.6b. The search algorithm is performed using a *point-in-polygon* test with the outer boundary Σ^k_o (shown in red).

2.c) **Remove particles inside correction region**: Remove the set of particles inside the correction region $\mathcal{P}^k_{in}$. These particles have a total circulation of $\Gamma^{L,k}_{in}$ (as used in Equation 1.12) which needs to be compensated later to ensure conservation of total circulation. Figure 1.6c shows the final set of particles after *remove particle* step.

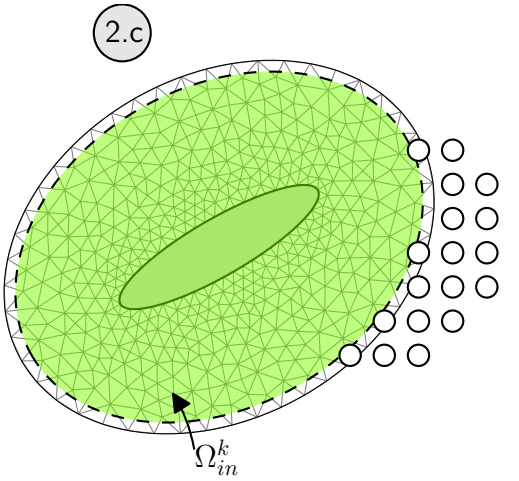
To perform the point-in-polygon test, we used the `pnpoly` function of `matplotlib`, the python 2D plotting library created by Hunter [**?**]. The function implemented the *point inclusion in polygon* test algorithm developed by Franklin [**?**]. The algorithm is based on the crossings test, which determines 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.



(a) Step 2.a: Find the particles inside the BBOX of Ω^k_{in}

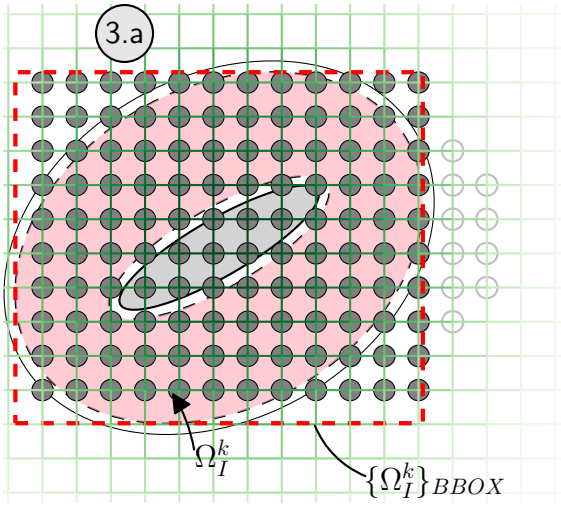


(b) Step 2.b: Among them, find the particles that are also inside the correction region Ω^k_{in}

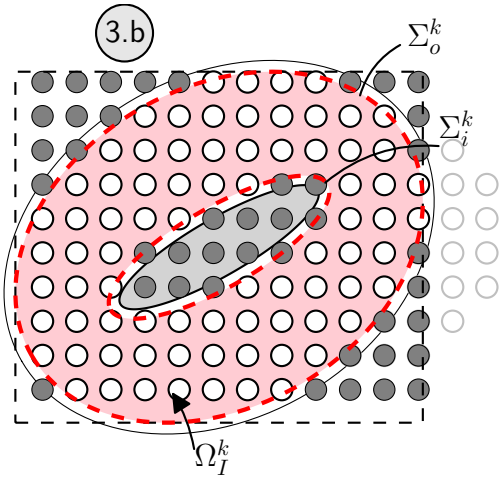


(c) Step 2.c: Remove the particles that are inside the correction region Ω^k_{in}

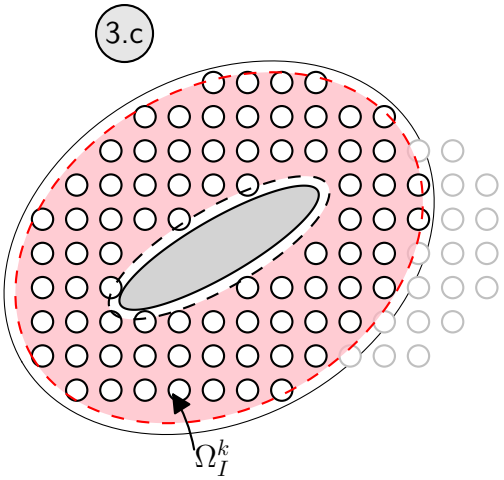
Figure 1.6: Figures depicts the computationally optimized algorithm for finding and removing the vortex particles inside the correction region Ω^k_{in}.



(a) Step 3.a: Generate particles inside the BBOX of the interpolation domain Ω_I^k matching the Lagrangian mesh



(b) Step 3.b: Among them, find the particles that are outside the interpolation domain Ω_I^k



(c) Step 3.c: Remove the newly generated particles that are outside interpolation domain Ω_I^k

Figure 1.7: Figures depicts the algorithm for generating vortex particles inside the interpolation domain Ω_I^k matching the Lagrangian mesh.

1.2.3 Generate Particles

In order to interpolate the solution from the Eulerian domain onto Lagrangian domain, we require the knowledge of the particle positions. In this step, we will generate zero-strengths particles inside the interpolation region Ω_I^k (Figure 1.1).

The algorithm for generating zero-strength particles inside the k^{th} interpolation domain Ω_I^k is as follows:

3.a) **Generate particles inside the BBOX of interpolation domain** Ω_I^k: Generate zero-strength particles inside the bounding box of the interpolation region Ω_I^k, the gray particles shown in Figure 1.7a. The particles are positioned at the nodes of the global Lagrangian remeshing grid (introduced in section **??**) such that particles are equally distributed as satisfies the overlap ratio criteria. Let $\hat{\mathcal{P}}^k_{\text{BBOX}}$ be the set all newly generated particles (gray) inside the bounding box at the nodes of the Lagrangian method (green).

3.b) **Find newly generated particles inside interpolation region** Ω_I^k: We perform a point-in-polygon test for the newly generated particles $\hat{\mathcal{P}}^k_{\text{BBOX}}$, so that we can neglect the particles that are outside the interpolation region Ω_I^k. Let $\hat{\mathcal{P}}^k_{in} \subset \hat{\mathcal{P}}^k_{\text{BBOX}}$ be the set of particles (white) that are within the interpolation region Ω_I^k. Figure 1.7b shows the set of particles that are not inside the interpolation region $p_i \notin \hat{\mathcal{P}}_{in}$ (gray).

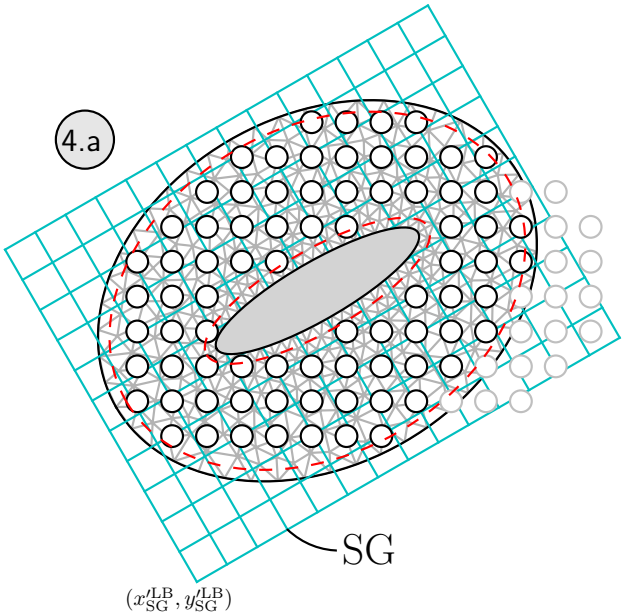
3.c) **Remove all the newly generated particles outside the interpolation region** Ω_I^k: Remove all the newly generated particles that outside the interpolation region $p_i \notin \hat{\mathcal{P}}_{in}$. Figure 1.7 shows the final distribution of the particles, with a uniformly distributed particles inside the interpolation region Ω_I^k without any gaps.

In combination with the *remove particle* step and the *generate particle* step, we can ensure that vortex blobs are uniformly distributed inside the interpolation region Ω_I^k without any gaps. If the initial distribution of the particles (as shown in Figure 1.6a) was not modified, we would see that due to the gaps in particle distribution not all vorticity could be interpolated to the Lagrangian method. However, now with the uniform distribution (as shown in Figure 1.7c), this is no longer the case.

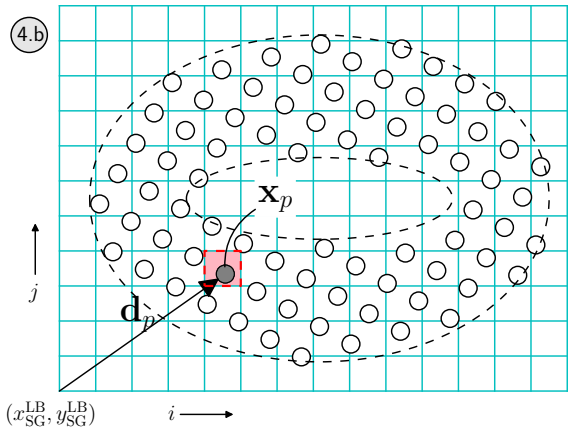
1.2.4 Assign Strengths of Particles

In this step, we will describe the algorithm for assigning strengths to the newly generated particles based on theory summarized in section 1.1.1. The strengths of the particle are determined by first interpolating the vorticity from the structured grid onto the position of the newly generated zero-strength vortex blobs, then using equation 1.2 to assign the strengths.

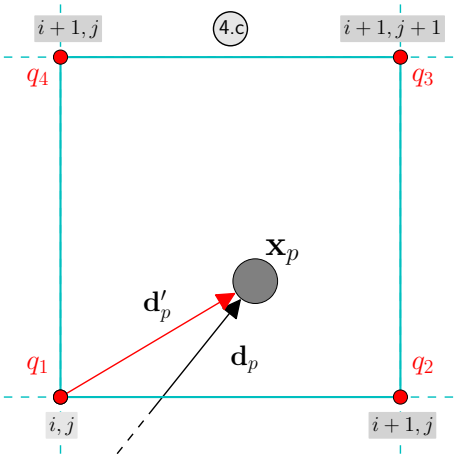
The algorithm for assigning the strengths of the newly generated particles inside the interpolation domain Ω_I^k is as follows:



(a) Step 4.a: Determine the orientation of the SG w.r.t to the newly generated particles



(b) Step 4.b: Project vortex blobs onto the coordinate system of the SG



(c) Step 4.c: Determine the weights of the bilinear interpolation

Figure 1.8: Figures depicts the algorithm for assigning the strengths of the vortex particles inside the interpolation domain Ω_I^k.

4.a) **Determine the orientation of the SG w.r.t to the newly generated particles**: The interpolation algorithm starts by first determining the position of the structure grid SG w.r.t to the vortex blobs. The key parameters that define the position and the orientation of the structured grid in the global coordinate system is the position of the left bottom corner $(x_{\text{SG}}^{\text{LB}}, y_{\text{SG}}^{\text{LB}})$, and the rotational angle θ_{loc}, as described in section 1.2.1. Figure 1.8a, shows the global position of the structured grid w.r.t to the newly generated vortex blobs.

4.b) **Project the blobs into the coordinate system of the SG**: Once we know the position of the SG, we need to determine in which cells of the SG, the newly generated set of particles $\hat{\mathcal{P}}_{in}^{k}$ are located. To determine this, we need to project the vortex blobs onto the coordinate system of the structured grid. Figure 1.8b shows the projected orientation of the vortex blobs in the local coordinate system of the structured grid. Let $\mathbf{d}_p = |\mathbf{x}_p - \mathbf{x}_{SG}|$ be the offset of the particle p w.r.t to the left bottom corner of the structured grid. In the local coordinate system, the offset $\mathbf{d}_p$ is given by,

$$\mathbf{d}_p = \begin{bmatrix} d_{x,p} \\ d_{y,p} \end{bmatrix} = \begin{bmatrix} \cos\theta_{loc} & \sin\theta_{loc} \\ -\sin\theta_{loc} & \cos\theta_{loc} \end{bmatrix} \cdot \begin{bmatrix} x_p - x_{\text{SG}}^{\text{LB}} \\ y_p - y_{\text{SG}}^{\text{LB}} \end{bmatrix} \tag{1.22}$$

where $d_{x,p}$ and $d_{y,p}$ are the horizontal and the vertical distance from the left bottom corner, as shown in the Figure 1.8b.

In the local coordinate, the nodes of the structured grid are addressed by index i and j for x and y direction respectively. Let q_1, q_2, q_3, q_4 refer to the four nodes of the cell in which the particle p is located, as shown in Figure 1.8b). In this cases, the coordinates of these nodes in the local coordinate system are given as,

$$\begin{aligned} \mathbf{x}'_1 &= (i_p \cdot h, j_p \cdot h), \\ \mathbf{x}'_2 &= ([i_p + 1] \cdot h, j_p \cdot h), \\ \mathbf{x}'_3 &= ([i_p + 1] \cdot h, [j_p + 1] \cdot h), \\ \mathbf{x}'_4 &= (i_p \cdot h, [j_p + 1] \cdot h), \end{aligned} \tag{1.23}$$

where i_p and j_p are determined by a floor function,

$$i_p = \left\lfloor \frac{d_{x,p}}{h} \right\rfloor \tag{1.24a}$$

$$j_p = \left\lfloor \frac{d_{y,p}}{h} \right\rfloor . \tag{1.24b}$$

4.c) **Determine the weights of the bilinear interpolation**: Onces we determine in location of the particle p in terms of index i_p and j_p, we can set up a bilinear interpolation problem for transferring the vorticity from the nodes of the structured grid onto the position of the particle. The bilinear interpolation of vorticity is given as,

$$\hat{\omega}(\mathbf{x}_p) = \sum_{q=1}^{4} W_{q,p} \cdot \omega_q \tag{1.25}$$

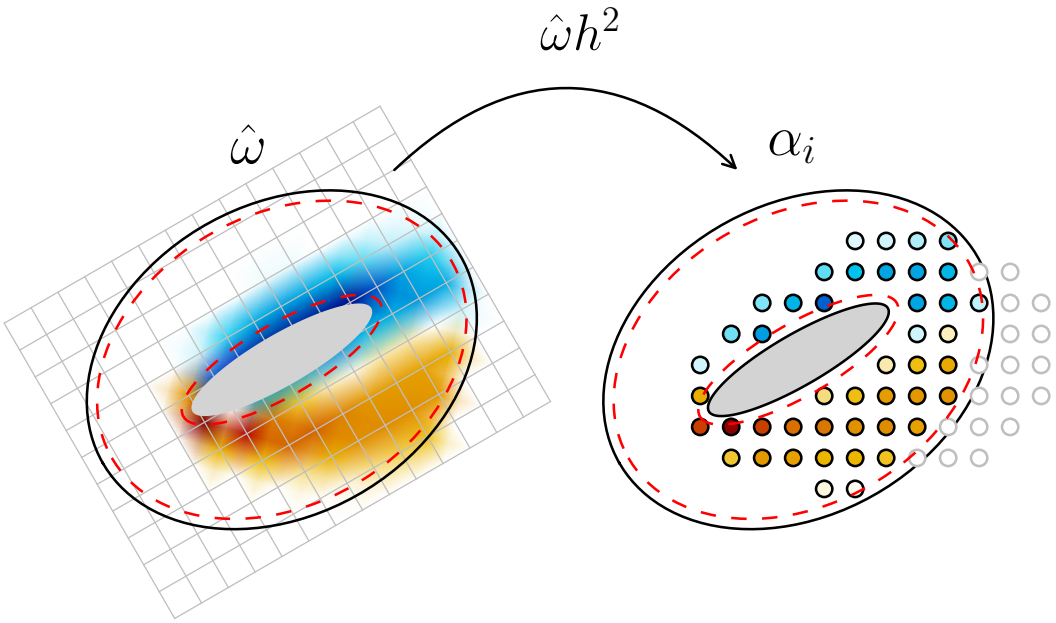


Figure 1.9: Bilinear interpolating the strengths from the structured grid SG onto the vortex blobs using equation 1.25

where $\hat{\omega}$ is the interpolated vorticity at the particle position $\mathbf{x}_p$. The weights of the bilinear interpolation $W_q p$ is given as,

$$\begin{aligned} W_{1,p} &= \frac{(d'_{x,p} - h)(d'_{y,p} - h)}{h^2}, \\ W_{2,p} &= \frac{-d'_{x,p}(d'_{y,p} - h)}{h^2}, \\ W_{3,p} &= \frac{d'_{x,p} d'_{y,p}}{h^2}, \\ W_{4,p} &= \frac{-d'_{y,p}(d'_{y,p} - h^2)}{h^2}, \end{aligned} \tag{1.26}$$

where $\mathbf{d}'_p = (d'_{x,p}, d'_{y,p})$ is given by,

$$\mathbf{d}'_p = \begin{bmatrix} d'_{x,p} \\ d'_{y,p} \end{bmatrix} = \begin{bmatrix} d_{x,p} - i_p \cdot h \\ d_{y,p} - j_p \cdot h \end{bmatrix} \tag{1.27}$$

such that $0 \leqslant d'_{x,p} \leqslant 1$ and $0 \leqslant d'_{y,p} \leqslant 1$ and reflects the offset from the four nodes.

4.d) **Interpolate the vorticity from *SG* onto the particle**: Finally, we can solve equation 1.25 to interpolate the vorticity from the nodes of the structured grid onto the pre-generated particle positions. Once we know the vorticity at their position, we can use equation 1.2 to assign the strengths of the particles. These particles will then have a total circulation of $\hat{\Gamma}_I^{L,k}$ according to equation 1.10. Figure 1.9 depicts the transfer of the Eulerian solution from the structured grid onto the particle.

However, as we described in section 1.1, we must still determine the vortex sheet strengths that satisfied a no-slip boundary condition and ensure conservation of circulation in the domain Ω_{in}^k due to the correction.

1.2.5 Correct Total Circulation

In this step, we will determine the strengths of the vortex sheet and the total circulation of newly generated vortex particles inside the correction region Ω_{in}^k such that the circulation conserved during the Lagrangian correction step. The algorithm is however depended on whether the problem is unbounded (without wall boundaries), or bounded (with wall boundary). If we do not have a wall boundary, the step to determine the vortex sheet is not applicable because there exists no vortex sheet.

The algorithm for correcting the total circulation inside Ω_{in}^k is as follows:

5.a) **Determine the strengths of the panels** (with solid wall): The total strength of the vortex panels $\Gamma_{\gamma}^{L,k}$ is determined from equation 1.11. The Eulerian circulation $\Gamma_{in}^{E,k}$ is the circulation within the boundary Σ_o^k (see Figure 1.2) and also includes the circulation within the Eulerian body Ω_b. In the case of a moving boundary, there exists a non-zero circulation within the body. The circulation $\Gamma_I^{L,k}$ is determined from step 4.d and represents the total circulation of the newly generated vortex particles inside the domain Ω_I^k. Once we determine the required strength for the vortex sheet $\Gamma_{\gamma}^{L,k}$ for all the k Eulerian domains, we can use the approach described in section **??** to solve the multi-body vortex panel problem, satisfy the no-slip boundary condition.

5.b) **Correct the strengths of particles to conserve circulation**: The final step of the *modified Lagrangian correction* algorithm is to ensure conservation of total circulation. This can be achieved by determining the adjusting the strengths of particles $\tilde{\alpha}_p$ using equation 1.19. Note that, in case we have unbounded problem, we have the strength of the vortex sheet $\Gamma_{\gamma}^{L,k} = 0$.

1.3 Determining Eulerian Substep Boundary Conditions

In this section, we elaborate the methodology for determining the Eulerian boundary conditions from the evolved solution of the Lagrangian method. Furthermore, we will explain how to determine the boundary conditions for the substeps of the evolution.

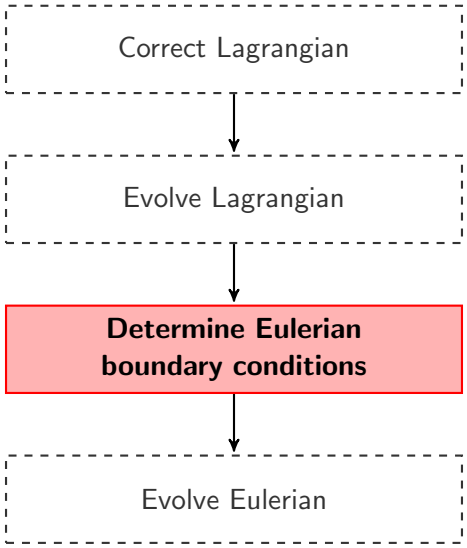


Figure 1.10: Flowchart of the hybrid evolution, focusing on the 3rd step: Determine the Eulerian boundary conditions.

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}. The velocity at the Eulerian boundary $\mathbf{x}_{bdry}$ is given by,

$$\mathbf{u}(\mathbf{x}_i, t_{n+1}) = \sum_p \mathbf{K}_\sigma[\mathbf{x}_i - \mathbf{x}_p(t_{n+1})]\alpha_p(t_{n+1}), \tag{1.28}$$

where $\mathbf{x}$ are the coordinates of the nodes of the Eulerian mesh that lie in the boundary Σ_d, as shown in figure 1.11.

1.3.1 Multi-step evolution

In section **??**, we defined the Eulerian time step Δt_E and the Lagrangian time step, such that $\Delta t_E \leqslant \Delta t_L$. The main advantage of this different time step size is that the wake region, where the Lagrangian solver lies, can now be evolved with a much larger time step size.

We perform k_E Eulerian substeps within one Lagrangian time step t_{n+1},

$$t_n^k = t_n + k\Delta t_E, \tag{1.29}$$

where $k = \{0, ..., k_E\} \subset \mathbb{N}_0$, and is given by,

$$k_E = \frac{t_{n+1} - t_n}{\Delta t_E} = \frac{\Delta t_L}{\Delta t_E}. \tag{1.30}$$

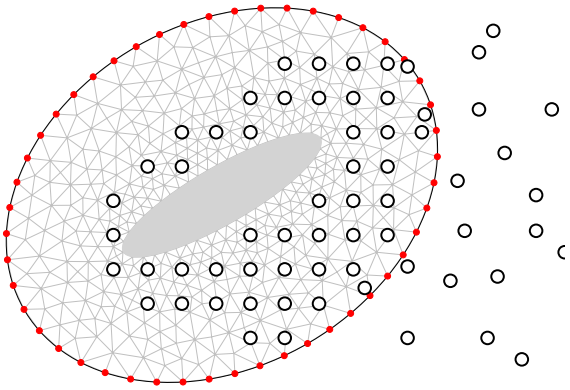


Figure 1.11: Dirichlet boundary conditions at boundary of Eulerian domain Σ_d. We evaluate the induced velocities from the Lagrangian solution at the nodes $\mathbf{x}_i \in \Sigma_d$ [●, red dot].

and we require that Δt_L is a multiple of Δt_E. When $k = 0$, we have $t_n^0 = t_n$ and when $k = k_E$, we have $t_n^{k_E} = t_{n+1}$. Figure 1.12 depicts the multi-stepping of the Eulerian solution from t_n^0 to $t_n^{k_E}$ to match the Lagrangian time. As the Eulerian solver requires boundary conditions at each substep, we have to perform an interpolation of the boundary conditions for each substep t_n^k. We can perform a linear interpolation of the boundary condition in order to determine the boundary conditions at each sub-step,

$$\mathbf{u}(t_n^k) = \mathbf{u}(t_n) + k\Delta\mathbf{u}, \tag{1.31}$$

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

$$\Delta\mathbf{u} = \frac{\mathbf{u}(t_n^{k_E}) - \mathbf{u}(t_n^k)}{k_E}, \tag{1.32}$$

and is a linear variation of the velocity between each substep.

Once that we have the boundary condition for each of the Eulerian substeps, we can use the approach described in section **??** to evolve the Eulerian method from t_n to t_{n+1} using k_E Eulerian substeps.

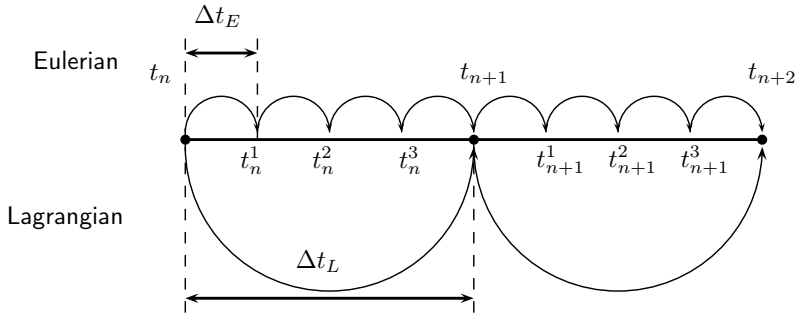


Figure 1.12: Eulerian multi-stepping to match the Lagrangian time step. The figure shows $\Delta t_L = 4\Delta t_E$ and requires $k_E = 4$ Eulerian substeps to time march from t_n to t_{n+1}.

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] Renac, F., Gérald, S., Marmignon, C., and Coquel, F. Fast time implicit-explicit discontinuous Galerkin method for the compressible NavierStokes equations. *Journal of Computational Physics 251* (Oct. 2013), 272–291.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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).

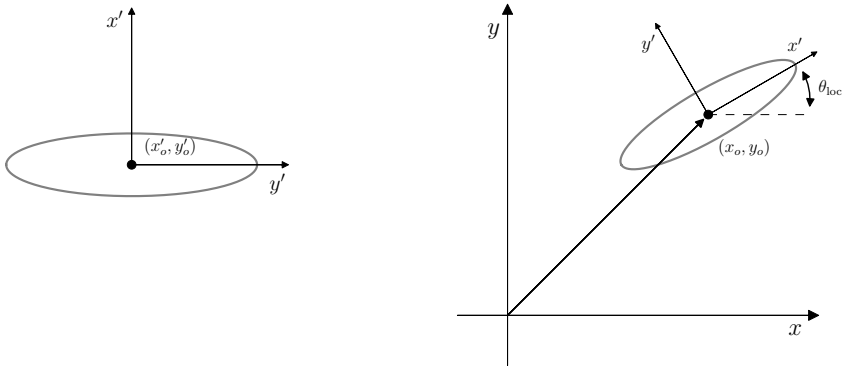
[18] Tryggeson, H. *Analytical Vortex Solutions to the Navier-Stokes Equation.* PhD thesis, Växjö University, Sweden, 2007.

[19] Vermeer, L., Sø rensen, J., and Crespo, A. Wind turbine wake aerodynamics. *Progress in Aerospace Sciences 39*, 6-7 (Aug. 2003), 467–510.

Appendix A

Coordinate Systems

The geometries in the simulation are defined in their respective local coordinate systems $[x', y']$. Figure A.1a shows an elliptical geometry defined in its local coordinate system about its origin. The local origin point is defined such that it is the center of rotation and any rotation will be prescribed about this point.



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

Figure A.1: Ellipse defined 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.

The body is then transformed to the global coordinate system $[x, y]$ by the displacement vector $\mathbf{x}_o = [x_o, y_o]$ (i.e the global position of the local origin) and a local rotation by θ_{loc} about the local origin $[x'_o, y'_o]$. The transformation of an arbitrary point $\mathbf{x}'_p = [x'_p, y'_p]$ in the local coordinate system to the global coordinate system is given as,

$$\mathbf{x}_p = \begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} \cos\theta_{loc} & -\sin\theta_{loc} \\ \sin\theta_{loc} & \cos\theta_{loc} \end{bmatrix} \cdot \begin{bmatrix} x'_p - x'_o \\ y'_p - y'_o \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix} \tag{A.1}$$

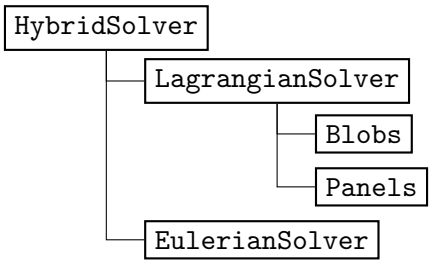
where $\mathbf{x}_p = [x_p, y_p]$ is the global position of the point. The Eulerian solver defines the body mesh in the local coordinate system is then transformed to global position using these parameters. Similarly, the panel geometry for the Lagrangian solver is defined in the same fashion. For a moving bodies problem, the displacement vector and the local rotation angle can be updated to prescribe the motion.

Appendix B

pHyFlow Code Structure

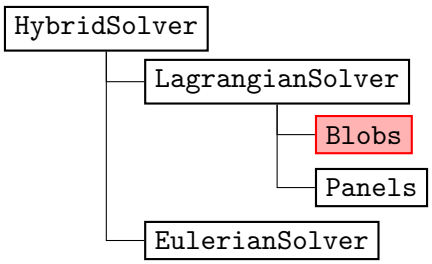
The document outlines the pHyFlow code structure. The pHyFlow functions are organized into several classes. The functions related to the vortex particles are placed inside the Blobs class. The functions related to the panel problem are inside Panels class. The LagrangianSolver class is made to couple 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:

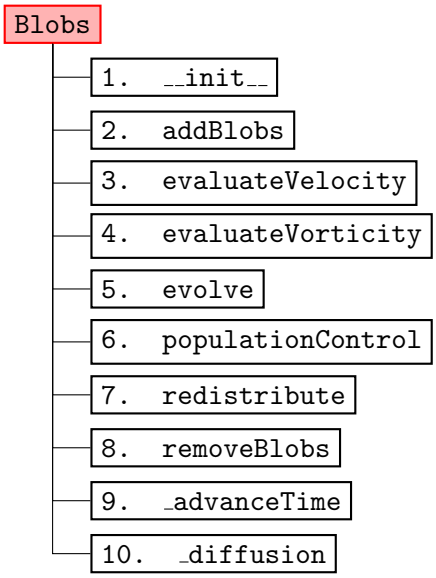


Blobs Class

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 B.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.	-

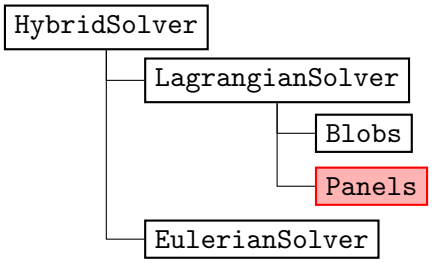
Time step parameters		*Default*
`deltaTc`	: the size of the convective time step Δt_c.	-
`nu`	: the fluid kinematic viscosity ν, used to calculate the diffusion coefficient c^2 and diffusion time step size ΔT_d.	-
`stepRedistribution`	: the redistribution step frequency.	1
`integrationMethod`	: the time 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}.	-

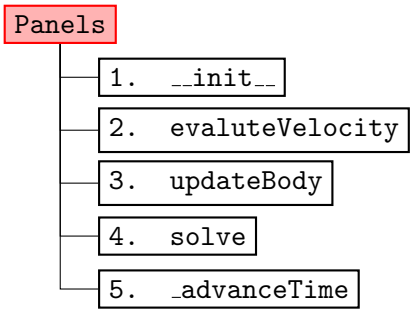
Free stream velocity		*Default*
`vInf`	: The free-stream velocity function, returning the velocity action on the vortex blobs.	-
	Input parameters : `t`	
	Assigns : `-`	-
	Returns : `vx,vy`	

`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 B.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.

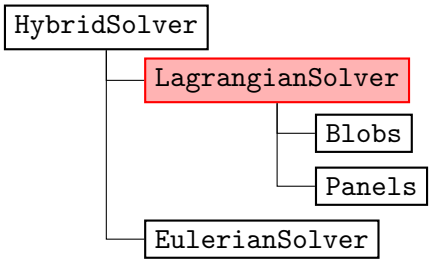
External velocity

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.

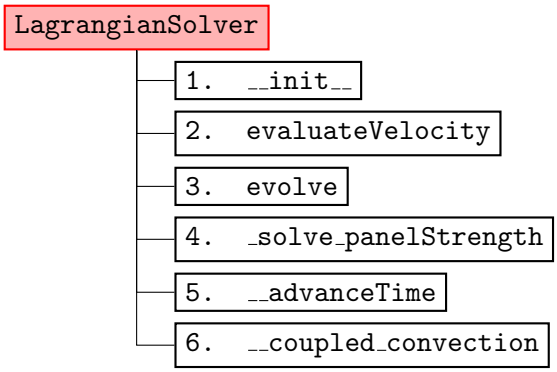
Input parameters : xCP,yCP
Assigns : -
Returns : vxCP,vyCP

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 B.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.

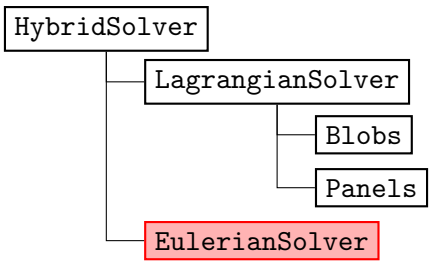
Input parameters:

`Blobs`: vortex particle class

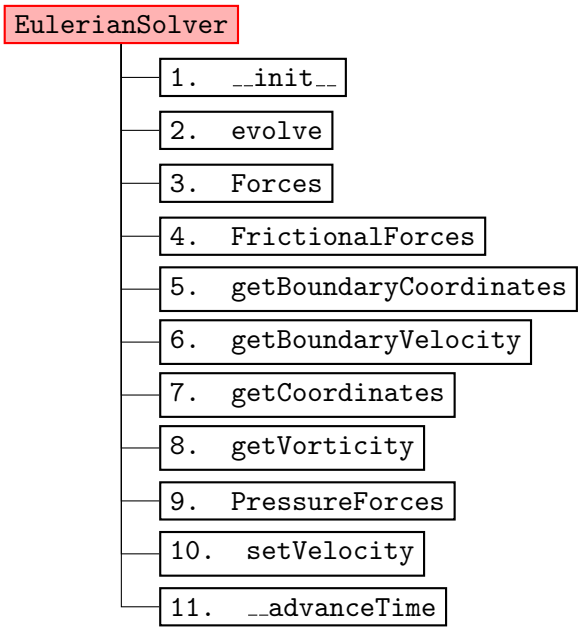
`Panels`: panel method class

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 B.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

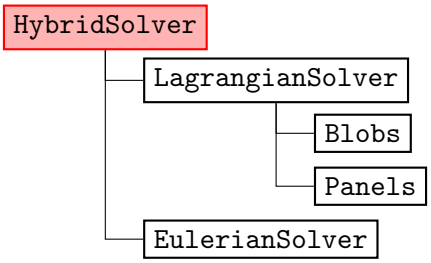
cfl	: the CFL stability parameter. If explicit time marching scheme, $CFL < 1$.

Probe grid parameters

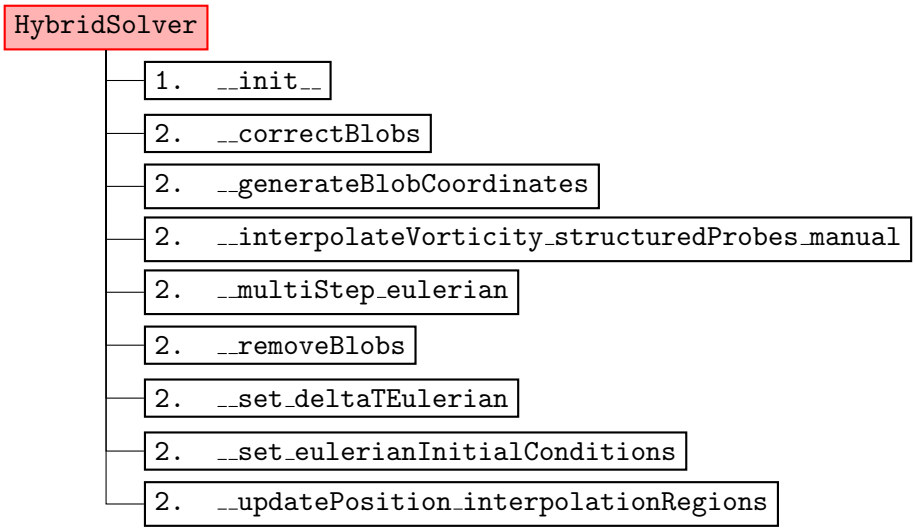
`x0,y0`	: the x, y coordinate of the origin of the probe grid.
`Lx,Ly`	: the x, y size (width and height) of the probing grid.
`hx,hy`	: the x, y spacing of the probe grid cell.

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 B.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`