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 $\Omega_E \subset \Omega_L$.

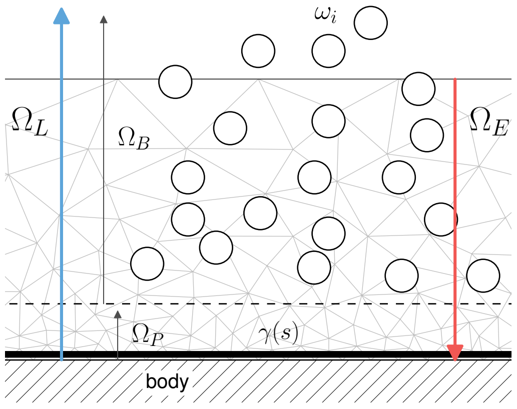


Figure 4.1: Schematic of the domain decomposition. The two subdomain are the Lagrangian domain $\Omega_L : \Omega_p \cup \Omega_b$ and the Eulerian domain Ω_E where $\Omega_E \subset \Omega_L$.

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:

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

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 $\Omega_p : \Omega_p \in [\partial\Omega_{body}, \partial\Omega_p]$.

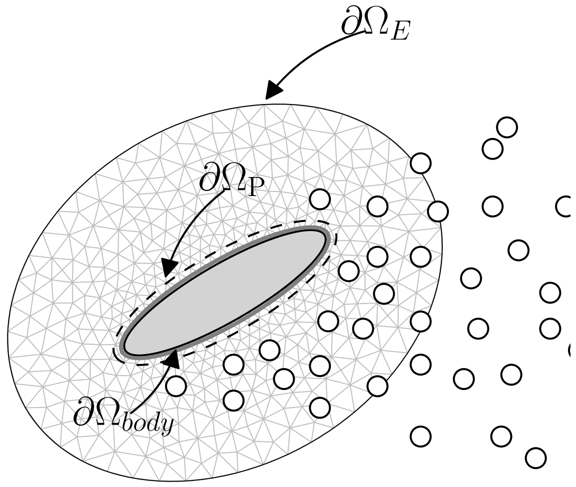


Figure 4.2: Boundaries of the decomposed domains. No-slip boundary $\partial\Omega_{body}$, exterior vortex panel boundary $\partial\Omega_p$, exterior Eulerian boundary $\partial\Omega_E$.

- 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.

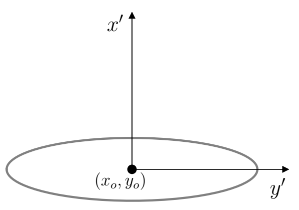
The solutions in the overlap region $\Omega_E \cap \Omega_L$ will be used to couple the Eulerian and the Lagrangian solver, based on the procedures of Stock [59] and the Daeninck [23].

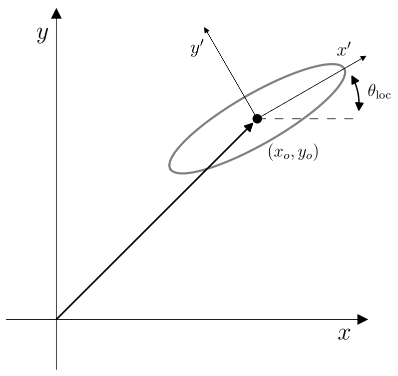
4.1.1 Local to Global Transformation

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 $\Omega_{int} : \Omega_{int} \subset \Omega_E \cap \Omega_b$.

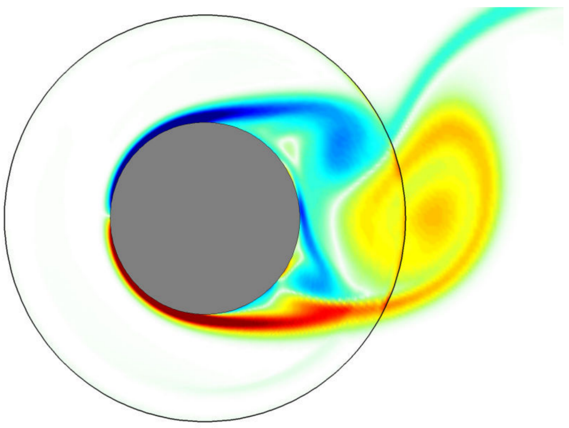
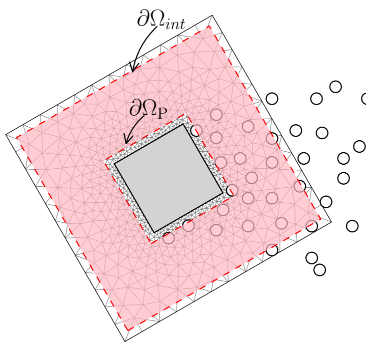


Figure 4.4: Result of hybrid coupling by Daeninck [23]. The figure shows artificial vorticity at the boundary of the Eulerian domain.

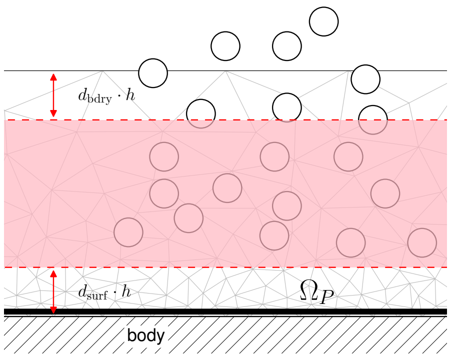
- The region is starts from the outer vortex panel boundary $\partial\Omega_p$. All the vorticity of the vortex panel boundary is represented using the vortex panels. The interpolation region ends at $\partial\Omega_{int}$, slight distance away from the outer Eulerian boundary $\partial\Omega_E$ ignoring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

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

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.

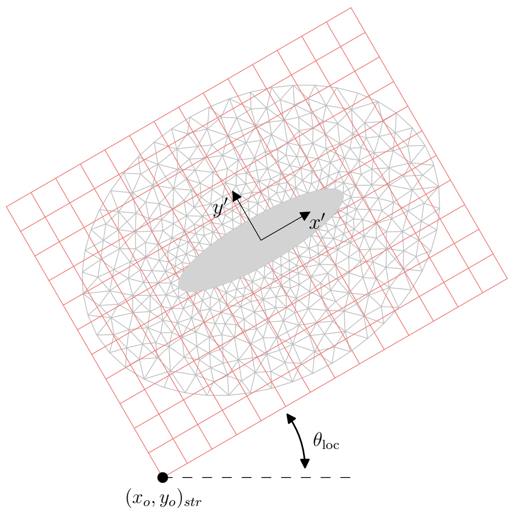


Figure 4.6: Structured interpolation grid $\mathbf{x}_{str}$ (pink) covering the entire Eulerian mesh (gray).

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.

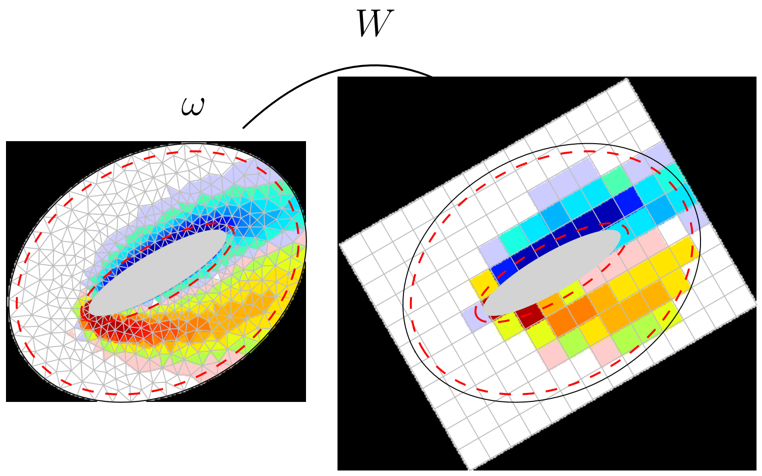
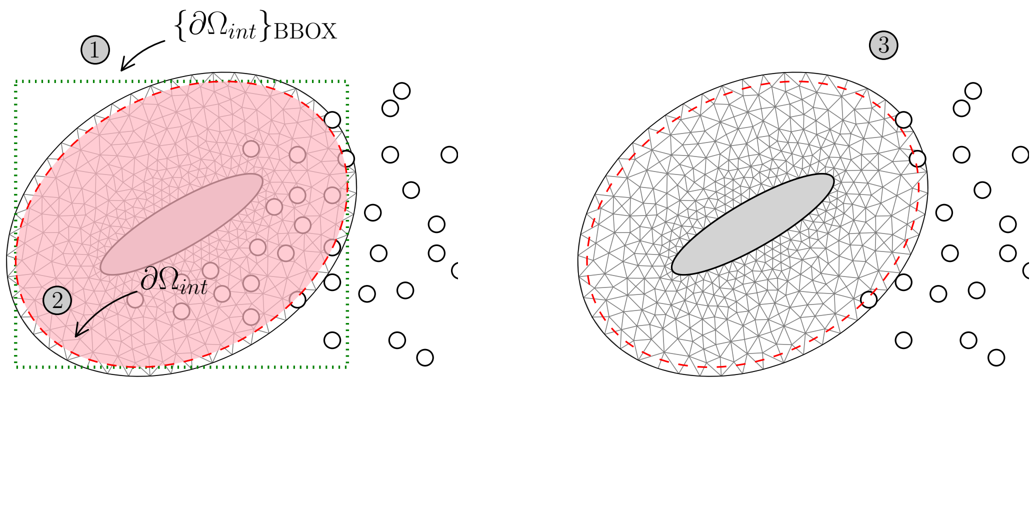


Figure 4.7: Interpolated vorticity $\hat{\omega}$ on the structured grid $\mathbf{x}_{str}$ from interpolating ω of the unstructured grid $\mathbf{x}_{unstr}$ with the interpolation weights W.

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 $\mathbf{x}_{str}$.

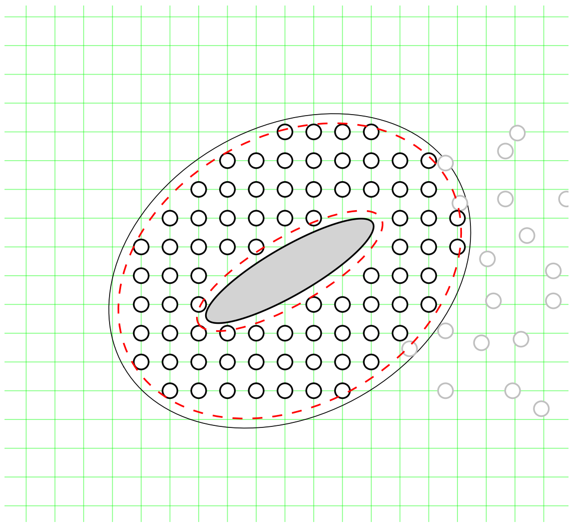


Figure 4.9: Particles inside the interpolation domain Ω_{int} located at $\mathbf{x}_i$ coinciding the Lagrangian remeshing grid.

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

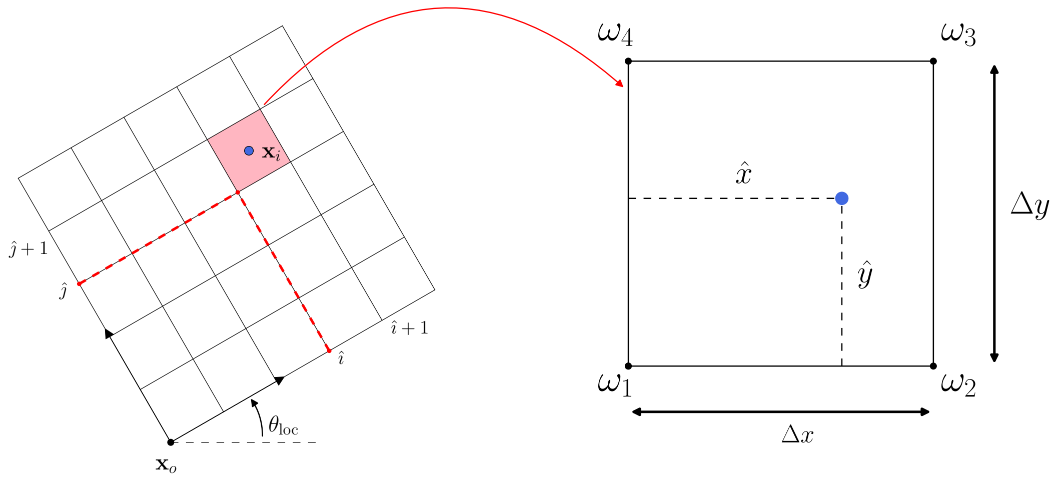


Figure 4.10: Interpolating the strengths from the structured grid $\mathbf{x}_{str}$ onto the vortex blobs $\mathbf{x}_i$ using a bilinear interpolation

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.

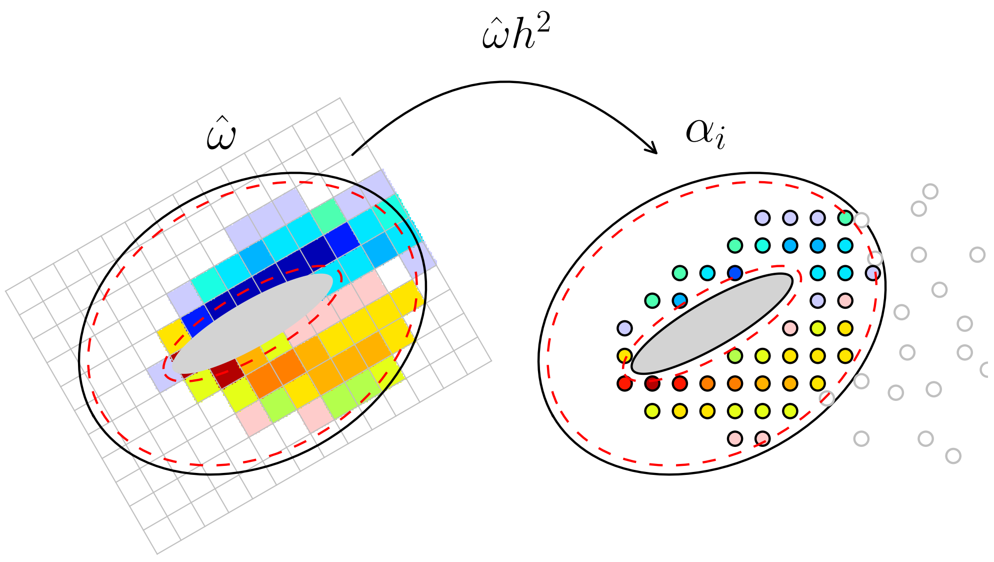


Figure 4.11: Interpolated strengths α_i from the structured grid $\mathbf{x}_{str}$ using bilinear interpolation.

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}.

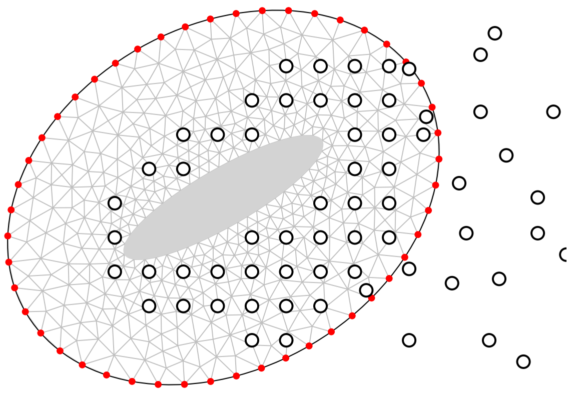


Figure 4.12: Dirichlet boundary conditions at boundary of Eulerian domain $\mathbf{u} \in \partial\Omega_E$. We evaluate the induced velocities from the Lagrangian solution at the nodes of the boundary [●, red dot].

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 2.2.2, 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,

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

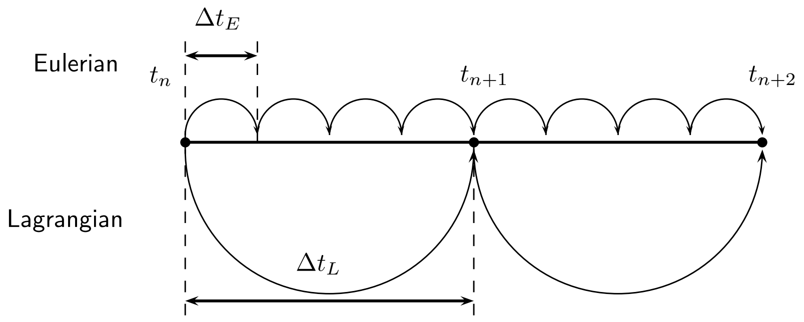


Figure 4.13: Eulerian multi-stepping to match the Lagrangian Δt_L. The figures shows $\Delta t_L = 4\Delta t_E$ and required $k_E = 4$ iterations to time march from t_n to t_{n+1}.

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.

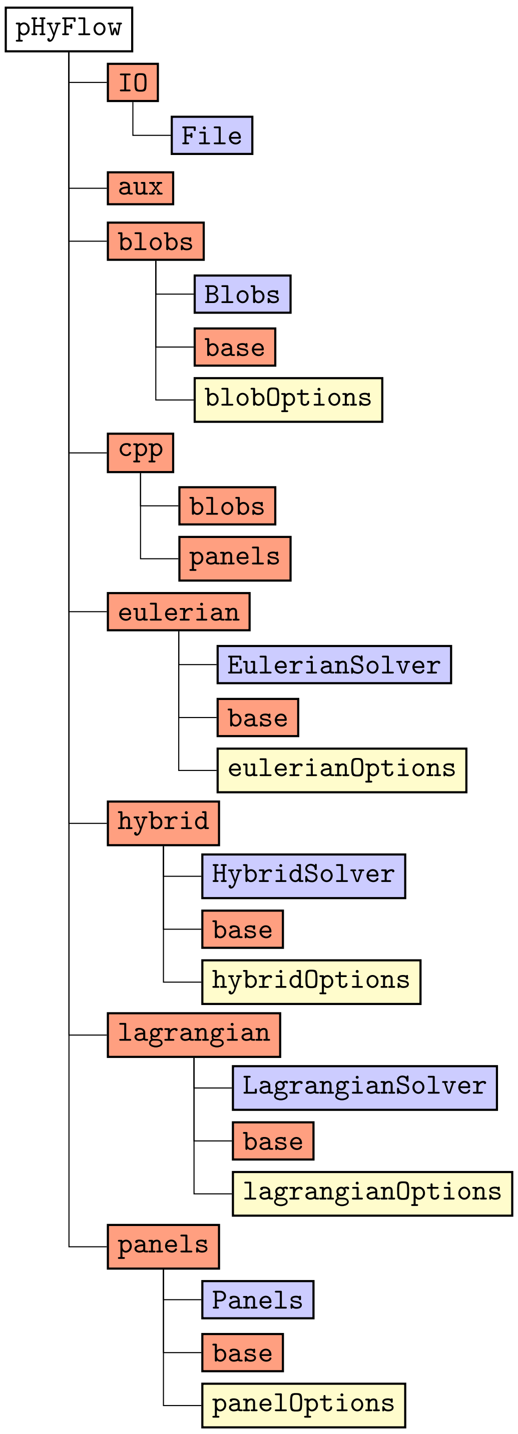


Figure 4.14: Flowchart of the `pHyFlow` library structured into modules, option script files, and classes.

- `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.

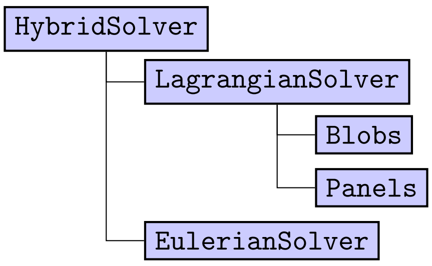


Figure 4.15: Flowchart of the `HybridSolver` hierarchy. The `HybridSolver` couples the `LagrangianSolver` class and the `EulerianSolver` class using the hybrid coupling schemes.

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

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.

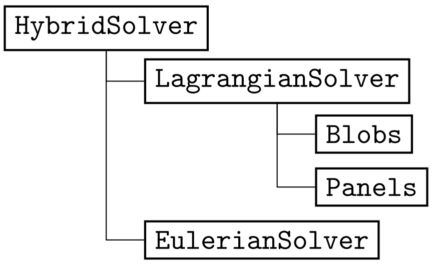
[66] Winckelmans, G., Cocle, R., Dufresne, L., and Capart, R. Vortex methods and their application to trailing wake vortex simulations. *Comptes Rendus Physique 6*, 4-5 (May 2005), 467–486.

Appendix A

pHyFlow **Code Structure**

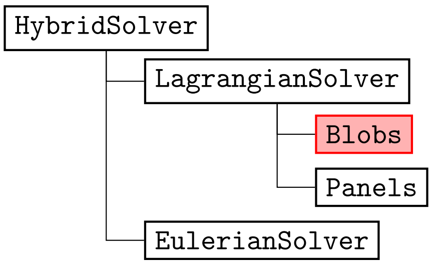
The document outlines the pHyFlow code structure. The pHyFlow functions are organized into several classes. The functions related to the vortex particles are placed inside the Blobs class. The functions related to the panel problem are inside Panels class. The LagrangianSolver class is made to couple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

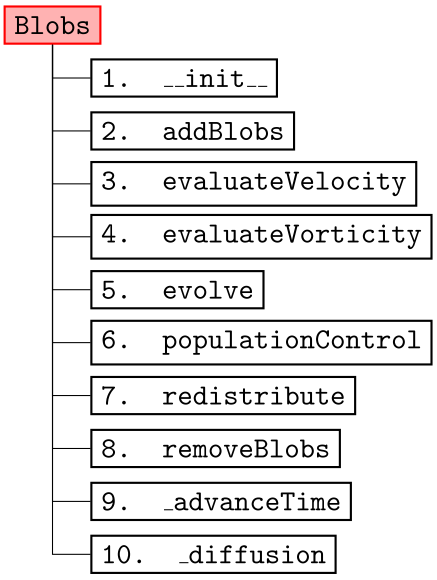


A.1 Blobs Class

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.

A.1.1 `__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.	-

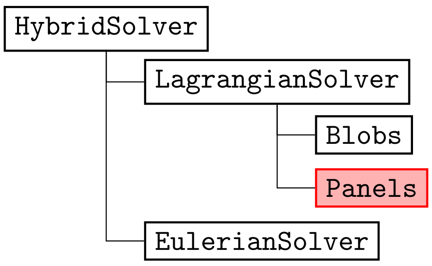
Time step parameters		*Default*
`deltaTc`	: the size of the convective time step Δt_c.	-
`nu`	: the fluid kinematic viscosity ν, used to calculate the diffusion coefficient c^2 and diffusion time step size ΔT_d.	-
`stepRedistribution`	: the redistribution step frequency.	1
`integrationMethod`	: the time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

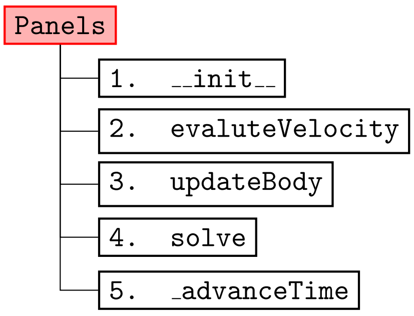
Free stream velocity		*Default*
`vInf`	: The free-stream velocity function, returning the velocity action on the vortex blobs. **Input parameters** : `t` **Assigns** : `-` **Returns** : `vx,vy`	-

A.2 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 **A**, the LHS of the problem.
`cmGlobal`	The global position vector for each of the **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 **M** panel corners.), `yPanel` (The y coordinate of the **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 **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.

A.2.1 `__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.

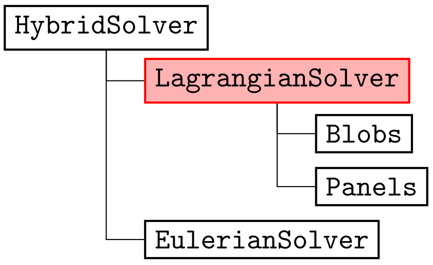
External velocity

`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`.

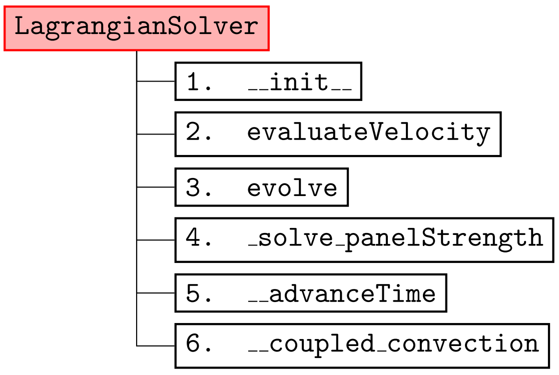
Input parameters	: `xCP,yCP`
Assigns	: -
Returns	: `vxCP,vyCP`

A.3 `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.

A.3.1 `__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.

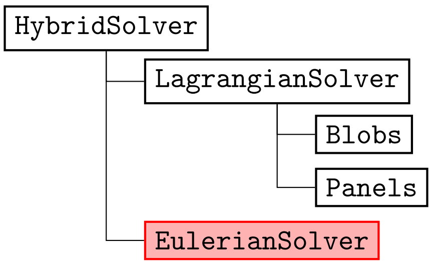
Input parameters:

`Blobs`: vortex particle class

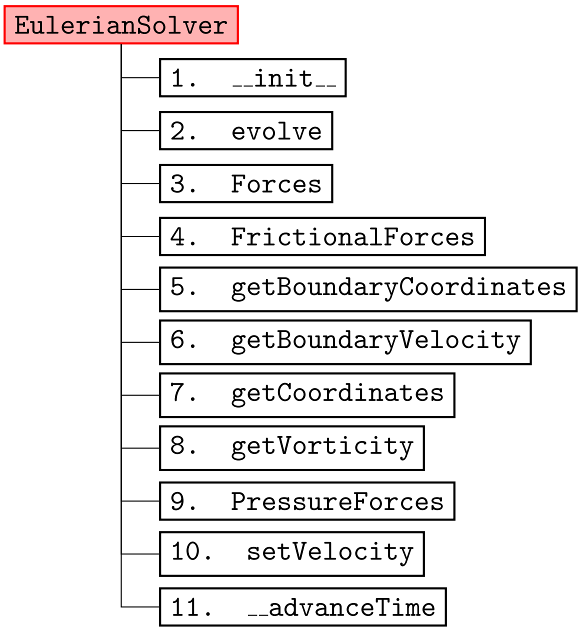
`Panels`: panel method class

A.4 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.

A.4.1 `__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.

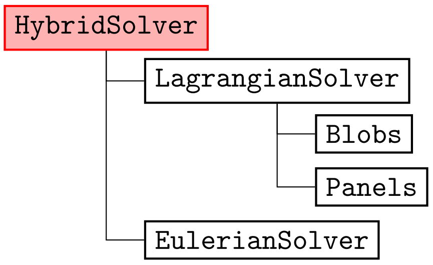
Solver options

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

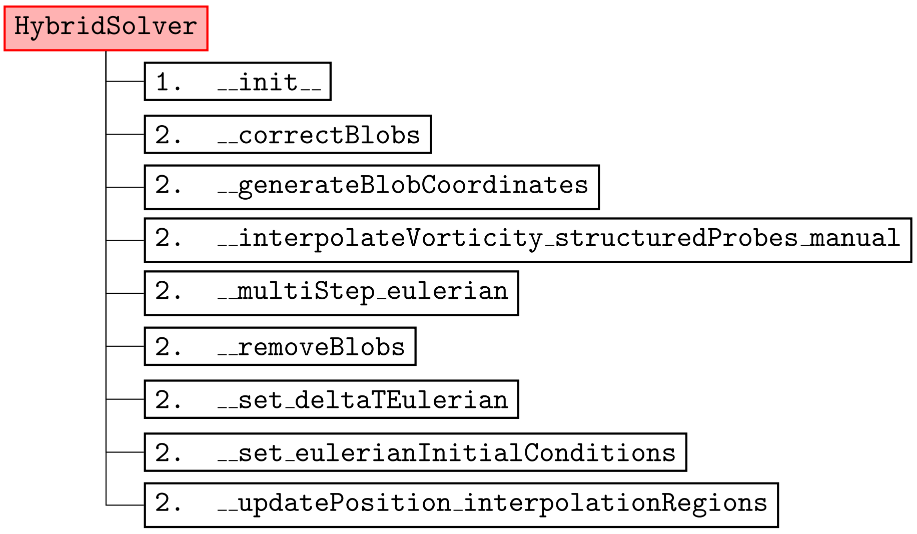
Probe grid parameters	
`x0,y0`	: the x, y coordinate of the origin of the probe grid.
`Lx,Ly`	: the x, y size (width and height) of the probing grid.
`hx,hy`	: the x, y spacing of the probe grid cell.

A.5 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:


HybridSolver
1. __init__
2. __correctBlobs
2. __generateBlobCoordinates
2. __interpolateVorticity_structuredProbes_manual
2. __multiStep_eulerian
2. __removeBlobs
2. __set_deltaTEulerian
2. __set_eulerianInitialConditions
2. __updatePosition_interpolationRegions

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.

A.5.1 __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`