AMSC 664: Final Report:

Introducing Topography to Large-Eddy Simulations

Victoria Whitley

May 13, 2020

1 Introduction

1.1 Background

As computer power grows, along with the introduction of new technology, computational fluid dynamics (CFD) allows researchers to explore fluid related questions without expensive prototypes or complicated experiments. Many important applications of CFD involve the crucial inclusion of turbulence in the system. In the ocean, turbulence ranges from eddies with scales set by the size of the turbulent boundary layers (orders of 10s of meters), down to the molecular level where viscosity dissipates energy. Large-eddy simulation (LES) is one of the most promising methodologies for simulating turbulent flows. LES resolves the largest eddies and eliminates the smaller motions through physically motivated models, allowing for a computationally feasible and accurate method.[22] Unfortunately, the complex geometry in the flows involved in many scientific inquiries can be challenging and costly for most LES systems.[7]

One of the largest applications of CFD, modeling the ocean and atmosphere, often requires implementing aspects of LES. With recent pushes for updating the commonly used modeling practices and parameters, the Climate Modeling Alliance (CliMA) aims to build a climate model from scratch that will simulate small-scale ocean physics at high resolutions.[21] As part of this project, they are developing the "fast and friendly" software package, Oceananigans, to solve the incompressible fluid problems of the ocean.[21] Written in Julia, the package supports LES and Direct Numerical Simulations on both CPUs and GPUs. Oceananigans aims to create a user-friendly interface, allowing researchers to "focus on the science," while relying on algorithms that will run as fast as possible.[21] While Oceananigans has a functioning LES package, the group continues to improve the algorithms and add more capabilities for scientists to utilize.[24] However, they lacked the ability to represent solid boundaries in the fluid, severely limiting its utility for scientific problems involving flow-topography interaction.

Through this work, we implement the immersed boundary method (IBM) within Oceananigans to assist in answering the many research questions involving flow along and over topography. The aspects of the implementation were chosen to best benefit the application of the method to geophysical problems. We found that most of the literature focused on more engineering aspects, which will have different goals and complexities than that of ocean flow.[8,19,20] Through studying flow around a cylinder, we found good agreement between our implementation and that of other numerical and experimental results.[8,20] Due most likely to our choices favoring geophysical applications, we did encounter some differences in our results compared to similar IBM codes.[20] Some of these differences, showed better convergence and accuracy, while other aspects were not as favorable. However, taking into consideration the accuracy of the parameterized models that ocean simulations already rely on, we have successfully created a tool for scientists to use in their studies of topography in turbulent flows.

1.2 Motivation

An important aspect of ocean modeling is the inclusion of topography in the system. The interaction of terrain with the ocean could be one of the major pathways of extracting energy from mesoscale flows (10-100 km), creating unbalanced submesoscale (0.1- 10 km) turbulence.[10] The influx of energy into these smaller flows can lead to elevated local dissipation and mixing outside of the oceanic boundary layers, of significance to the energy budget for general circulation.[10] While, in horizontal flow, the bottom boundary layer (BBL) is usually relegated to just a thin layer near the bottom, if you introduce more realistic topography, such as a sloping bottom, this could separate through increased instability. The topography generates vertical vorticity or relatedly, low potential vorticity, which is susceptible to submesoscale instabilities. Once unstable, the vorticity can generate secondary instabilities, that will then enhance turbulence.[17,27] This effect can be seen in the results of a simulation from Gula,[10] shown in Figure (1), where currents flowing along the slope of the Great Bahamas Banks generate negative potential vorticity within the BBL (a). This layer separates from the slope (b) and mixes with the interior, causing instabilities and energy dissipation (c,d).[10] These dynamics due to topography interactions could also impact the mixing of nutrients or oxygen from the bottom of the ocean.[28] To consider these topography related problems, one must use a turbulence resolving simulation such as LES, but currently there are not many options for scientists to add terrain in LES. By developing a way to implement topography into LES that is widely available and easy to use, we can assist many scientists in their inquiries into these physical scenarios and give Oceananigans an advantage over many LES codes currently in use.

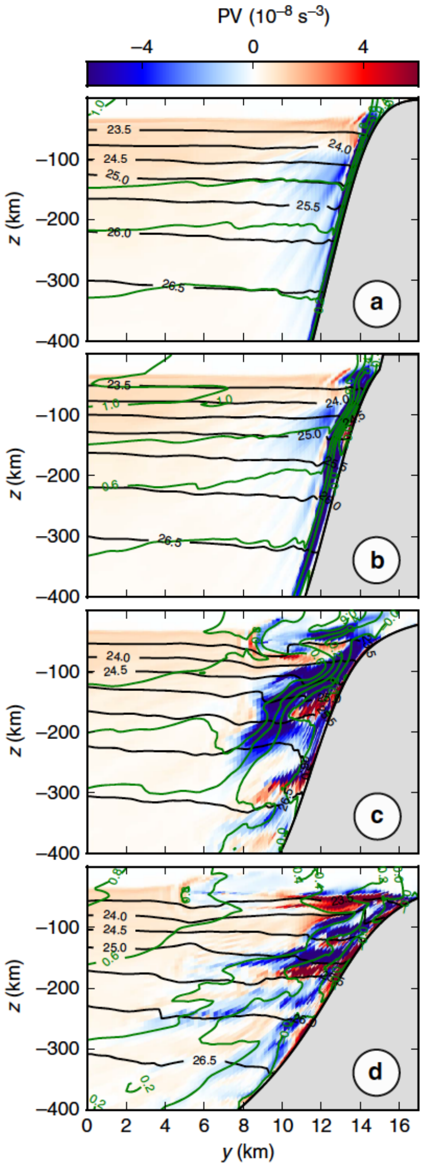


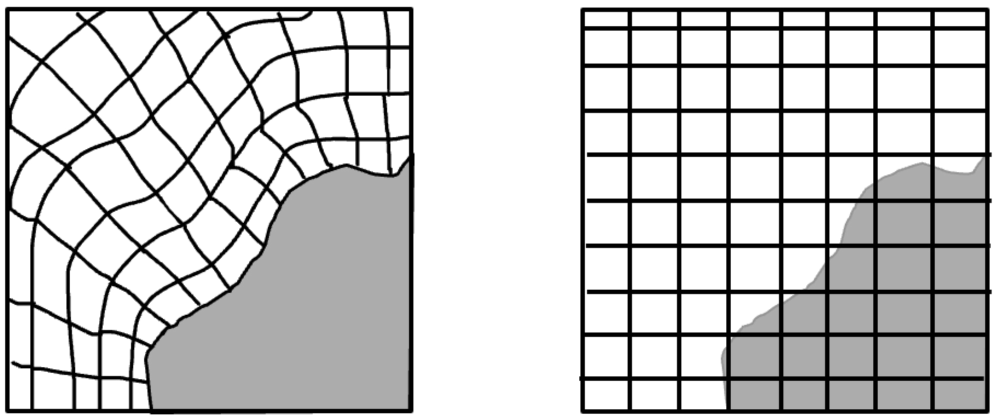
Figure 1: Vertical sections of potential vorticity along the Great Bahamas Banks, adapted from Gula

1.3 Methodology

1.3.1 Immersed boundary method

One alternative to the complicated methodologies used in modeling complex geometries and topography is the immerse boundary method (IBM). In this method, instead of using meshes to conform to the terrain or solid body, as depicted in Figure (2a), topography is represented by an "immersed boundary" that can arbitrarily intersect a standard Cartesian grid, as in Figure (2b). Then the boundary conditions are applied along the IB by modifying the governing equations through a forcing term.[1,19] In this way, you can employ many types of boundary conditions onto whatever shape the surface has, while still maintaining a simple Cartesian grid, easily refined and implemented.[19] By implementing IB methods in LES code, we can study these important questions involving small-scale ocean interactions with topography.

Figure 2: Sketch of IBM vs Common Alternative



(a) Grid generated to conform to solid body **(b)** Grid generated regardless of immersed object

2 Implementation

2.1 Oceananigans Temporal Discretization

To discuss the changes we have made to include IBM, first we must look at how Oceananigans solver works without IBM. Consider the typical spatially-filtered, incompressible Boussinesq equations,[21] with the inclusion of an arbitrary tracer,

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + (f - \nabla \times \boldsymbol{u}^s) \times \boldsymbol{u} = -\nabla\phi + b\hat{\boldsymbol{z}} - \nabla \cdot \boldsymbol{\tau} - \partial_t \boldsymbol{u}^s + \boldsymbol{F}_u \tag{1}$$

$$\partial_t c + \boldsymbol{u} \cdot \nabla c + U \cdot \nabla c + \boldsymbol{u} \cdot \nabla C = -\nabla \cdot q_c + F_c \tag{2}$$

$$\nabla \cdot \boldsymbol{u} = 0. \tag{3}$$

This gives the equation for the velocity vector $\boldsymbol{u}$, including f, the Coriolis parameter, b, buoyancy, $\boldsymbol{\tau}$, the kinematic stress tensor, and ϕ the the potential associated with both kinematic and constant hydrostatic contributions to pressure. The superscript, s, indicates stokes drift, and $\boldsymbol{F}$ indicates internal forcing on the associated field. In the tracer equation for c, capital letters represent the background fields and q, the diffusive flux.

After discretization, the time-integral of the momentum equation is given by the following,[21]

$$\begin{aligned}
\boldsymbol{U}^{n+1} - \boldsymbol{U}^{n} &= \int_{t_n}^{t_{n+1}} \Big[-\boldsymbol{\nabla}\phi_{non} - \boldsymbol{\nabla}\phi_{hyd} - (\boldsymbol{U}\cdot\boldsymbol{\nabla})\boldsymbol{u} - f\times\boldsymbol{U} + \boldsymbol{\nabla}\cdot\boldsymbol{\tau} + \boldsymbol{F_u} \Big] dt \\
&= \int_{t_n}^{t_{n+1}} \Big[-\boldsymbol{\nabla}\phi_{non} + \boldsymbol{G_u} \Big] dt \\
&\approx -\Delta t \boldsymbol{\nabla}\phi_{non}^{n+1} + \int_{t_n}^{t_{n+1}} \boldsymbol{G_u} dt.
\end{aligned}$$

Oceananigans implements a fractional step method where non-hydrostatic pressure, $\boldsymbol{\nabla}\phi_{non}$, is treated implicitly while the rest of the terms on the right hand side, defined as the tendency, $\boldsymbol{G_u}$, are treated explicitly.[21] The non-hydrostatic pressure is calculated to ensure that the velocity field at $n+1$ is divergence-free. Oceananigans uses a third order Runge-Kutta scheme with pressure correction, produced by Le and Moin.[15] At the kth stage of the time stepper, the velocity is updated by

$$\boldsymbol{U^k} = \boldsymbol{U^k} + \gamma^k \Delta t \boldsymbol{G}^k + \zeta^k \Delta t \boldsymbol{G}^{k-1}, \tag{4}$$

where $\boldsymbol{G}^k$ is the tendency calculated prior to the kth stage of the time stepper and γ^k, ζ^k are Runge-Kutta coefficients. Then the velocity is corrected for the non-hydrostatic pressure by

$$\boldsymbol{U^k} = \boldsymbol{U^k} - \boldsymbol{\nabla}\phi_{non}^k \Delta t(\gamma^k + \zeta^k). \tag{5}$$

For the tracer equations, the model time steps with the same Runge-Kutta method, but ignores the pressure correction step, as pressure is not included in those equations. It is within this part of the model that the IBM forcing is added, between the Runge-Kutta sub-step and the pressure correction of the velocity.

Oceananigans also implements a staggered Arakawa C-grid which shifts the evaluation of the velocity quantities on the edges of the tracer equations, as seen in Figure (3).[21] This will add complications to our IBM implementation that will be discussed in the following sections.

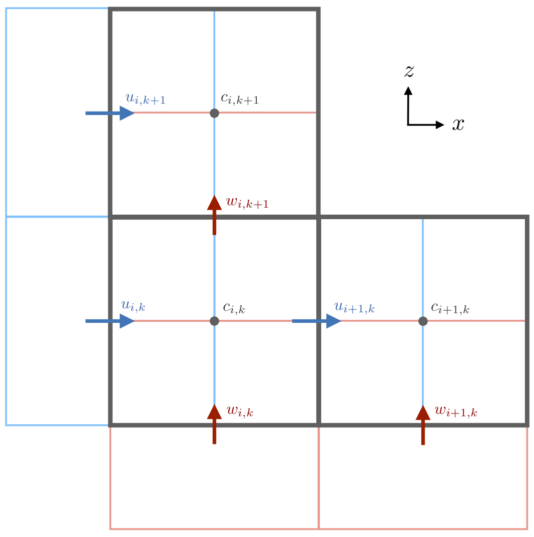


Figure 3: Depiction of the staggered Arakawa C-Grid used by Oceananigans[21]

2.2 IBM Implementation

This section describes the implementation of IBM within Oceananigans. The algorithm based on this description can be found in Appendix A.

2.2.1 Treatment at the boundary

After time stepping the velocities and tracers forward during a certain stage of the Runge-Kutta method, we want to alter the equations, so that the desired immersed boundary conditions hold. Other IBM versions implement a predictive step, and correct the tendency terms before the time stepping.[2,19] We found that this adds both computational cost and time, without quantitative differences in the results, so we used the ordering above, "masking" the quantities in the model. Unfortunately, the location of the boundary does not often coincide with the nodes in complex geometries, meaning that our boundary will cut through cells in between nodes. To enforce the boundary conditions on the desired boundary, we must use interpolation and alter nearby nodes instead. These nearby nodes are called "immersed" nodes.

2.2.2 Classification of nodes

As discussed above, given the location of the solid surface, we need to be able to classify which nodes are immersed nodes, solid nodes, and fluid nodes. Since there will also be geometric calculations later on where it will be valuable, users give the model a signed distance function, $\phi(\boldsymbol{x})$ that returns the shortest distance to the given boundary and the normal vector. If the coordinate is within the solid (fluid), the distance will be negative (positive). The immersed nodes, are then classified as solid nodes (negative) that have at least one neighboring fluid node (positive). Given a solid node, one can check the surrounding six nodes that share edges with the solid node, the "neighbors" of the node, for any fluid nodes. Since the model is not aware of the immersed solid, for higher order advection schemes one can include several layers of immersed nodes within the solid, merely by widening the radius of what is considered a "neighboring node." Note, that the velocities are enforced on the faces of the cell, as in Figure (3), with tracers in the center. For an arbitrary boundary, node identification must be performed separately for each velocity component and cell centered tracer, to better represent the boundary on a staggered grid.[20] For a stationary boundary, classification could be performed once at the beginning and one could store the values for use later on. Unfortunately, this adds considerable storage, which for use on GPUs could be problematic for larger simulations. For this implementation, the classification is done at every time step.

2.2.3 Reflection over the boundary

Once the nodes are classified, we can then work specifically with the immersed nodes to force the boundary. Consider a boundary, that arbitrarily intersects the grid, such as the one depicted in Figure (4). One layer of immersed nodes, based on the classification in Section 2.2.2, are represented by the letter I. We will focus on the immersed node, $\boldsymbol{x}_I$, denoted in the figure. We want to determine a value to enforce, so that the boundary condition is correct at the nearest point along the boundary, marked in blue as point $\boldsymbol{x}_b$. The boundary point is determined by the intersection of the boundary and the surface normal vector $\boldsymbol{n}$ that passes through $\boldsymbol{x}_I$. By using the distance function from our classification, we can calculate the surface normal at a point as $\nabla\phi(\boldsymbol{x})$. To interpolate, we also need physically motivated velocity in the fluid. This is done by reflecting the node $\boldsymbol{x}_I$ over the boundary in the direction of $\boldsymbol{n}$ into the fluid, thereby obtaining the point $\boldsymbol{x}_R$, in red. While $\boldsymbol{x}_R$ is not necessarily a node, we have surrounding fluid nodes, F_r, such that we can interpolate the value at $\boldsymbol{x}_R$. Then using the boundary condition and the value at $\boldsymbol{x}_R$, we can determine the required quantity at $\boldsymbol{x}_I$ to ensure the boundary condition holds.[20]

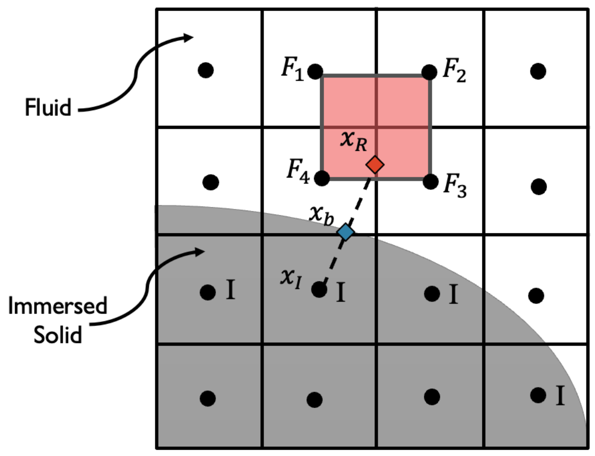


Figure 4: Immersed solid with classified immersed nodes and interpolation over the boundary

There are several popular interpolation techniques, but we will use the bilinear (trilinear for 3D) interpolation that Nasr-Azadani and Meiburg employ.[20] This gives us second order spatial accuracy and good stability for the numerical solution. Then for a quantity of interest, q, the value at reflected node, $\boldsymbol{x}_R = (x_R, y_R)$, in 2D is given by

$$q_R = \sum_{j=1}^{4} w_j q_j, \tag{6}$$

with interpolation coefficients of

$$w_1 = \alpha\beta \qquad w_2 = (1-\alpha)\beta$$
$$w_3 = \alpha(1-\beta) \qquad w_4 = (1-\alpha)(1-\beta)$$

where $\alpha = \frac{x_{i+1}-x_R}{x_{i+1}-x_i}$ and $\beta = \frac{y_{j+1}-y_R}{y_{j+1}-y_j}$. The indices i, j are given by the fluid node to the left and to the south respectively, of the point $\boldsymbol{x}_R$, as seen by the node F_4 in Figure (4). Just as node classification will have to take place separately for each velocity and cell-centered quantities, due to the staggered grid, we will have to determine these interpolation coefficients and reflected values independently for each quantity of interest.

2.2.4 Projection into tangential and normal components

For the geophysical applications of this work, the velocity boundary conditions are more useful if broken into tangential and normal components instead. One always knows that they do not want flow entering the ground (zero normal velocity), and researchers often want to include some kind of forcing in the tangential direction that can mimic drag along the surface. Therefore, for this implementation, we project the velocity components $\boldsymbol{V} = \langle u, v, w \rangle$ into tangential and normal components $\boldsymbol{V} = \langle V^{t1}, V^{t2}, V^n \rangle$ before interpolating over the boundary.

The projection is based on the surface normal vector used to reflect over the boundary. Given a normal vector, one can find two orthonormal vectors in 3D space that represent the two tangential directions. To keep things uniform, in 2D, the first tangential direction is always to the right of the normal vector in the x-y plane. One can see the projection of a velocity vector (purple) into tangential and normal components around a circle in Figure (5). The lighter arrows represent the positive normal and tangential directions at each point. The three unit normal and tangential direction vectors create a 3×3 projection matrix. The velocity vector can be multiplied by this calculated projection matrix to give the tangential and normal components, and by its inverse to return to cartesian components.

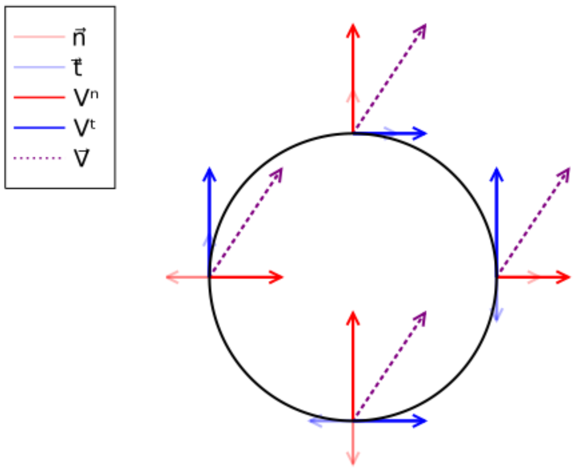


Figure 5: Projection of a given 2D velocity, $\vec{V}$ into tangential and normal components, at different locations around a circle, detailing the projection method used in IBM

Now, at each reflected node, the trilinear interpolation must be done for all three velocity components, $\langle u, v, w \rangle$. Then the velocity is projected into normal and tangential components $\langle V^{t1}, V^{t2}, V^n \rangle$. Using tangential and normal boundary conditions, the projected velocity at $\boldsymbol{x}_R$ can be interpolated to $\boldsymbol{x}_I$. Since the model values are still given in cartesian components, the velocity at $\boldsymbol{x}_I$ must be projected back. It was discussed in Section 2.2.3, that the reflection process is done independently for each velocity component. Therefore, once we have projected back at the immersed node x_I into cartesian velocity components, we are only going to "mask" the component we are working on at the time. The projection process means we interpolate 9 times for a given cell, but the added benefit to researchers mitigates added computational costs. Fortunately, as tracers are not vectors, we only project in the case of the three momentum equations. Projection is not done in most IBM implementations, and could cause differences in our results in comparison, as will be discussed in Section 4.1.

2.2.5 Boundary conditions

Given a quantity q, we can assume that we have the reflected value q_R, the distance between the points $L = |\boldsymbol{x}_I - \boldsymbol{x}_b|$, and the unit normal vector $\boldsymbol{n}$. Then for a Dirichlet boundary condition of $q(\boldsymbol{x}_b) = q_b$, we can determine q_I from linear interpolation along $\boldsymbol{n}$, giving

$$q_I = 2q_b - q_R.$$

For a Neumann boundary condition at the point $\boldsymbol{x}_b$, we would have $\frac{\partial q}{\partial n}\Big|_b = a_b$, such that if we discretize the left hand side with a central difference along $\boldsymbol{n}$, we find

$$\frac{q_R - q_I}{2L} = a_b$$

Therefore, the value at x_I, can be given by

$$q_I = q_R - 2\,a_b\,L.$$

2.3 Internal treatment of the immersed body

This process only holds at the boundary where you have boundary conditions to enforce, so some thought must be given to the treatment of the nodes within the solid that are not at the boundary. For this internal treatment, there are several possibilities, as described by Fadlun.[7] One could apply forcing within the body, leave the quantities within the body free to develop without alteration, or enforce a reflected velocity distribution that continues the linear profile of the fluid. In all of them it was found that the external flow remains unchanged as long as the boundary remains the same. Therefore, Fadlun suggests that the easiest treatment be used. For our implementation, we have forced the velocity and tracer values to zero, allowing the pressure to adjust accordingly. This will result in some unphysical flow within the body, but will not impact the simulated fluid.

3 Results

To demonstrate the accuracy and convergence of the IBM implementation we use several case studies involving flow around a circular cylinder.

3.1 Steady State Flow

First we will look at steady state flow around a cylinder, as it has been extensively studied, with many experimental and numerical results, including those with IBM implementation. We will be comparing our results to those collected by Fornberg and those from the Nasr-Azadani and Meiburg IBM implementation.[8,20] The specifications of the flow are given by very few parameters. The flow dynamics depend on the Reynolds number, which we will define as $Re_D = U_\infty D/\nu$, with U_∞ the free stream velocity, D the diameter of the cylinder, and ν the kinematic viscosity of the fluid. We will focus on steady-state flow, with $Re_D = 40$. The simulation was determined to reach steady state when the norms of the pressure and velocity changes in time were smaller than 10^{-5}, as described in Figure (7). Figure (6) describes the specifications of the problem. Periodic boundary conditions were used along the left and right sides of the the domain, and we imposed a vertical inflow and outflow velocity of $v = U_\infty = 1$. Our simulations were run with a cylinder of diameter $D = 2$. To minimize the impact of the outer boundary on the flow near the cylinder, our simulations were carried out on domains of $L_x \times L_y = 10D \times 15D$. The cylinder was centered at $(x_c, y_c) = (L_x/3, L_y/2)$, for the same reason.[20] Table 1 gives information about the grid resolution for the studies used for validation and convergence.

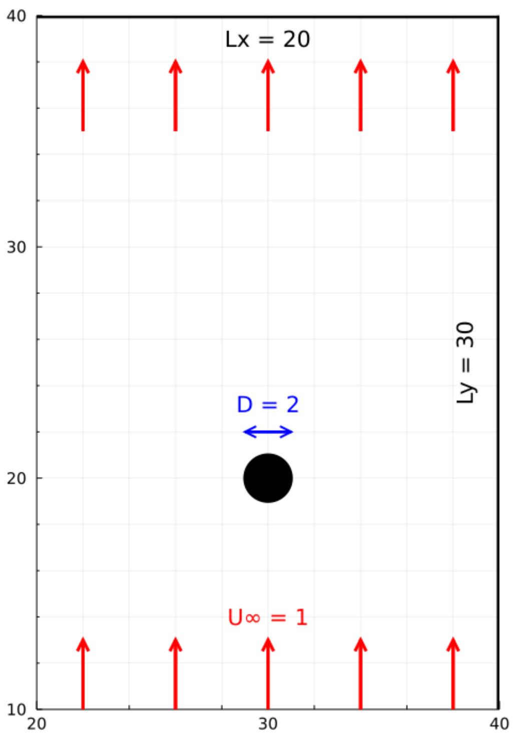


Figure 6: Diagram of domain and probelm specifications used for steady state case study

$\boldsymbol{\Delta x = \Delta y}$	0.04	0.08
$\boldsymbol{nx}$	500	250
$\boldsymbol{ny}$	750	375
$\boldsymbol{\Delta t}$	0.0057	0.0228

Table 1: Resolution for steady state studies

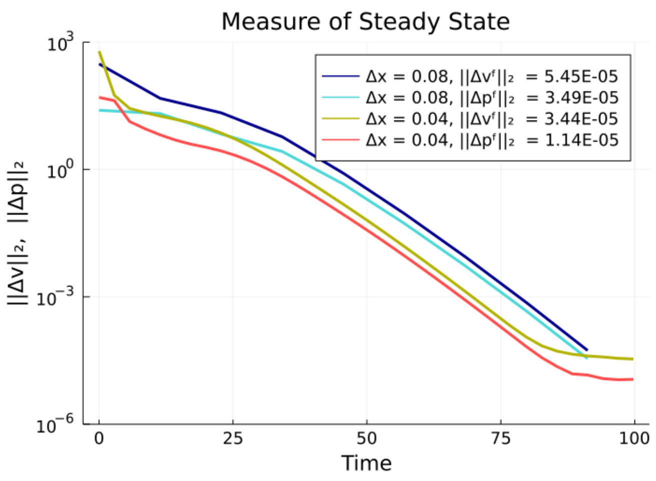


Figure 7: "Steady State" numericallly determined by when the model variables, velocity and pressure, vary less than $O(10^{-5})$ in time

3.1.1 Fluid characteristics around the cylinder

One can see the depiction of flow from the simulation though the streamlines in Figure (8). The arrows show the direction of flow and the separation of the boundary layer around the cylinder, creating the characteristic, symmetric twin vortices within the wake. The red line through the center corresponds to the recirculation length or wake length, L_w, extending from the back stagnation point on the cylinder to the end of the wake. The end of the wake is determined by the point where the velocity along the vertical center of the domain crosses from negative (recirculation) to positive values (uniform flow). The values of L_w obtained by the current study, scaled by the diameter, show some reasonable agreement with results from other authors, as indicated in Table(2). While these lengths are varied, our results are within 13% and 10% away from the mean. This could be due to the other boundary interaction dampening the wake length. It is encouraging that with grid refinement, the wake length begins to converge.

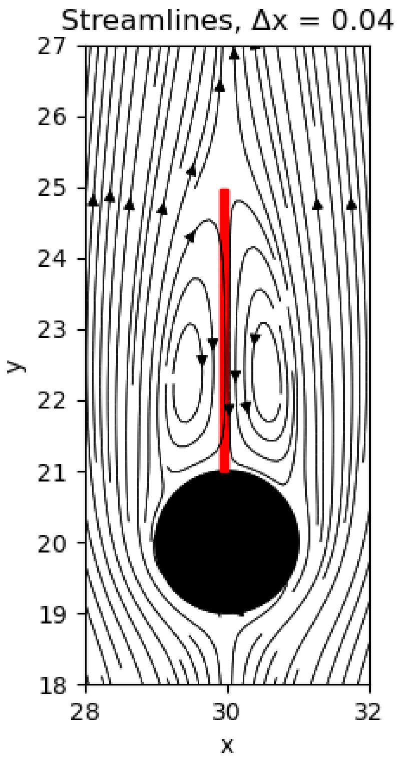


Figure 8: Streamlines around a cylinder for steady state flow with $\Delta x = 0.04$, with marked wake length, L_w

Study	$\boldsymbol{L_w/D}$
Current Study, $\Delta x = 0.04$	2.02
Current Study, $\Delta x = 0.08$	2.01
Ta	2.13
Fornberg	2.24
Ye et al.	2.27
Nasr-Azadani et al.	2.26
Dennis and Chang	2.35
Mean	2.25
Std Dev	0.08

Table 2: Comparison of wake length from steady state simulations to experimental and numerical results[8, 20]

Study	$\boldsymbol{p_{stag}}$	**Error**
$\Delta x = 0.04$	0.585	3.0%
$\Delta x = 0.08$	0.59	3.3%
Mean[8, 20]	0.571	

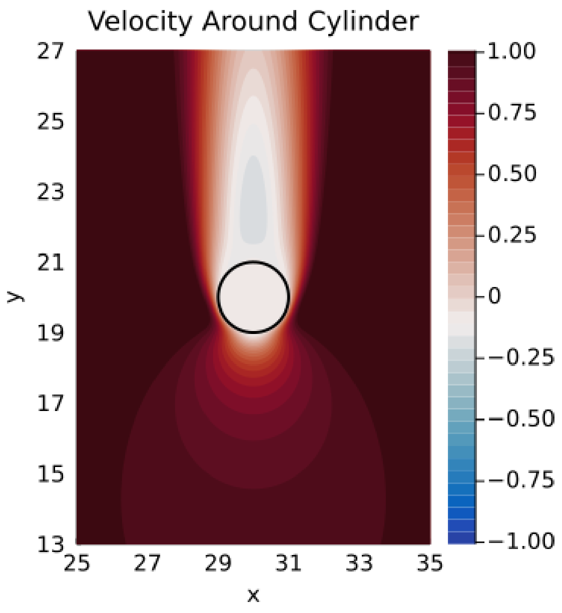
Table 3: Comparison of stagnation pressure to the mean stagnation pressure from the same studies as above

The contour plots in Figure (9) depict the velocity and pressure distribution near the cylinder after the simulation reached a steady-state solution. Within the velocity contours of Figure (9a) one can clearly see the wake formed once the solution reached steady-state flow, as depicted in the streamlines. Figure (9b) also demonstrates that the pressure contours are normal to the cylinder surface, as is prescribed by the analytical steady-state problem. One can see nonphysical pressure within the cylinder, but as long as our boundary is not moving, this should have little impact on the flow, as this noise will remain in the solid body. The larger pressure values within the cylinder, come from the large divergences created over the boundary via the immersed boundary method. The pressure correction step counteracts these divergences, creating the unphysical pressure you see here. To check the pressure in the fluid, we can look at the commonly recorded, front stagnation pressure. This is the pressure at the point where the velocity is zero before coming in contact with the cylinder. These values for our studies are given in Table(3). One can see that we are within 3% of the mean value from several studies, indicating that our results for pressure are in good agreement despite the noise within the immersed solid.

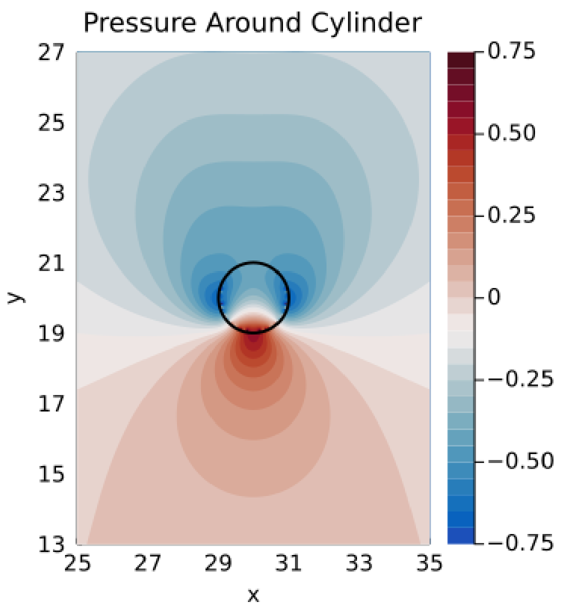
3.1.2 Fluid characteristics on the surface of the cylinder

We can also consider the results specifically on the surface of the cylinder where the boundary truly must be enforced. Instead of (x, y) we will look at positions by the angle θ, as measured counter-clockwise from the right side of the cylinder. This positioning is given in Figure (10).

Figure 9: Steady-state flow past a cylinder, $\Delta x = 0.04$



(a) Velocity contours showing wake formation behind cylinder and small magnitude inside of cylinder



(b) Pressure contours normal to the boundary, but with unphysical values inside the cylinder

To evaluate the accuracy of our model on the surface, we calculated the pressure coefficient,

$$C_p = \frac{p - p_\infty}{1/2 \rho U_\infty^2}.$$

Here, p is the pressure on the surface, p_∞ is the free stream pressure, and ρ is the density of the fluid.[20] The pressure coefficient results for both mesh sizes are compared to data obtained by Fornberg in Figure (11). Due to the symmetric nature of steady state flow, we have only plotted the coefficients of the right side of the surface. The increased grid resolution shows better accuracy and greater adherance to the smooth profile we would expect.

We can also look at the friction coefficient, given by

$$C_f = \frac{\tau_w}{1/2 \rho U_\infty^2} = 2\nu \frac{\partial V^t}{\partial n},$$

where τ_w is the stress at the "wall," or the surface of the cylinder in this case. This is an important quantity for ocean modeling, since it it comparable to the stress quantity we will need to be able model flow near the boundary. The friction coefficient results for both mesh sizes are compared to data obtained by Fornberg in Figure (12). One can see the convergence with increased grid resolution. Comparable results by Nazr-Azadani and Meiburg[20] with their IBM implementation were not nearly as accurate for this measure, as depicted in gray in the figure. This may be a positive impact of the projection step of this implementation, as it is not used in the other work.

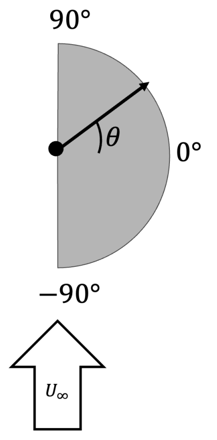


Figure 10: Angle position around surface of cylinder

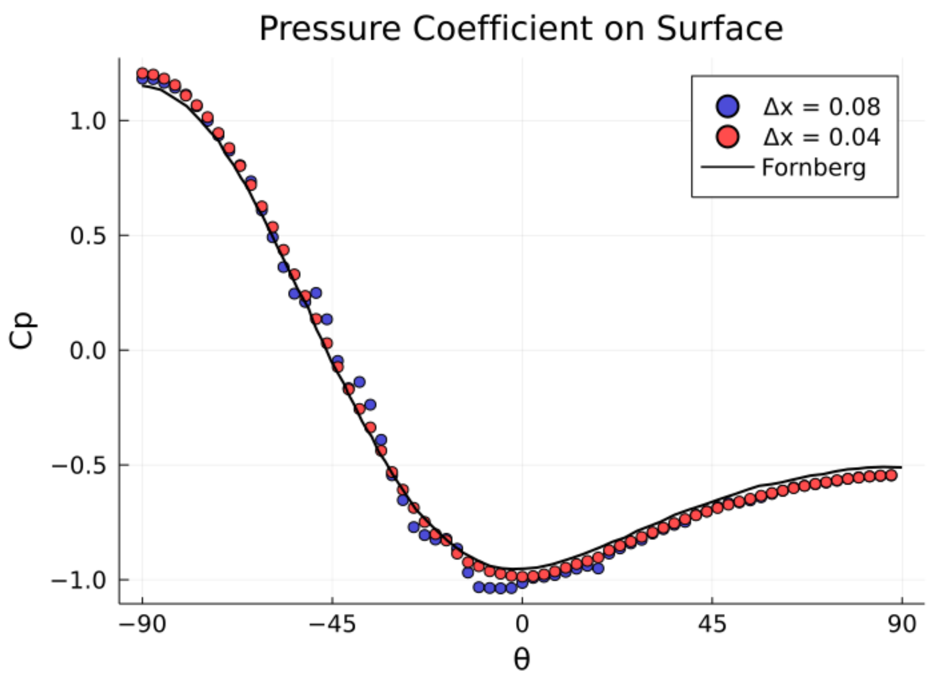


Figure 11: Pressure coefficient compared to Fornberg results shows convergence with increased resolution[8]

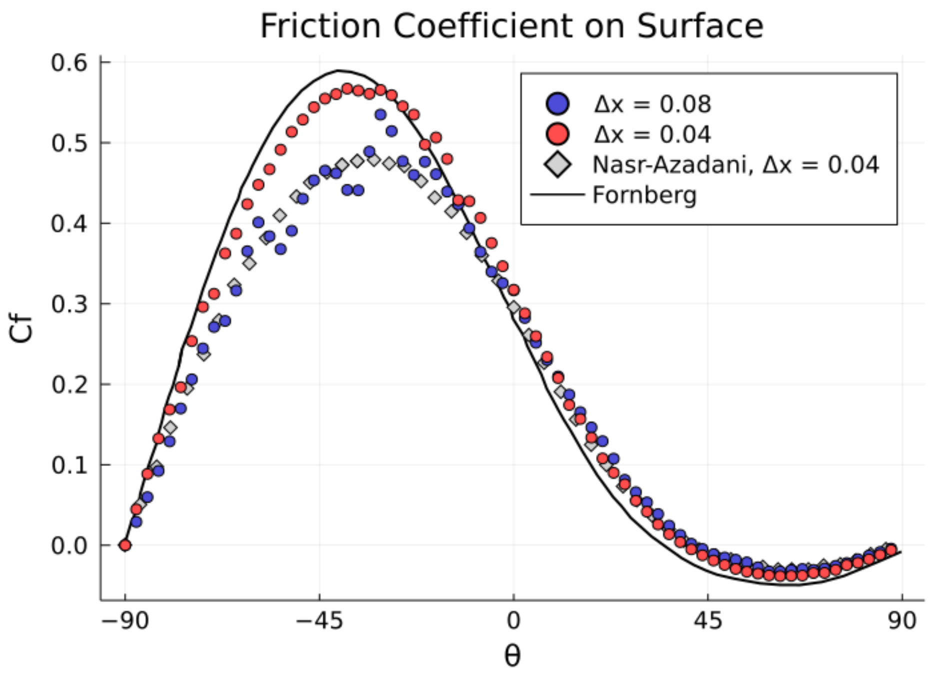


Figure 12: Friction coefficient compared to Fornberg results[8] shows better convergence than Nasr-Azadani and Meiburg results at similar resolution[20]

Finally, we can consider the velocity tangential and normal to the surface of the cylinder, which through the projection step, is where the model enforces zero Dirchlet boundary conditions. Therefore, any nonzero values, are erroneous. These distributions over the whole cylinder surface are given in Figures (13) and (14) for both mesh sizes. The norms of the errors in the normal and tangential velocity at the more refined $\Delta x = 0.04$, are $||V^n||_2 = 0.085$ and $||V^t||_2 = 0.062$. These values are only 25% and 18% of the errors in the coarse resolution, showing good convergence. About 40% of the velocity error in this case study can be attributed to the pressure correction step taken after the immersed boundary "masking." The rest of the error could be impacted by the process of projecting into tangential and normal components, but only enforcing one of three derived cartesian components at the immersed nodes. While the data is noisy, we would not necessarily expect it to be smooth, merely symmetric. Due to the curvature in the cylinder compared to the grid spacing, the surface normal and subsequent interpolation could vary greatly between staggered components of velocity. This could cause the calculated values during analysis to suffer a degree of variability.

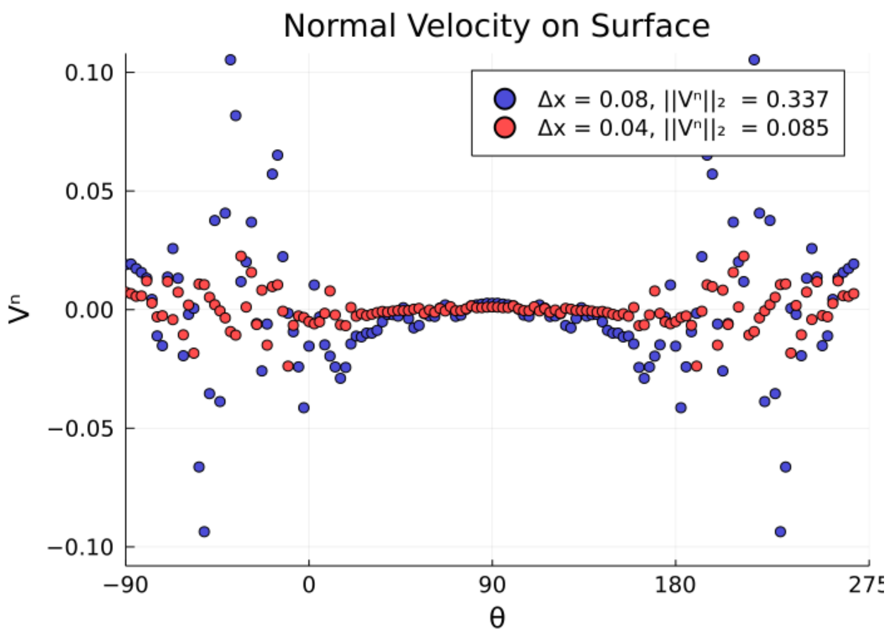


Figure 13: Normal Velocity on steady state flow for both tested resolutions, with error given by non zero values

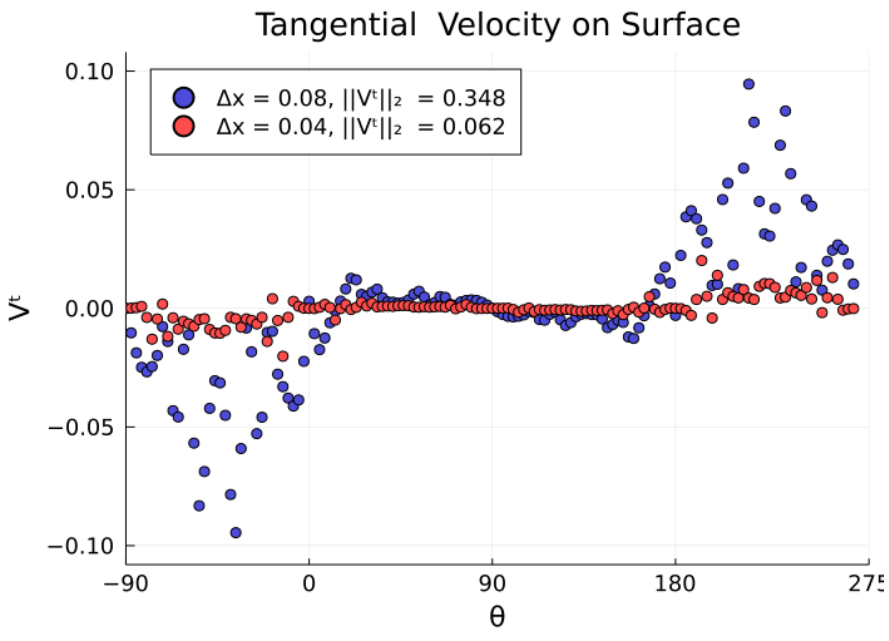


Figure 14: Tangential Velocity on steady state flow for both tested resolutions, with error given by non zero values

3.2 Inclusion of Tracer in Cylinder Flow

With the above results for the momentum equations enforced with Dirichlet boundary conditions, we will now look at implementing Neumann boundary conditions and the introduction of a tracer. The velocity boundary conditions will remain the same, but the model will also include an arbitrary tracer, C, with a diffusivity given by $\kappa = 2/Re$.

For the first case, we will continue with the same set up as in the previous studies, using a Reynolds number of $Re = 40$, with the original grid size of 500×750 points. On the cylinder boundary, the tracer is given a Neumann boundary condition of 0, such that it is insulated from the tracer in the fluid. Theoretically, there should be no change in the amount of concentration in fluid over time. Then the volume integrated concentration should remain steady. The simulation was run for 20 seconds, and the percent concentration leakage, as compared to the initial values was calculated. This is shown in Figure (15). After 20 seconds there is only a 1.4% increase in the amount of concentration in the fluid. This indicates that, even though the velocity shows $O(10^{-1})$ errors and "leakage," this is not impacting the tracers at the same magnitude. This could mean that the level of errors in the velocity will not greatly impact larger applications of this IBM code.

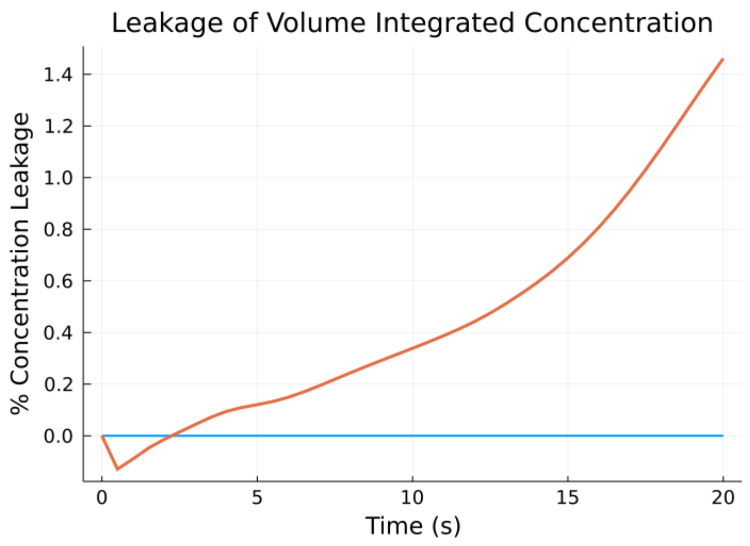


Figure 15: % Leakage of tracer concentration over time for cylinder boundary condition, $\frac{\partial C}{\partial n} = 0$. Any leakage is unphysical

For the second case, we used an unsteady flow simulation with a Reynold's number of $Re = 100$ and a coarser grid of 350×350 points. On the cylinder boundary the tracer is given a Neumann boundary condition of 1, such that it is leaking tracer into the fluid at a steady rate. The fluid starts with a concentration of zero. When the normal gradient is along the cylinder boundary, we can take the volume integrated tracer concentration and calculate the rate of leakage into the fluid. The theoretical rate of leakage into the fluid is given by the tracer's diffusivity, κ, the surface area of the cylinder, and the Neumann boundary condition,

$$\frac{dC}{dt} = \kappa \frac{\partial C}{\partial n}(\pi D) = 2\kappa\pi.$$

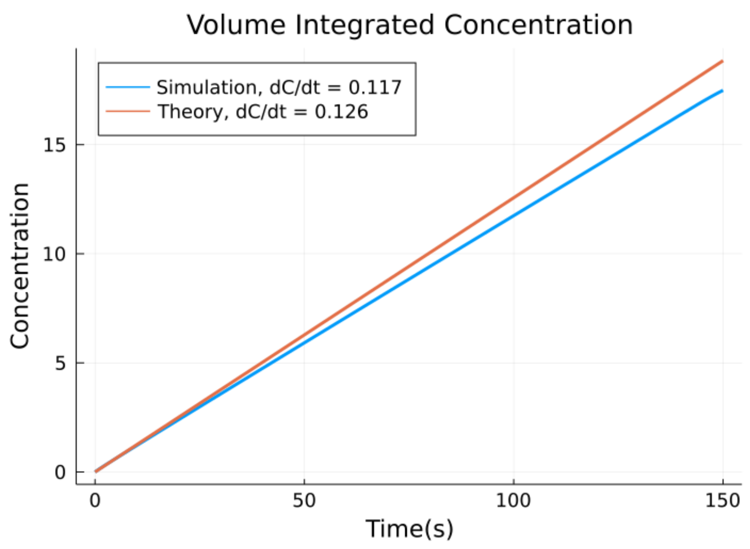


Figure 16: Tracer concentration leakage for cylinder boundary condition, $\frac{\partial C}{\partial n} = 1$ compared to theory. Slopes or rates given for comparison

By calculating the volume integrated concentration of the tracer over time, we can determine if our results reflect this theoretical rate. This comparison is given in Figure (16). One can see that the difference between simulation and theory is only 7%, and the concentration is steadily increasing, despite the larger Reynolds number. This also indicates that a more developed model will not see large impacts from velocity errors advecting tracer through the cylinder.

4 Conclusions and Future Directions

4.1 Summary

The implementation of immersed boundaries shows promising results, with errors that can be scaled as per the user's requirements through grid refinement. For the test case of uniform flow around a circular cylinder, our results show qualitatively good agreement with the expected flow. The pressure contours are normal to the surface, and we see the symmetry in the cylinder wake as is desired. There is some variation within the measurements for wake length, but this could be due to exterior boundary interaction rather than the immersed boundary. On the surface of the cylinder we see normal and tangential velocity measurements with errors on the order of 10%. This is larger than one would like, but is most likely an artifact of the projection step and the pressure correction, and can be mitigated with increased resolution. The pressure and friction coefficients show very good agreement with the non-IBM, numerical results by Fornberg,[8] and indicate convergence of the model with refinement of the grid size. With the inclusion of tracers, we find that the velocity errors are not causing a similar magnitude error in the tracers. The errors in the tracer budget are within reasonable limits for geophysical applications. Ultimately, we have taken the first steps towards addressing the many unknowns involving flow-topography interactions in the ocean in the submesoscale.

4.2 Future Directions

With these results we can confidently move forward to applying IBM to topography problems, such as the breaking of an internal wave approaching a shoaling coast. This topic has been considered with IBM before by Winters in 2015[29] and would allow us to investigate geophysical applications with comparable results. Winters used a no-slip immersed bottom boundary slope, which is well within the capabilities of our model. Depictions of the velocity and kinetic energy dissipation rate from the Winters simulation are given in Figure (17). One can see that the slope has triggered "ejection events" where the near-boundary fluid moves into the interior, indicated by the yellow arrows. With our immersed boundary method, we can simulate a bottom slope, such as this one, and determine the impacts of topography on mixing in the ocean.

A large part of our future work will be determining the impact of the projection method and the different avenues of its implementation we could take. Currently the step of projecting the velocity into tangential and normal components improves the results of the friction coefficient and the tangential velocity compared to other IBM methods,[20] but it could be doing worse at mass conservation with respect to the momentum equations. By only enforcing one component of the three, there could be variation near the boundary, leading to the magnitudes of error we are seeing.

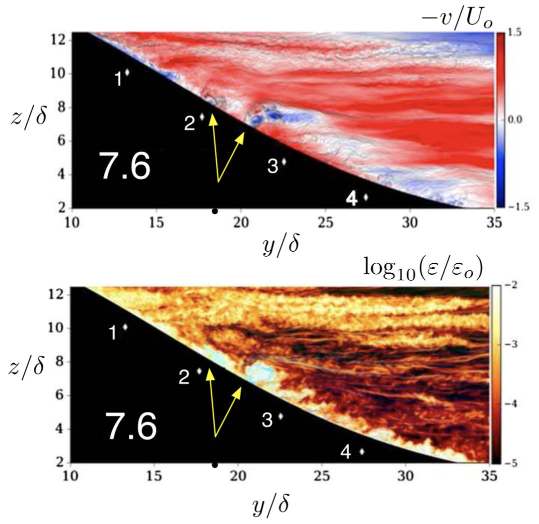


Figure 17: Flow topography interactions along a slope, adapted from Winters[29]

Among our efforts to include the final implementation into the next Oceananigans release, we will also continue to improve the efficiency and usability of the model. These are some of the main tenants of Oceananigans,[21] and are necessary updates for the IBM implementation to becomes more widely used and appreciated in LES modeling. We must determine the best way for users to specify the necessary information, such as the distance function and individual boundary conditions. Currently the code is also not running at the necessary speeds. One way to improve speeds would be to make better use of Julia's multiple dispatch concept. In Julia, functions can have multiple definitions as long as each definition has a different combination of arguments, allowing the programmer to avoid typical nested "if" statements.[3] This can greatly speed up the code, if used smartly. We can also take advantage of Julia's ability to handle message passing interface (MPI) code. Since our topography remains constant in time and the geometry could be calculated from the start and stored locally on each processor, IBM can be paralellized, for improved speeds.[20] With some of these specifications and improvements the method can be confidently released to the public for general use.

4.3 Conclusion

The goal of this project was to determine and implement the best method of including topography into Oceananigans' LES code. We wanted it to handle both Dirichlet and Neumann boundary conditions for the momentum equations as well as an arbitrary number of tracers. The described implementation has fulfilled these goals. Ultimately the product will be a tool our team and other researchers can use in future investigations simulating the separation of ocean boundary layers over rough terrain. By adding this ability to the up-and-coming Oceananigans, we have opened up new avenues of research which will pave the way to even more accurate models and simulations that include the impact of topography on ocean dynamics.

A Algorithm

Algorithm 1 IBM Algorithm for kth stage of RK3 method

```
**INPUT:** $\boldsymbol{U}^{k-1}, Q^{k-1}$

**Update** Velocities and tracers with RK step: $\boldsymbol{U^k}, Q^k$, immersed distance function: $\phi(\boldsymbol{x})$

**for** Velocity fields **do**
    **if** Solid Node **then**
        **if** Immersed Node **then**
            $\boldsymbol{x_I} = \boldsymbol{x}$
            $\boldsymbol{n} = \nabla\phi(\boldsymbol{x}_I)$
            $L = |\phi(\boldsymbol{x}_I)|$
            $\boldsymbol{x_b} = \boldsymbol{x_I} + L\boldsymbol{n}$
            $\boldsymbol{x}_R = \boldsymbol{x_I} + 2L\boldsymbol{n}$
            **Interpolate** $\boldsymbol{U} = (u, v, w)$ at Reflected Point: $\boldsymbol{U}_R$
            **Project** $\boldsymbol{U_R}$ at Reflected Point: $\hat{\boldsymbol{U}}_R$
            **Interpolate** to Immersed Node: $\hat{\boldsymbol{U}}_I$
            **Project** $\hat{\boldsymbol{U}}_I$ back at Immersed Node: $\boldsymbol{U_I}$
            **Isolate** the desired velocity component: $U_I = \boldsymbol{U}_I[idx]$
            **Update** velocity $U^k(\boldsymbol{x_I}) = U_I$
        **else**
            **Update** velocity $U^k(\boldsymbol{x}) = 0$
        **end if**
    **else**
        Fluid node, so do nothing.
    **end if**
**end for**

**for** Tracer fields **do**
    **if** Solid Node **then**
        **if** Immersed Node **then**
            $\boldsymbol{x_I} = \boldsymbol{x}$
            $\boldsymbol{n} = \nabla\phi(\boldsymbol{x}_I)$
            $L = |\phi(\boldsymbol{x}_I)|$
            $\boldsymbol{x_b} = \boldsymbol{x_I} + L\boldsymbol{n}$
            $\boldsymbol{x}_R = \boldsymbol{x_I} + 2L\boldsymbol{n}$
            **Interpolate** tracer, $Q$, at Reflected Point: $Q_R$
            **Interpolate** to Immersed Node: $Q_I$
            **Update** tracere $Q^k(\boldsymbol{x_I}) = Q_I$
        **else**
            **Update** tracers $Q^k(\boldsymbol{x}) = 0$
        **end if**
    **else**
        Fluid node, so do nothing.
    **end if**
**end for**
**Correct** $\boldsymbol{U}^k$ with pressure
```

B Github Code

Make sure you are on the `vw/arbitrary_immersedboundary` branch of the repository. The README.md file in the src directory will direct you to relevant files.

`https://github.com/whitleyv/Oceananigans.jl/tree/vw/arbitrary_immersedboundary/src`

C Data and Results

Simulation data can be found in the google drive as `.jld2` files. Jupyter, Julia notebooks from analysis are found as `.ipynb` files. The papers by Fornberg and Nasr-Azadani and Meiburg, which contain the data used in comparison to our results, are also included here. There is also a README file here to explain each of the files.

`https://drive.google.com/drive/folders/1B6JDxxg7gUy1wFDT7HVXI5K7GuFqztTS?usp=sharing`

References

[1] Arthur, Robert S., et al. "Evaluating Implementations of the Immersed Boundary Method in the Weather Research and Forecasting Model." *Monthly Weather Review*, vol. 148, no. 5, 2020, pp. 2087–2109., doi:10.1175/mwr-d-19-0219.1

[2] Balaras, Elias. "Modeling complex boundaries using an external force field on fixed Cartesian grids in large-eddy simulations." *Journal of Computers and Fluids*, vol. 33, 2004, pp.375-404.

[3] Bezanson, et al. "Array operators using multiple dispatch: a design methodology for array implementations in dynamic languages." *ARRAY'14 Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, 2014, pp 56–61., doi: 10.1145/2627373.2627383.

[4] Briscolini, M. and P. Santangelo. "Development of the mask method for incompressible unsteady flows." *Journal of Computational Physics*, vol. 84. 1989, pp 57-75., dot:0021-9991/89.

[5] E. Griffith, Boyce, and Xiaoyu Luo. "Hybrid Finite Difference/Finite Element Immersed Boundary Method." *International Journal for Numerical Methods in Biomedical Engineering*, vol. 33, no. 12, 2017, doi:10.1002/cnm.2888.

[6] Eymard, Robert, and Thierry Galllouã. "Finite Volume Method." *Scholarpedia*, vol. 5, no. 6, 2010, p. 9835., doi:10.4249/scholarpedia.9835.

[7] Fadlun, E.A., et al. "Combined Immersed-Boundary Finite-Difference Methods for Three-Dimensional Complex Flow Simulations." *Journal of Computational Physics*, vol. 161, no. 1, 2000, pp. 35–60., doi:10.1006/jcph.2000.6484.

[8] Fornberg B. "A numerical study of steady viscous flow past a circular cylinder." *Journal of Fluid Mechanics*, vol 98, no. 4, 1980, pp. 819–55.

[9] Guermond, J.l., et al. "An Overview of Projection Methods for Incompressible Flows." *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 44-47, 2006, pp. 6011–6045., doi:10.1016/j.cma.2005.10.010.

[10] Gula, Jonathan, et al. "Topographic Generation of Submesoscale Centrifugal Instability and Energy Dissipation." *Nature Communications*, vol. 7, no. 1, 2016, doi:10.1038/ncomms12811.

[11] Hamlington, Peter, et al. "Langmuir-Submesoscale Interactions: Descriptive Analysis of Multiscale Frontal Spindown Simulations." *Journal of Physical Oceanography*, vol. 44, 2014, pp. 2249-2272, doi:10.1175/JPO-D-13-0139.1.

[12] He, Yinnian, and Jian Li. “Numerical Implementation of the Crank-Nicolson/Adams-Bashforth Scheme for the Time-Dependent Navier-Stokes Equations.” *International Journal for Numerical Methods in Fluids*, 2009, doi:10.1002/fld.2035.

[13] Iaccarino, Gianluca, and Roberto Verzicco. “Immersed Boundary Technique for Turbulent Flow Simulations.” *Applied Mechanics Reviews*, vol. 56, no. 3, 2003, pp. 331–347., doi:10.1115/1.1563627

[14] Kallemov, Bakytzhan, et al. “An Immersed Boundary Method for Rigid Bodies.” *Communications in Applied Mathematics and Computational Science*, vol. 11, no. 1, 2016, pp. 79–141., doi:10.2140/camcos.2016.11.79.

[15] Le, Hung, and Parviz Moin. “An Improvement of Fractional Step Methods for the Incompressible Navier-Stokes Equations.” *Journal of Computational Physics*, vol. 92, 1991, pp. 369–379., doi:0021-9991/91.

[16] Marshall, John, et al. “A Finite-Volume, Incompressible Navier Stokes Model for Studies of the Ocean on Parallel Computers.” *Journal of Geophysical Research: Oceans*, vol.102, no. C3, 1997, pp. 5753–5766., doi:10.1029/96jc02775.

[17] Mcwilliams, James C. “Submesoscale Currents in the Ocean.” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 472, no. 2189, 2016, p. 20160117., doi:10.1098/rspa.2016.0117.

[18] M.H. Chung. “Cartesian cut cell approach for simulating incompressible flows with rigid bodies of arbitrary shape.” *Journal of Computers and Fluids*, 35(6):607623, 2006. doi:10.1016/j.compfluid.2005.04.005.

[19] Mittal, Rajat, and Gianluca Iaccarino. “Immersed Boundary Methods.” *Annual Review of Fluid Mechanics*, vol. 37, no. 1, 2005, pp. 239–261., doi:10.1146/annurev.fluid.37.061903.175743.

[20] Nasr-Azadani, M.M., and E. Meiburg. “TURBINS: An Immersed Boundary, Navier-Stokes Code for the Simulation of Gravity and Turbidity Currents Interacting with Complex Topographies.” *Journal of Computers and Fluids*, vol. 45, 2011, pp. 14–28., doi:10.1016/j.compfluid.2010.11.023.

[21] Ramadhan, Ali, et al. “Oceananigans.jl.” *CliMA/Oceananigans.jl*, GitHub, 2020, github.com/CliMA/Oceananigans.jl.

[22] Rodriguez, Sal.“LES and DNS Turbulence Modeling.” *Applied Computational Fluid Dynamics and Turbulence Modeling: Practical Tools, Tips and Techniques*, by Sal Rodriguez, Springer, 2019, pp. 197–223.

[23] Schumann, Ulrich, and Roland A Sweet. “Fast Fourier Transforms for Direct Solution of Poisson’s Equation with Staggered Boundary Conditions.” *Journal of Computational Physics*, vol. 75, no. 1, 1988, pp. 123–137., doi:10.1016/0021-9991(88)90102-7.

[24] Souza, Andre Nogueira, et al. “Uncertainty Quantification of Ocean Parameterizations: Application to the K-Profile-Parameterization for Penetrative Convection.” *Journal of Advances in Modeling Earth Systems*, 2020, doi:10.1002/essoar.10502546.1.

[25] Verzicco, Roberto, et al. “Large Eddy Simulation in Complex Geometric Configurations Using Boundary Body Forces.” *AIAA Journal*, vol. 38, 2000, pp. 427–433., doi:10.2514/3.14430.

[26] Wagner, Gregory, et al. “Near-Inertial Waves and Turbulence Driven by the Growth of Swell.” *Journal of Physical Oceanography*, 30 July 2020, doi:10.1002/essoar.10503838.1.

[27] Wenegrat, Jacob O., and Leif N. Thomas. “Centrifugal and Symmetric Instability during Ekman Adjustment of the Bottom Boundary Layer.” *Journal of Physical Oceanography*, vol. 50, no. 6, 2020, pp. 1793–1812., doi:10.1175/jpo-d-20-0027.1.

[28] Wenegrat, Jacob O., et al. “Submesoscale Baroclinic Instability in the Bottom Boundary Layer.” *Journal of Physical Oceanography*, vol. 48, no. 11, 2018, pp. 2571–2592., doi:10.1175/jpo-d-17-0264.1.

[29] Winters, K. B. “Tidally driven mixing and dissipation in the stratified boundary layer above steep submarine topography.” *Geophysical Research Letters*. AGUPublications, 2015.

[30] Ye, T. et al. “An Accurate Cartesian Grid Method for Viscous Incompressible Flows with Complex Immersed Boundaries.” *Journal of Computational Physics*, 156(2):209240, 1999. doi:10.1006/jcph.1999.6356.

[31] Zhiyin, Yang. “Large-Eddy Simulation: Past, Present and the Future.” *Chinese Journal of Aeronautics*, vol. 28, no. 1, 2015, pp. 11–24., doi:10.1016/j.cja.2014.12.007.