January 2015

Exact Implementation of boundary conditions for Immersed Boundary Methods

Karan Bansal
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_theses

Recommended Citation
https://docs.lib.purdue.edu/open_access_theses/1207

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
This is to certify that the thesis/dissertation prepared

By Karan Bansal

Entitled
EXACT IMPLEMENTATION OF BOUNDARY CONDITIONS FOR IMMERSED BOUNDARY METHODS

For the degree of Master of Science in Aeronautics and Astronautics

Is approved by the final examining committee:

Tom I-P. Shih  Chair
Gregory A. Blaisdell  Co-chair
Alina Alexeenko  Co-chair

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University’s “Policy of Integrity in Research” and the use of copyright material.

Approved by Major Professor(s): Tom I-P. Shih

Approved by:  Tom I-P. Shih  July/27/2015

Head of the Departmental Graduate Program  Date
EXACT IMPLEMENTATION OF BOUNDARY CONDITIONS FOR IMMERSED BOUNDARY METHODS

A Thesis
Submitted to the Faculty
of
Purdue University
by
Karan Bansal

In Partial Fulfillment of the
Requirements for the Degree
of
Master of Science in Aeronautics and Astronautics

August 2015
Purdue University
West Lafayette, Indiana
For my family, friends and my brother Pratik who is dearly missed...
ACKNOWLEDGMENTS

First, I must express my sincere gratitude to my advisor, Dr. Tom I-P. Shih who has passionately guided me through the course of this research. I am grateful for his kindness, patience and for the effort he places in motivating students like me to push forward or thinking. I also want to thank my labmates Adwiteey, Chintan, Jason, Zach, Wanjia, Irsha, Kenny, Yongkai and Adeel in the "Computational Heat Transfer and Fluid Dynamics Lab" who were always ready to help, give valuable advice and plan random lunch outings which often ended in random boring, but interesting discussions on human psychology.

I wish to acknowledge Dr. Gregory A. Blaisdell and Dr. Alina Alexeenko for their time and willingness to be in my thesis committee. I would like to take this opportunity to especially thank my dear friend Dr. Chien-Shing Lee who has helped and encouraged me throughout my research. Finally, I am thankful to my family for their immense support and blessings.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>ix</td>
</tr>
<tr>
<td>ABBREVIATIONS</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2. LITERATURE REVIEW</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Finite Difference Methods Using IBM</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Finite Volume Methods Using IBM</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Cut Cell Method</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1.1 Issues with Cut Cell Methods</td>
<td>18</td>
</tr>
<tr>
<td>2.2.2 Embedded Boundary Method</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2.1 Issues with Embedded Boundary Method</td>
<td>21</td>
</tr>
<tr>
<td>CHAPTER 3. RESEARCH OBJECTIVES</td>
<td>24</td>
</tr>
<tr>
<td>CHAPTER 4. NEW METHOD: TRANSFINITE INTERPOLATION</td>
<td>25</td>
</tr>
<tr>
<td>4.1 Transfinite Interpolation on Parallel Straight Lines</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Transfinite Interpolation on Arbitrary Curves</td>
<td>26</td>
</tr>
<tr>
<td>4.3 Implementation</td>
<td>28</td>
</tr>
<tr>
<td>4.3.1 Dirichlet Boundary Condition</td>
<td>29</td>
</tr>
<tr>
<td>4.3.2 Neumann Boundary Condition</td>
<td>31</td>
</tr>
<tr>
<td>CHAPTER 5. GOVERNING EQUATIONS AND NUMERICAL METHOD</td>
<td>33</td>
</tr>
<tr>
<td>5.1 IBM for Potential Flow</td>
<td>33</td>
</tr>
<tr>
<td>5.1.1 Fluid Cells</td>
<td>34</td>
</tr>
<tr>
<td>5.1.2 Solid Cells</td>
<td>36</td>
</tr>
<tr>
<td>5.1.3 Immersed Cells</td>
<td>37</td>
</tr>
<tr>
<td>5.1.3.1 Flux Through Cartesian Cell Faces</td>
<td>37</td>
</tr>
<tr>
<td>5.1.3.2 Flux Through Boundary Face</td>
<td>38</td>
</tr>
<tr>
<td>5.1.3.2.1 IBM1</td>
<td>39</td>
</tr>
<tr>
<td>5.1.3.2.2 IBM2</td>
<td>40</td>
</tr>
<tr>
<td>5.1.3.2.3 PRESENT Method</td>
<td>42</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>12</td>
</tr>
<tr>
<td>6.1</td>
<td>69</td>
</tr>
</tbody>
</table>

2.1 FDM on boundary conforming grid vs FDM on IBM grid

2.2 FVM on boundary conforming grid vs FVM on IBM grid

6.1 Comparison of CPU time for the 3 methods on the 4 grid sizes used
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 An example of boundary conforming grid and an IBM grid</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Figure showing immersed boundary</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Identification of grid points for a 2-D circular cylinder</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Close up region near the immersed boundary showing classification of nodes</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Region near the immersed boundary classifying nodes</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Finite volume IBM grid for 2D circular cylinder</td>
<td>12</td>
</tr>
<tr>
<td>2.6 Finite volume IBM grid for 2D cylinder showing cell centers</td>
<td>13</td>
</tr>
<tr>
<td>2.7 Types of immersed cells encountered</td>
<td>14</td>
</tr>
<tr>
<td>2.8 Cut cells</td>
<td>16</td>
</tr>
<tr>
<td>2.9 Cell reshaping: Cell with its center inside the solid boundary gets merged into neighbouring cell</td>
<td>16</td>
</tr>
<tr>
<td>2.10 6 point stencil used to construct the interpolant to calculate $F_{sw}$</td>
<td>17</td>
</tr>
<tr>
<td>2.11 6 point stencil used to construct the interpolant to calculate $F_f$</td>
<td>18</td>
</tr>
<tr>
<td>2.12 Figure showing the calculation of $\alpha$ values for immersed cell faces</td>
<td>20</td>
</tr>
<tr>
<td>2.13 Figure showing how to calculate fluxes with second order accuracy on the Cartesian faces of immersed cells</td>
<td>22</td>
</tr>
<tr>
<td>2.14 Stencil to interpolate the boundary flux</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Set of $N$ parallel lines</td>
<td>26</td>
</tr>
<tr>
<td>4.2 Mapping of physical domain to computational domain</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Dirichlet boundary condition</td>
<td>29</td>
</tr>
<tr>
<td>4.4 Neumann boundary condition</td>
<td>31</td>
</tr>
<tr>
<td>5.1 Cell types</td>
<td>35</td>
</tr>
<tr>
<td>5.2 A rectangular differential volume element</td>
<td>36</td>
</tr>
<tr>
<td>5.3 Fluxes and their computation on immersed cells</td>
<td>38</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>5.4 Figures showing the original boundary and its approximation as linear elements</td>
<td>39</td>
</tr>
<tr>
<td>5.5 Stencil to interpolate the boundary flux using IBM1</td>
<td>40</td>
</tr>
<tr>
<td>5.6 Figure showing bilinear interpolation on point P and the stencil used</td>
<td>41</td>
</tr>
<tr>
<td>5.7 Stencil to interpolate the boundary flux using IBM2</td>
<td>43</td>
</tr>
<tr>
<td>5.8 Stencil to interpolate the boundary flux using Transfinite interpolation</td>
<td>44</td>
</tr>
<tr>
<td>5.9 Mapping the physical domain to computational domain</td>
<td>45</td>
</tr>
<tr>
<td>6.1 Figure showing the Computational domain for the flow over a 2D cylinder</td>
<td>49</td>
</tr>
<tr>
<td>6.2 An example of the grid used for flow simulations</td>
<td>51</td>
</tr>
<tr>
<td>6.3 Grids used for grid sensitivity studies (Magnified near immersed boundary)</td>
<td>52</td>
</tr>
<tr>
<td>6.4 Grid Sensitivity Studies for IBM1</td>
<td>53</td>
</tr>
<tr>
<td>6.5 Grid Sensitivity Studies for the IBM2</td>
<td>55</td>
</tr>
<tr>
<td>6.6 Grid Sensitivity Studies for the present method</td>
<td>56</td>
</tr>
<tr>
<td>6.7 Solution variation on 2-D cylinder with variation in $\beta$ for a 61x61 grid</td>
<td>59</td>
</tr>
<tr>
<td>6.8 Solution variation on 2-D cylinder with variation in $\beta$ for a 121x121 grid</td>
<td>60</td>
</tr>
<tr>
<td>6.9 Comparison of the 3 numerical methods on 31 x 31 grid</td>
<td>62</td>
</tr>
<tr>
<td>6.10 Comparison of the 3 numerical methods on 61 x 61 grid</td>
<td>63</td>
</tr>
<tr>
<td>6.11 Comparison of the 3 numerical methods on 121 x 121 grid</td>
<td>64</td>
</tr>
<tr>
<td>6.12 Comparison of the 3 numerical methods on 241 x 241 grid</td>
<td>65</td>
</tr>
<tr>
<td>6.13 Relative Error for $C_p$ on different grid sizes</td>
<td>67</td>
</tr>
<tr>
<td>6.14 Absolute Error for $C_p$ on different grid sizes</td>
<td>68</td>
</tr>
<tr>
<td>A.1 Figure showing concept of ray casting algorithm</td>
<td>72</td>
</tr>
</tbody>
</table>
SYMBOLS

\( \phi \)  A generic flow variable
\( \psi \)  Stream-function
\( x, y \)  Cartesian coordinate system
\( \xi, \eta \)  Coordinate system in computational domain
\( u \)  Velocity in x direction
\( v \)  Velocity in y direction
\( p \)  Pressure
\( C_p \)  Pressure Coefficient
\( \theta \)  Angle
\( dx, dy \)  Dimensions of a 2-D cell
\( m \)  meter
\( s \)  second
ABBREVIATIONS

CFD  Computational Fluid Dynamics
IBM  Immersed Boundary Method
IB   Immersed Boundary
Fig.  Figure
Eqn. Equation
i.e.  that is
et al and others
FDM  Finite Difference Methods
FVM  Finite Volume Methods
2-D  Two Dimensional
3-D  Three Dimensional
BCs  Boundary Conditions
ABSTRACT

Bansal, Karan MSAA, Purdue University, August 2015. Exact Implementation of Boundary Conditions for Immersed Boundary Methods. Major Professor: Tom I-P. Shih.

Most CFD flow solvers obtain solution on boundary-conforming grids. Generating a boundary-conforming grid is, in general, a tedious and time consuming task. To simplify the grid generation process, a technique called Immersed Boundary Method was developed which can be applied to grids that are not boundary-conforming. However, implementing boundary conditions is not straight forward. To address this issue, several Immersed Boundary Methods have been developed over the years. All of these methods were found to satisfy the boundary conditions only on selected points on the boundary but not on the entire boundary. In this thesis, a new method is developed that satisfies the boundary conditions on the entire boundary. This method is demonstrated by applying it to solve potential flow past a circular cylinder. Results from the new method and the existing methods are compared and it is observed that the new method gives more accurate solutions on identical grids.
CHAPTER 1. INTRODUCTION

Grid generation and Flow solver are two major parts of most CFD methods. In grid generation, the physical domain used for flow simulation is discretized into a finite number of nodes and cells (see Figure 1.1). The placement of these nodes and cells, in general, depends upon the geometry and the physics of the problem to be solved. In the flow solver part, the governing equations are discretized and solved on the nodes/cells to compute the numerical solution.

Most conventional flow solvers work with a boundary-conforming grid (see Figure 1.1(a)). Generating a boundary-conforming grid, in general, is a tedious and a time consuming task. A lot of effort has been made in the past few decades to speed up and automate the grid generation process. Unstructured grids, immersed boundary methods, cut cell methods and other similar methods have been developed to reduce the “human” effort needed in grid generation which effectively speeds up the grid generation process.

Immersed Boundary Method (IBM) is a technique that uses a grid that need not be boundary-fitted so that the grid generation process is simple. To illustrate, a Cartesian grid based IBM, as the name suggests, uses a Cartesian grid for the flow domain and then the body over which the flow needs to be simulated is placed appropriately over the Cartesian Grid (see Figure 1.1(b)). This body is then located on the Cartesian grid and a suitable technique is used to enforce the boundary conditions on the body. The advantage of this method is that it requires little, to no human effort in order to generate a grid for flow simulation. Additionally, Cartesian grids allow the use of line iterative techniques to speed up the solution process.

Clearly, IBMs possess attractive qualities, however, there are many issues which need to be addressed when implementing IBMs. Some of the concepts, assumptions and issues are as follows:
Figure 1.1.: An example of boundary conforming grid and an IBM grid

- Identifying grid points and cells inside and outside the boundary surface.
- Having poor grid quality next to the boundary.
- Satisfying boundary conditions on the boundary surface.
- Requiring size of cells that contain the boundary surface to be small enough so that the faces of that cell only intersect the boundary once.
- Having enough cells to capture the geometry with sufficient accuracy. If the boundary curvature in a cell is too high to approximate it as a linear element, the mesh needs to be refined further.

Most of the research to improve IBMs has been on developing methods for accurately implementing the boundary conditions. As it will be made clear in the subsequent chapters, existing IBMs satisfy the boundary condition on “specific points” on the boundary. This does not ensure that boundary conditions are satisfied on the entire boundary surface. Some methods use piecewise linear approximation of the boundary and intend to satisfy the boundary condition on these linear elements. In
fact, in this study, we show that the boundary conditions are not even satisfied on these linear elements.

Thus, the objective of this study is to develop an IBM that will satisfy the boundary conditions on the entire boundary. This method mimics the prismatic grids in satisfying the boundary conditions, hence the naming of this method.

To verify the new method, a potential flow solver is developed and results obtained for flow over a 2-D circular cylinder are compared with the analytical solutions.

The remainder of this thesis is arranged as follows: Chapter 2 provides a review of the relevant literature and highlights issues that have not been addressed in the literature. Chapter 3 provides the research objective and the approach taken to achieve the objective. Chapter 4, presents the new IBM developed in this study to satisfy different types of boundary conditions. In Chapter 5, the new IBM is applied to potential flow equations. In Chapter 6, the algebraic equations developed in Chapter 5 are applied to a flow over a 2-D circular cylinder. Chapter 7 provides a summary of the thesis.
CHAPTER 2. LITERATURE REVIEW

Before moving forth to the literature review, some important terminology will be explained,

**Immersed Boundary** The boundary of the body which is placed or immersed inside the grid (see Figure 2.1).

**Cartesian grid** A 2D Cartesian grid is a grid where all the grid lines are parallel to either x-axis or y-axis.

**IBM grid** The grid used for IBM. In an IBM grid, cells or nodes are classified into different types based on the type of discretization used on each cell or node (see Figure 2.2).

“Immersed Boundary Method” was developed by Peskin (1972) to compute the flow patterns around heart valves. Peskin placed a heart valve on a Cartesian grid, and to implement the boundary conditions on the heart valve, the flow equations were modified by including a forcing term in the governing equations that simulates the force applied by the immersed boundary on the fluid flow. Since Peskin’s introduction of this method, several refinements and modifications have been made to simulate different kinds of flows. In the literature, the IBM methods that use a forcing function to produce the effect of the boundary are broadly classified into continuous and discrete forcing methods. These methods, and the relevant research papers, can be found in the review paper by Mittal and Iaccarino (2005). The problem with these methods is that they use a smooth forcing function which distributes the effects of immersed boundary over a band of cells. This makes these methods unable to provide a sharp representation of the immersed boundary which is especially undesirable in high Reynolds number flows.
In the current study, it is desired that the immersed boundary is “sharply” represented. Therefore, methods which use a distributed forcing function will not be discussed further. Apart from methods that use forcing functions to produce the effect of the boundary, many other Cartesian grid based methods have been developed over the years which implement the boundary conditions directly on the grid nodes or cells next to the boundary. These methods are expected to produce a sharp representation of the immersed boundary which is desirable for the current study. These methods include Cartesian grid methods (Delanaye, Aftosmis, Berger, Liu, and Pulliam (1999)), cut cell methods (Ye et al. (1999)) and embedded boundary methods (Johansen and Colella (1998)). All of these methods use a non-boundary conforming Cartesian grid for flow simulations and differ from each other in the way they handle the immersed boundary. For this study, the immersed boundary methods which implement the boundary conditions directly, are categorised as IBM based on finite difference methods and IBM based on finite volume methods.
2.1 Finite Difference Methods Using IBM

For a clearer understanding, it is important to compare the finite difference methods on body-conforming grids and on IBM grid (refer table 2.1).

Table 2.1: FDM on boundary-conforming grid vs FDM on IBM grid

<table>
<thead>
<tr>
<th>#</th>
<th>FDM on boundary-conforming grids</th>
<th>FDM on IBM grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Construct boundary-fitted grid</td>
<td>Construct a Cartesian grid and place the body on the grid</td>
</tr>
<tr>
<td>2</td>
<td>Develop finite difference equations on the grid</td>
<td>Identify grid points in the fluid domain, solid domain and near the immersed boundary</td>
</tr>
<tr>
<td>3</td>
<td>Develop algorithm to get the solution</td>
<td>Construct interpolants that satisfy the boundary conditions</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Develop finite difference equations on the grid and couple the finite difference equations near the boundaries with the interpolants to ensure that the boundary conditions are satisfied</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Develop algorithm to get the solution</td>
</tr>
</tbody>
</table>

The basic steps involved in the generation of an IBM grid for finite difference methods are presented next. IBM grid generation process starts with the generation of a suitable Cartesian grid required for the simulation. Then the immersed boundary is appropriately placed on the Cartesian grid. This is followed by the identification of nodes as fluid, solid and ghost nodes (Figure 2.2). Details about this can be found in the Appendix.

The identification of grid points as fluid, solid and ghost nodes is an important step. The grid we obtain after the identification process is called the IBM grid. This
grid helps distinguish between the nodes where the flow equations are to be solved (fluid nodes), the nodes where the boundary conditions have to be implemented (ghost nodes), and the nodes which do not contribute to the fluid flow (solid nodes).

Next comes the development of finite difference equations on an IBM grid. There are 3 types of nodes where we need to formulate the finite difference equations:

- Fluid points: The equations on the fluid points would be constructed the same way the equations are solved on a regular boundary-conforming grid.

- Ghost points: At each ghost point, for each flow variable, an interpolant is constructed that satisfies the boundary conditions by using values at select points on the boundary surface and neighboring fluid points (yellow and green points in Figure 2.3). These interpolants are then used to find the value of the flow variable at that ghost point (red point in Figure 2.3). Notice that
the flow variable value calculated on the ghost nodes, implicitly incorporates the boundary condition on the immersed boundary. When performing finite difference on fluid points next to the boundaries, their finite difference stencils would include the ghost nodes and, in this way, the boundary condition is indirectly implemented on an IBM grid.

- Solid points: Flow equations on the solid points do not affect the solution in the flow domain. Many researchers prefer not to solve the flow equations on the solid points to save on computations. But if it is required to solve the flow equations inside the solid domain, then the flow equations would be the same as those on fluid points.

![Figure 2.3: Close up region near the immersed boundary showing classification of nodes](image)

Next, the interpolants used on the ghost nodes to implement the boundary conditions are discussed. From the review of the literature it has been identified that the following factors are important when constructing an interpolant:

- Order of the interpolant or type of interpolant used including least-square.
• The number and location of fluid points and boundary points used to construct the interpolant.

• Central or biased one-sided differencing for fluid points next to boundaries.

The literature provides a number of these interpolants, which will be discussed in reference to Figure 2.4. These interpolants shall be discussed to interpolate a generic flow variable $\phi$. The simplest of these interpolants is a linear interpolant (Majumdar et al. (2001), Tseng and Ferziger (2003)) given as

$$\phi = C_1 x + C_2 y + C_3.$$  \hspace{1cm} (2.1)

The coefficients $C_1$, $C_2$, $C_3$ can be found by satisfying the three data points $F_2$, $F_3$ and $B$. $B$ is generally the mid point of the line segment connected by $P_1$ and $P_2$. In some cases $B$ can also be the point obtained by dropping a perpendicular line from the ghost node onto the immersed boundary. This interpolant is less accurate.
as compared to higher order interpolants but has been used sometimes when the grid
resolution is high enough. Another simple option is bilinear interpolation which is
given as

\[ \phi = C_1 xy + C_2 x + C_3 y + C_4. \]  \hspace{1cm} (2.2)

The four coefficients in the above equation can be evaluated by satisfying the four
points \( F_1, F_2, F_3 \) and \( B \). Again \( B \) can be either the mid point or the point obtained
by dropping the perpendicular line from the ghost point on the immersed boundary.
Other higher order interpolation techniques include quadratic interpolation, as shown
in Eqn 2.3 (Tseng and Ferziger (2003)). One boundary point and five fluid points
or two boundary points and four fluid points can be used to find the six coefficients
required to construct the interpolant.

\[ \phi = C_1 x^2 + C_2 y^2 + C_3 xy + C_4 x + C_5 y + C_6 \]  \hspace{1cm} (2.3)

The interpolants can also be constructed with reference to the tangential and normal
direction of the immersed boundary. An interpolant of such type has been used by
Majumdar et al. (2001) which is linear in the tangential direction and quadratic in
the normal direction (Equation 2.4). Five points need to be satisfied to find the
five unknown coefficients in this case. Again, there is a choice of choosing points
\( P_1, P_2, F_1, F_2 \) and \( F_3 \) or using just one boundary point and four fluid points (for
example, \( B, F_1, F_2, F_3 \) and \( F_4 \)).

\[ \phi = C_1 n^2 + C_2 nt + C_3 n + C_4 t + C_5 \]  \hspace{1cm} (2.4)

Other interpolation techniques can also be used (Ghias et al. (2004)).

Often, extrapolation leads to numerical instabilities (Tseng and Ferziger (2003)).
In such cases image points are used to prevent any extrapolation. Image points are
mirror images of ghost points across the immersed boundary. These image points fall inside the fluid domain and the solution on the image points are interpolated using the interpolant constructed in the above discussion. The solution on the ghost point is then calculated using a simple linear extrapolation using the boundary point and the image point. More information about the use of image points can be found in Tseng and Ferziger (2003), Majumdar et al. (2001) and Ghias, Mittal, and Dong (2007). The paper Mittal et al. (2008), extends the method used in Ghias et al. (2004), Ghias et al. (2007) and Majumdar et al. (2001), to 3-D flows.

Once the interpolants are constructed, these interpolants are used to compute the solution on the ghost nodes. When the finite difference equation are solved on the fluid points next to boundaries, their computational stencil includes the ghost nodes. Thus, the value of the ghost nodes calculated using the interpolants is coupled with the finite difference equations in this manner. One can infer that the boundary condition on the immersed boundary is implicitly satisfied in this manner through the use of ghost nodes.

FDM using IBM are simple and elegant, but none of the methods discussed so far are designed to satisfy the conservation laws. In the next section, the IBM methods based on finite volume methods are discussed.

### 2.2 Finite Volume Methods Using IBM

Implementation of strict local and global conservation laws demands for a finite volume formulation of the governing equations and this is the primary motivation of studying FVM.

At first, a comparison between using FVM on boundary-conforming grids and FVM on IBM grid, has been shown in Table 2.2.

The IBM grid for finite volume methods is constructed first. Details about construction of IBM grid when using FVM can be found in the appendix.
Table 2.2: FVM on boundary conforming grid vs FVM on IBM grid

<table>
<thead>
<tr>
<th>#</th>
<th>FVM on boundary-conforming grids</th>
<th>FVM on IBM grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Construct boundary-fitted grid</td>
<td>Construct a Cartesian grid and place the body on the grid</td>
</tr>
<tr>
<td>2</td>
<td>Develop finite volume equations on the grid</td>
<td>Identify cells in the fluid domain, solid domain and on the immersed boundary</td>
</tr>
<tr>
<td>3</td>
<td>Develop algorithm to get the solution</td>
<td>Develop finite volume equations on fluid, solid and immersed cells</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Develop algorithm to get the solution</td>
</tr>
</tbody>
</table>

Figure 2.5.: Finite volume IBM grid for 2D circular cylinder
The IBM grid when using finite volume formulation is shown in Figure 2.5 for a 2-D cylinder. The identification of fluid, solid and immersed cells is important because it helps distinguish between the cells where the flow equations are solved on a full Cartesian cell (i.e. fluid and solid cells) and the cells where the flow equations are solved on an arbitrary cell (i.e. immersed cells). When using a cell centred approach, it is important to locate the cell centers as shown in Figure 2.6.

Figure 2.6.: Finite volume IBM grid for 2D cylinder showing cell centers

The method and concept of using IBM methods with finite volume formulation is discussed next. The first step is to develop the finite volume equations. It is followed by the discretization of finite volume equations for full cells i.e. fluid and solid cells. Next, the finite volume equations are discretized on the immersed cells. Most of the IBM methods based on finite volume differ in the way they handle the immersed cells. In this thesis, two of such methods will be discussed i.e. Cut Cell Method (Udaykumar et al. 1996, 1997, 2001; Ye et al. (1999)) and Embedded Boundary Methods (Johansen and Colella (1998)).
Before going over the equations on the immersed cells, the assumptions and requirements when handling immersed cells are discussed first. It is required that the grid is fine enough such that any face of the cell does not cut the immersed boundary more than once. The second requirement is that the grid should be fine enough such that the immersed boundary, if cuts any cell, should not cut more than two of its faces. This eliminates many complicated cases. The remaining cases possible are shown in Figure 2.7. There are three possible cases with four possible orientations for each case, making a total of twelve possible types of immersed cells as shown.

![Figure 2.7: Types of immersed cells encountered](image)

As discussed earlier, the immersed cells are the only cells in a grid which contain the immersed boundary. Clearly, some of the portion of an immersed cell belongs to the fluid domain and the remaining portion belongs to the solid domain. Also, some, or all of the faces of the immersed cell are partially in fluid domain and partially in solid domain. Thus, it becomes important to think about how to calculate the correct fluxes \((F_e, F_w, F_n, F_s)\) passing through the partial faces of the
immersed cells. One extra face also appears, which is the immersed boundary that cuts through the immersed cells. For the boundary face, the flux \( F_f \), needs to be calculated. Lastly, the size of the fluid portion of the immersed cells could get very small. This can restrict the stability criterion for time marching. All these problems have been addressed in the two methods (Cut cell methods and Embedded boundary methods) which are discussed next.

2.2.1 Cut Cell Method

In this method, the immersed cells are cut by the immersed boundary so that the solid portion of the immersed cells are discarded and the grid fits the boundary. This results in the modified cells next to the boundary as shown in Figure 2.8(a). The fluxes needing evaluation are shown in Figure 2.8(b). The cell type 3 and sometimes type 2 tend to get very small and put a severe restriction on the stability condition. In general, the immersed cells whose cell centers lie inside the solid domain are considered to be “small cells”. To address this issue, the small cells are merged with neighbouring cells. This process is called “cell reshaping” or “cell merging”. This leads to the formation of control volumes as shown in Figure 2.9.

Solving a flow equation using finite volume discretization requires computation of flux integrals on the faces of each cell and, as discussed earlier, the issue is to calculate these fluxes on the faces of cut cells. To solve this issue, Ye et al. (1999) proposed constructing a 2-D interpolant in an appropriate region which would be used to evaluate the flux \( F \) wherever required. The interpolant proposed for this purpose would be explained in context of Figure 2.10. For the trapezoidal cell in this Figure, flux through the north face is easy to calculate since it is a regular Cartesian face. There is no south face so the flux through for this face would be zero. Flux through east, west and boundary face are more difficult and an interpolant is used to calculate the flux through these faces. The west face is split into two i.e. west face \( F_w \) and south west face \( F_{sw} \) as can be seen in Figure 2.10. \( F_w \) is easy to calculate since it
(a) 3 possible cases of cut cells

(b) Fluxes through the faces

Figure 2.8.: Cut cells

Figure 2.9.: Cell reshaping: Cell with its center inside the solid boundary gets merged into neighbouring cell
Figure 2.10.: 6 point stencil used to construct the interpolant to calculate $F_{sw}$

is a full Cartesian face. To calculate the flux $F_{sw}$ for a generic flow variable $\phi$, a 2-D interpolant is used, which is given as,

$$\phi = C_1 xy^2 + C_2 y^2 + C_3 xy + C_4 y + C_5 x + C_6$$

(2.5)

where, the six coefficients $C_1$ to $C_6$ are calculated by satisfying the six stencil points (see Figure 2.10) in the equation 2.5. Once the interpolant is constructed, the location of $F_{sw}$ is put in the equation to get the value of the flow variable $\phi$. If the normal derivative through the face is of interest at that point, then the analytical derivative of Equation 2.5 is used:

$$\frac{\delta \phi}{\delta x} = C_1 y^2 + C_3 y + C_5$$

(2.6)
A similar approach can be employed to find the east-face flux ($F_e$). The boundary flux ($F_f$) is calculated using a similar approach but the stencil used is modified (see 2.11). This approach is designed to be globally and locally second order accurate and satisfies mass and momentum conservation.

![Diagram](image)

Figure 2.11.: 6 point stencil used to construct the interpolant to calculate $F_f$

2.2.1.1. Issues with Cut Cell Methods

The method is second order accurate as discussed, but there unused information on the boundary. When constructing the interpolant, the only data which is exactly known is the boundary condition on the immersed boundary, but, all of this information has not been used. Consider a general case of no-slip wall boundary condition on the immersed boundary, which gives the $u$ and $v$ velocity as zero. The interpolant for $u$ velocity would be given using equation 2.5 as,
\[ u = C_1xy^2 + C_2y^2 + C_3xy + C_4y + C_5x + C_6 \] (2.7)

Since the velocity on the boundary is zero, therefore, if the \( u \) velocity is set zero in the above equation, it should produce an equation which represents the boundary:

\[ 0 = C_1xy^2 + C_2y^2 + C_3xy + C_4y + C_5x + C_6 \] (2.8)

Clearly, the assumption made was that the boundary is approximated as a linear element, but the above equation is not an equation of a line. Instead it is some random curve over which the boundary condition is being satisfied. This is the case for each and every flow variable. If \( v \) velocity is set to zero in Equation 2.5, then different equations representing the boundary are obtained.

This issue is attributed to not fully utilising the information known on the immersed boundary. When using the interpolants, the boundary conditions are being satisfied only on select point on the boundary. An infinite number of curves can pass through these select points and thus this is not an accurate representation for immersed boundaries.

2.2.2 Embedded Boundary Method

This method was introduced by Johansen and Colella (1998) which is a finite volume method based on Cartesian grids. The way the immersed cells are handled in this method are different compared to cut cell method. The IBM grid used in this technique is the same as shown in Figure 2.5 and 2.6. The type of immersed cells encountered are the same as shown in Figure 2.7.

As discussed earlier, the issue is to calculate the fluxes through the Cartesian faces and boundary face on the immersed cells. In this method, the cells are not merged. Instead the cells are all handled individually no matter how small they are.
The solution on immersed cells is treated as a cell center quantity even if the cell centers are outside the fluid domain. Here, it is assumed that the solution can be extended smoothly a small distance beyond the immersed boundary inside the solid domain.

The flux through partial cell faces are calculated using second order accuracy. This method requires some additional information for implementation. A parameter $\alpha$ is defined for each face of all the immersed cells as the ratio of the length of the face in the fluid domain with the full length of the face. This essentially provides an estimation of the percentage of face in the fluid domain (Figure 2.12).

Consider cell $i, j$ in Figure 2.13. The fluxes through the faces are calculated through the center of the part of the face in the fluid domain to achieve second order accuracy (Figure 2.13(a)). This is done by linearly interpolating the flux from the nearby full face centers (Figure 2.13(b)). Consider calculating the flux through the west face of cell $i, j$ where flux is given as normal gradient of the flow variable $\phi$ multiplied by face area $\alpha_w dy$. The flux for this face is linearly interpolated from the nearby full face centers as shown in Figure 2.13(b). The linear interpolation results in the following formula,
\[ F_w = \alpha_w dy \left[ \frac{1 + \alpha_w}{2} \left( \frac{\phi_{i-1,j} - \phi_{i,j}}{dx} \right) + \frac{1 - \alpha_w}{2} \left( \frac{\phi_{i-1,j+1} - \phi_{i,j+1}}{dx} \right) \right] \] (2.9)

Flux through remaining Cartesian faces are calculated using the same procedure. To calculate the flux through the boundary face \( (F_f) \), a normal line is extended from its center inside the fluid domain and the first pair of parallel grid lines passing through the cell centers that intersect with this normal are selected (the blue points in Figure 2.14). These grid lines should not pass through the current cell’s center. The value of flow variable \( \phi \) at these points is found by interpolating from nearby data points. To obtain a second order accurate gradient on the boundary face, the following equation is used,

\[
\frac{\delta \phi}{\delta n} = \frac{1}{d_2 - d_1} \left( \frac{d_2}{d_1} (\phi_f - \phi_1) - \frac{d_1}{d_2} (\phi_f - \phi_2) \right)
\] (2.10)

and the flux through the boundary is given by multiplying the boundary face area contained in that cell.

2.2.2.1. Issues with Embedded Boundary Method

The method is overall second order accurate but again the boundary conditions have been implemented on select points. A lot of exact information is available on the boundary which can be used to further improve the accuracy of the solution.
(a) Fluxes through the immersed cells based on finite volume formulation

(b) Linear interpolation to calculate flux through partial face

Figure 2.13.: Figure showing how to calculate fluxes with second order accuracy on the Cartesian faces of immersed cells
Figure 2.14.: Stencil to interpolate the boundary flux
CHAPTER 3. RESEARCH OBJECTIVES

Existing methods satisfy the boundary condition on selected “points” on the boundary. Typically, one point is used for each boundary surface in a cut cell. Clearly, satisfying boundary condition on just a few “points” on the boundary does not ensure that the boundary condition is being satisfied on the entire boundary. A method needs to be developed which uses the data available on the entire boundary. To capture the boundary conditions on the entire boundary and account for the non-linearity of the boundary, transfinite interpolation has been used.

The objectives of this thesis are:

- Develop a new IBM technique that satisfies the boundary condition on the entire boundary.
- Implement and verify the method developed via potential flow past a circular cylinder.
- Compare the method developed with existing IBM methods on accuracy and performance.
CHAPTER 4. NEW METHOD: TRANSFINITE INTERPOLATION

The goal of the new IBM method is to ensure that the boundary conditions are truly satisfied on the entire boundary surface. This is done using transfinite interpolation that constructs an interpolant to match a given function on a set of curves (refer Gordon and Hall (1973)). This will be demonstrated by constructing interpolants for two cases: parallel lines and arbitrary curves.

4.1 Transfinite Interpolation on Parallel Straight Lines

Consider a Cartesian coordinate system with \( \xi \) and \( \eta \) directions (see Figure 4.1). Now, consider a set of \( N \) parallel lines in this coordinate system given as,

\[
\mathbf{C}_n = (\xi)\mathbf{i} + (\eta_n)\mathbf{j} \quad \text{for } n = 0 \text{ to } N - 1
\]  

(4.1)

where, \( \mathbf{C}_n \) represents the \( n^{th} \) line. Data for a function \( \phi \) is available on these \( N \) lines and is given by \( \phi_n(\xi) \). The objective is to construct an interpolant which interpolates the variable \( \phi \) between the given lines while satisfying the given data on the lines. For this purpose, the following formula is used:

\[
\phi(\xi, \eta) = \sum_{i=n}^{N} \phi_n(\xi)f_n(\eta)
\]  

(4.2)

where, \( f_n(\eta) = \frac{\prod_{k \neq n}(\eta - \eta_k)}{\prod_{k \neq n}(\eta_n - \eta_k)} \)  

(4.3)

and \( f_n \) is called the blending function. Equation 4.2 shows a transfinite interpolation formula based on the use of lagrange polynomial blending functions.
4.2 Transfinite Interpolation on Arbitrary Curves

Transfinite interpolation to interpolate a function on arbitrary curves is a two-step process:

1. Develop a map \( \tilde{T} : P \rightarrow C \) which maps the arbitrary curves in the physical domain (P) to parallel lines in the computational domain (C). This map provides correspondence between the function \( \phi(x, y) \) in the physical domain and the function \( \phi^*(\xi, \eta) \) in the computational domain.

2. Develop transfinite interpolation in the computational domain which interpolates the function \( \phi^*(\xi, \eta) \) in between the given curves.

Transfinite interpolation for a function \( \phi \) on arbitrary curves is demonstrated next. Consider a set of \( N \) arbitrary curves \( C_0, C_1, \ldots, C_{N-1} \) whose parametric equations are given by,

\[
\tilde{C}_n = x_n(\xi)i + y_n(\xi)j \quad \text{where } n = 0 \text{ to } N - 1
\]  

(4.4)

where \( \xi \) is a parameter in the range \([0,1]\). The function \( \phi \) is available on these curves and is given by,

\[
\phi(x_n, y_n)
\]  

(4.5)
The first step is to map the arbitrary curves in the physical domain \((x, y)\) to parallel straight lines in the computational domain \((\xi, \eta)\). For this purpose, each curve \(C_n\) associated with a constant \(\eta\) line in the computational domain. Now, the entire physical domain is mapped to the computational domain using the following equations:

\[
\begin{align*}
x(\xi, \eta) &= \sum_{n=0}^{N} x_n(\xi) f_n(\eta) \quad (4.6) \\
y(\xi, \eta) &= \sum_{n=1}^{N} y_n(\xi) f_n(\eta) \quad (4.7)
\end{align*}
\]

where, \(f_n(\eta) = \frac{\prod_{k\neq n}(\eta - \eta_k)}{\prod_{k\neq n}(\eta_n - \eta_k)} \quad (4.8)\)

The mapping \(\overrightarrow{T}\) is thus given as

\[
\overrightarrow{T} = x(\xi, \eta)\vec{i} + y(\xi, \eta)\vec{j} \quad (4.9)
\]

Using the mapping \(\overrightarrow{T}\) the function \(\phi\) in the physical and computational domain is related as

\[
\phi(x, y) = \phi(x(\xi, \eta), y(\xi, \eta)) = \phi^*(\xi, \eta) \quad (4.10)
\]

The second step is to interpolate the solution in the computational domain. At first, the data \(\phi(x_n, y_n)\) on the given arbitrary curves is mapped to the computational domain by substituting for \(x_n\) and \(y_n\) to give,

\[
\phi(x_n, y_n) = \phi(x_n(\xi), y_n(\xi)) = \phi_n(\xi) \quad (4.11)
\]
Next, $\phi$ is interpolated between the lines in the computational domain using the
equations developed in previous section,

$$
\phi^*(\xi, \eta) = \sum_{n=1}^{N} \phi_n(\xi)f_n(\eta)
$$

(4.12)

where,

$$
f_n(\eta) = \frac{\prod_{k\neq n}(\eta - \eta_k)}{\prod_{k\neq n}(\eta_n - \eta_k)}
$$

(4.13)

Once $\phi^*(\xi, \eta)$ has been interpolated, $\xi$ and $\eta$ can be substituted by $x$ and $y$ using
the map $\bar{T}$ as shown in Equation 4.11. This would only be possible if an explicit
expression for $\xi$ and $\eta$ can be developed in terms of $x$ and $y$. But, this is generally
not the case since it is difficult to find an explicit expression for $\xi$ and $\eta$. In such
cases, a different approach can be used which is explained in the context of boundary
conditions to be implemented on the immersed boundary.

![Figure 4.2.: Mapping of physical domain to computational domain](a) Physical domain   (b) Computational domain)

Figure 4.2.: Mapping of physical domain to computational domain

4.3 Implementation

Two types of boundary conditions are dealt with here:

1. Dirichlet type: Specifies the values that a solution needs to take at the boundary.
2. Neumann type: Specifies the values that the derivative of a solution needs to take at the boundary.

4.3.1 Dirichlet Boundary Condition

Consider a case where Dirichlet boundary condition for a variable $\phi$ is given on the boundary (refer Fig. 4.3). When calculating the boundary flux, two situations might occur depending upon the flow equations. Either the flow variable $\phi$ is required or the gradient of $\phi$ is required on the boundary. Both the cases are addressed here.

Figure 4.3.: Dirichlet boundary condition

When the value of flow variable is required to calculate the boundary flux, the given Dirichlet boundary condition can be directly used. To capture the boundary exactly, the flux is integrated on the boundary using appropriate analytical or numerical integration schemes.

$$Boundary Flux = \int_{l_b} \phi dl$$  \hspace{1cm} (4.14)

where, $l_b$ is the length of the boundary where the flux is to be found.

When the value of the gradient of the flow variable is required to calculate the boundary flux, a transfinite interpolation is developed on the boundary (refer
equations 4.12). Since the transfinite interpolation gives \( \phi \) as a function of \( \xi \) and \( \eta \), the gradients with respect to \( x \) and \( y \) cannot be calculated directly. To find the gradients, the following equations are used:

\[
\frac{\delta \phi}{\delta x} = \frac{\delta \xi}{\delta x} \frac{\delta \phi^*}{\delta \xi} + \frac{\delta \eta}{\delta x} \frac{\delta \phi^*}{\delta \eta} \tag{4.15}
\]

\[
\frac{\delta \phi}{\delta y} = \frac{\delta \xi}{\delta y} \frac{\delta \phi^*}{\delta \xi} + \frac{\delta \eta}{\delta y} \frac{\delta \phi^*}{\delta \eta} \tag{4.16}
\]

Henceforth, the following notations will be used:

\[
\frac{\delta \xi}{\delta x} = \xi_x \quad \frac{\delta \xi}{\delta y} = \xi_y \quad \frac{\delta \eta}{\delta x} = \eta_x \quad \frac{\delta \eta}{\delta y} = \eta_y \tag{4.17}
\]

\[
\frac{\delta x}{\delta \xi} = x_\xi \quad \frac{\delta x}{\delta \eta} = x_\eta \quad \frac{\delta y}{\delta \xi} = y_\xi \quad \frac{\delta y}{\delta \eta} = y_\eta \tag{4.18}
\]

These are called metric coefficients. Since an explicit expression for \( \xi \) and \( \eta \) as a function of \( x \) and \( y \) is not available, the metric coefficients are found using the following equations:

\[
\xi_x = \frac{y_\eta}{J} \quad \xi_y = -\frac{x_\eta}{J} \quad \eta_x = -\frac{y_\xi}{J} \quad \eta_y = \frac{x_\xi}{J} \tag{4.19}
\]

where \( J \) is called the Jacobian. \( J \) is given as:

\[
J = x_\xi y_\eta - x_\eta y_\xi \tag{4.20}
\]

The above equations can be solved either analytically or numerically to calculate the Jacobian and metric coefficients. Using the metric coefficients, equations 4.15
and 4.16 are solved to find the gradients. Once the gradients are available, they are integrated over the boundary to calculate the flux as shown:

$$\text{Boundary Flux} = \int_{l_b} (-\frac{\delta \phi}{\delta x} dy + \frac{\delta \phi}{\delta y} dx)$$

(4.21)

where $l_b$ is the length of the boundary curve.

4.3.2 Neumann Boundary Condition

Consider a case where Neumann boundary condition for a variable $\phi$ is given on the boundary (Figure 4.4). When calculating the boundary flux, 2 situations might occur depending upon the flow equations. Either the flow variable $\phi$ or the gradient $\frac{d\phi}{dn}$ is required. Both the cases are addressed here.

Figure 4.4.: Neumann boundary condition

When the gradient of the flow variable is required to calculate the boundary flux, the given Neumann boundary condition can be directly used. To capture the boundary exactly, the flux is integrated on the boundary using appropriate analytical or numerical integration schemes.
When the value of flow variable is required to calculate the boundary flux, a transfinite interpolation is developed on the boundary first (Equation 4.12). To find the value of flow variable on the boundary, the gradient is calculated by differentiating the polynomial and is equated to the Neumann boundary condition given on the boundary to obtain Equation 4.23.

\[
\frac{\delta \phi}{\delta n} = \frac{-\frac{\delta \phi}{\delta x} dy + \frac{\delta \phi}{\delta y} dx}{\sqrt{dx^2 + dy^2}}
\] (4.23)

Substituting the values of gradients from Equations 4.15 and 4.16 in Equation 4.23 gives,

\[
\frac{\delta \phi}{\delta n} = \frac{-(\xi_x \frac{\delta \phi^*}{\delta \xi} + \eta_x \frac{\delta \phi^*}{\delta \eta}) dy + (\xi_y \frac{\delta \phi^*}{\delta \xi} + \eta_y \frac{\delta \phi^*}{\delta \eta}) dx}{\sqrt{dx^2 + dy^2}}
\] (4.24)

Using Equation 4.24, the value of the flow variable \( \phi \) can found on the boundary. Once the value of flow variable is available, it is integrated over the boundary using Equation 4.14 to calculate the flux.
CHAPTER 5. GOVERNING EQUATIONS AND NUMERICAL METHOD

5.1 IBM for Potential Flow

The governing equation for potential flow is the Laplace equation given by

$$\nabla^2 \psi = \frac{\delta^2 \psi}{\delta x^2} + \frac{\delta^2 \psi}{\delta y^2} = 0.$$ \hspace{1cm} (5.1)

The flow variable is stream function denoted by $\psi$. To solve the equation, a time derivative term is added to the governing equation to give

$$\frac{\delta \psi}{\delta t} - \nabla^2 \psi = 0.$$ \hspace{1cm} (5.2)

This allows the numerical method to march in time. Since this is a steady state problem, the time derivative term approaches zero at the end of the simulation which gives the solution to the original laplace equation (Equation 5.1). In potential flow simulations, the boundary is treated as a constant streamline. Therefore, the Dirichlet boundary condition is implemented by

$$\psi_b = \text{constant}$$ \hspace{1cm} (5.3)

where, $\psi_b$ is the value of stream-function on the boundary. The flow equation is solved using a finite volume formulation. Equation 5.1 is in strong conservation form. The equations are integrated over a differential volume element to get Equation 5.4.
Using divergence theorem, Equation 5.4 is transformed from a volume integral over the volume \( V \) to a surface integral over the boundary of the volume \( V \) (Equation 5.5).

\[
\iiint_{V} \frac{\delta \psi}{\delta t} dV - \nabla^2 \psi dV = 0 \tag{5.4}
\]

Using divergence theorem,

\[
\iiint_{V} \frac{\delta \psi}{\delta t} dV - \nabla^2 \psi dV = \iiint_{V} \frac{\delta \psi}{\delta t} dV - \iiint_{S} (\nabla \psi \cdot \vec{n}) dS = 0 \tag{5.5}
\]

Equation 5.5 needs to be solved numerically over the IBM grid. When using finite volume formulation on an IBM grid, three types of cells are encountered: fluid, solid and immersed cells (Figure 5.1). The solid and fluid cells are full cells since they do not encounter the boundary. The immersed cells are the only cells on an IBM mesh which contain the immersed boundary. Equations on the three types of cells are shown next.

5.1.1 Fluid Cells

The stencil for fluid cells is shown in Fig. 5.2. Equation 5.5 are integrated over cell \( i, j \) to give Equation 5.6.

\[
\frac{\delta \bar{\psi}_{i,j}}{\delta t} V - (F_e + F_w + F_n + F_s) = 0 \tag{5.6}
\]

where,

- \( \bar{\psi}_{i,j} \) = Average of \( \psi \) over cell \( i, j \)
- \( V = dx dy \)
- \( F_e \) = Flux leaving the cell through east face
- \( F_n \) = Flux leaving the cell through north face
- \( F_w \) = Flux leaving the cell through west face
Figure 5.1.: Cell types

\[ F_s = \text{Flux leaving the cell through south face} \]

Hereafter, the average value over a cell \( \bar{\psi} \) would be written without the bar over \( \psi \).

The fluxes through each face are approximated using central differencing as shown next:

\[
F_e = \frac{\psi_{i+1,j} - \psi_{i,j}}{dx} \\
F_w = \frac{\psi_{i-1,j} - \psi_{i,j}}{dx} \\
F_n = \frac{\psi_{i,j+1} - \psi_{i,j}}{dy} \\
F_s = \frac{\psi_{i,j-1} - \psi_{i,j}}{dy}
\]  

(5.7)

This concludes the discretization of diffusive fluxes. Now, the time derivative term is discretized using an Euler explicit equation as shown in Equation 5.8. The solution
for cell $i, j$ is updated on every time step using Equation 5.9. The fluxes in the equation are computed on time level $n$.

$$\frac{\psi_{i,j}^{n+1} - \psi_{i,j}^n}{dt} V - (F_e + F_w + F_n + F_s) = 0 \tag{5.8}$$

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n - \frac{dt}{V} ((F_e + F_w + F_n + F_s)) = 0 \tag{5.9}$$

Figure 5.2.: A rectangular differential volume element

5.1.2 Solid Cells

Clearly there in no fluid flow in the solid domain so it would be perfectly fine if the flow equations are not solved on the solid cells. It would not affect the flow solution in the fluid and immersed cells. But, sometimes it might be required to solve the flow equations inside the solid domain as well. This would include situations, where not solving the flow equations inside the solid domain would be more computationally expensive and situations when the fluid flow inside the solid domain would be of
interest, for e.g., in potential flow theory, flow over a 2D cylinder is made up of uniform flow over a doublet. Therefore, when solving flow inside the solid cells (for potential flow over 2D cylinder) it is expected to see a doublet inside the cylinder. When solving the flow equations on the solid cells, the same equations developed for fluid cells are used since the solid cells are also full Cartesian cells. Flow equations and interpolants are formulated such that the flow solution on solid cells will not affect the flow solution on fluid and immersed cells in any way.

5.1.3 Immersed Cells

Equation 5.5 when integrated on an immersed cell (for e.g. see Figure 5.3(a)) gives the following equation,

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n - \frac{dt}{V}((F_e + F_w + G_n + G_s + F_f)) = 0$$

(5.10)

where $F_f$ is the flux through the boundary face. The Cartesian cell faces are handled using the Embedded boundary method of Johansen and Colella (1998). When calculating the boundary fluxes, both the original method on Johansen and Colella (1998) and the new method developed in this thesis is used and compared with the results of the existing methods. The formulations through the partial cell faces and the boundary face are shown next.

5.1.3.1. Flux Through Cartesian Cell Faces

When calculating the fluxes on the partial Cartesian faces, the gradients should still be on the midpoint of that face (see $F_w$ and $F_n$ in Fig. 5.3(a)). For this purpose, the gradients are calculated using a linear interpolation of gradients from the neighbouring faces as shown in Fig. 5.3(b). This gives a second order accurate formulation of the gradients. The fluxes through all the Cartesian faces for the immersed cell shown in Fig. 5.3 are as follows:
\[ F_e = 0 \]
\[ F_w = \alpha_w dy \left[ \frac{1 + \alpha_w}{2} \left( \frac{\psi_{i-1,j} - \psi_{i,j}}{dx} \right) + \frac{1 - \alpha_w}{2} \left( \frac{\psi_{i-1,j} - \psi_{i,j+1}}{dx} \right) \right] \]
\[ F_n = \alpha_n dy \left[ \frac{1 + \alpha_n}{2} \left( \frac{\psi_{i,j+1} - \psi_{i,j}}{dy} \right) + \frac{1 - \alpha_n}{2} \left( \frac{\psi_{i+1,j} - \psi_{i-1,j}}{dy} \right) \right] \]
\[ F_s = 0 \]

A similar procedure is performed for all other cell types when calculating the fluxes through the cell faces.

A figure showing fluxes through the faces of an immersed cell and another figure showing use of linear interpolation to calculate the fluxes on the partial faces.

Figure 5.3.: Fluxes and their computation on immersed cells

5.1.3.2. Flux Through Boundary Face

The flux calculation through the boundary face is presented in this section. Three methods have been employed to calculate the boundary flux. The first one
is the original method as described in the paper of Johansen and Colella (1998) which will be called “IBM1” here on. The Second method will be called “IBM2” which has been employed widely when calculating the flux through the boundary face when using Cut cell methods UdayKumar, Shyy, and Rao (1996). The third method is the “PRESENT Method” developed for this thesis which employs “Transfinite Interpolation” to calculate the boundary flux. Both IBM1 and IBM2 approximate the curved immersed boundary as a linear element inside each immersed cell as shown in Figure 5.4.

Figure 5.4.: Figures showing the original boundary and its approximation as linear elements

5.1.3.2.1. IBM1

This approach has been discussed in the literature review section. The boundary flux is calculated using the stencil shown in Figure 5.5 and the following equation,
\[ F_f = \frac{d_f}{d_2 - d_1} \left( \frac{d_2}{d_1} (\phi_f - \phi_1) - \frac{d_1}{d_2} (\phi_f - \phi_2) \right) \] (5.12)

This gives a gradient which is second order accurate in the normal direction.

Figure 5.5.: Stencil to interpolate the boundary flux using IBM1

5.1.3.2.2. IBM2

Before moving further the terminology “Bilinear interpolation” and “Bilinear interpolation stencil” used later will be explained first. Consider the grid shown in figure 5.6. Assume that the solution is known on all the cell centres. Consider point P as shown in the figure. The data at this point when found using bilinear interpolation means that the four points A, B, C and D are used to find the solution \( \psi_P \) on the point P using a bilinear interpolation formula given by,
\[
\psi_P = \frac{(\psi_A dx_2 + \psi_B dx_1) dy_1 + (\psi_C dx_2 + \psi_D dx_1) dy_2}{(dx_1 + dx_2)(dy_1 + dy_2)}
\] (5.13)

Here, the stencil used for interpolation contains the four points \(A, B, C,\) and \(D\) which surround the point \(P\). This stencil will be called the bilinear interpolation stencil from here forth.

In IBM2, the points required for computing the gradient on the boundary are found in a different manner as compared to IBM1. A normal is extended from the mid-point of the boundary face into the fluid domain and the first point is placed on the normal at a distance \(d_1\). The second stencil point is placed at a distance \(d_2\) from the boundary point on the same normal (Figure 5.7). The data on the two stencil points is calculated using a bilinear interpolation from the surrounding four points. In the cited paper UdayKumar et al. (1996), only one point is chosen on the extended
normal to calculate the boundary flux. Whereas, here two points are chosen which gives a second order accurate gradient and is comparable to the IBM1. The flux is calculated using the following equation,

$$F_f = \frac{d_f}{d_2 - d_1} \left( \frac{d_2}{d_1} (\phi_f - \phi_1) - \frac{d_1}{d_2} (\phi_f - \phi_2) \right)$$

(5.14)

The distance $d_1$ and $d_2$ are found using the following equation,

$$d_1 = \beta (dx^2 + dy^2)^{1/2}$$

$$d_2 = 2d_1$$

(5.15)

where, $\beta$ is a parameter used to vary the value of $d_1$. A numerical study for showing the variation of solution by varying the parameter $\beta$ is shown in the section “Study of Solution Variation with $\beta$” of Chapter 6. Based on the numerical study, $\beta$ is chosen to be one which is considered a safe value. With $\beta$ as one, the distance $d_1$ is equal to $(dx^2 + dy^2)^{1/2}$. This value is equal to the diagonal length of the Cartesian cell which is the longest length which can be contained inside a Cartesian cell of dimensions $dx$ and $dy$. It ensures that the first point lies far enough from the immersed boundary such that its bilinear interpolation stencil does not include the solid cells and it also ensures that the two points do not have the same bilinear interpolation stencil.

5.1.3.2.3. PRESENT Method

The thinking process FOR “PRESENT Method” which uses transfinite interpolation is as follows: The solution is known on the entire immersed boundary (boundary condition). We wish to use this entire information when constructing the interpolant. Also, we require the polynomial to be second order accurate in the normal direction to have a fair comparison with the previous two methods. To get a second order accurate interpolant in normal direction, three layers would be needed. The first layer is the immersed boundary itself. The other two layers are constructed
Figure 5.7.: Stencil to interpolate the boundary flux using IBM2

as follows: Let the boundary contained inside the immersed cell be represented by $C_0$ with end points $A$ and $B$ (Figure 5.8). At first, normals are extended from points $A$ and $B$ and two points are chosen on each normal (points $C$, $E$ and $D$, $F$ in Figure 5.8). The points are chosen at distances $d_1$ and $d_2$ given by the following equations,

$$d_1 = \beta (dx^2 + dy^2)^{1/2}$$
$$d_2 = 2d_1$$

(5.16)

The solution on the four points $C$, $D$, $E$ and $F$ are found using bilinear interpolation. Next, the first points on the two normals i.e. $C$ and $D$ are connected using a straight line represented by $C_1$ and the next two points i.e. $E$ and $F$ are connected with a straight line as well and this line will be called $C_2$. Thus, the three layers used for interpolation have been constructed. Now, the boundary curve $C_0$ and the two lines $C_1$ and $C_2$ are parameterized as follows,
Figure 5.8.: Stencil to interpolate the boundary flux using Transfinite interpolation

Curve $C_0$: $x_0 = x(\xi) \quad y_0 = y(\xi)$

Curve $C_1$: $x_1(\xi) = x_C + \xi(x_D - x_C) \quad y_1(\xi) = y_C + \xi(y_D - y_C)$

Curve $C_2$: $x_2(\xi) = x_E + \xi(x_F - x_E) \quad y_2(\xi) = y_E + \xi(y_F - y_E)$

where, $\xi$ is a parameter which varies in the interval $[0, 1]$. A general representation for curve $C_0$ is used here because the parametric equations of the immersed boundary would depend on the body over which the flow is being simulated.
Now, we have the parameterized equations of three curves. These three curves are mapped to three constant \( \eta \) lines in the computational domain i.e. Curve \( C_0 \) is mapped to \( \eta_0 = 0 \), \( C_1 \) is mapped to \( \eta_1 = 1 \) and \( C_2 \) is mapped to \( \eta_2 = 2 \) as shown in Figure 5.9. The physical domain is then mapped to the computational domain using the following formula,

\[
x(\xi, \eta) = \sum_{i=0}^{2} x_i(\xi) f_i(\eta) \\
y(\xi, \eta) = \sum_{i=0}^{2} y_i(\xi) f_i(\eta)
\]

where, \( f_i(\eta) = \frac{\prod_{j \neq i}(\eta - \eta_j)}{\prod_{j \neq i}(\eta_i - \eta_j)} \)  

(5.18)

Figure 5.9.: Mapping the physical domain to computational domain
The solution on these three curves is found next. The solution on curve $C_0$ is the boundary condition $\psi_f$. The solution on $C_1$ and $C_2$ represented by $\psi_1$ and $\psi_2$ respectively, are developed using a linear interpolation from the two end points as shown,

Solution on $C_0$: $\psi_0 = \psi_f$

Solution on $C_1$: $\psi_1(\xi) = \psi_C + \xi(\psi_D - \psi_C)$

Solution on $C_2$: $\psi_2(\xi) = \psi_E + \xi(\psi_F - \psi_E)$

Now, a transfinite interpolation is developed on the computational domain which interpolates the solution on the three curves,

$$
\psi(\xi, \eta) = \sum_{i=0}^{2} \psi_i(\xi) f_i(\eta) \\
\text{where, } f_i(\eta) = \frac{\prod_{j\neq i}(\eta - \eta_j)}{\prod_{j\neq i}(\eta_i - \eta_j)}
$$

(5.20)

As discussed in Chapter 4, to calculate the gradient in the physical domain, the following formulas are used,

$$
\frac{\delta \psi}{\delta x}(\xi, \eta) = \xi \frac{\delta \psi}{\delta \xi} + \eta_x \frac{\delta \psi}{\delta \eta} \\
\frac{\delta \psi}{\delta y}(\xi, \eta) = \xi_y \frac{\delta \psi}{\delta \xi} + \eta \frac{\delta \psi}{\delta \eta}
$$

(5.21)

The metric coefficients are calculated as discussed in Chapter 4. When the gradients are calculated on the boundary curve $C_0$, the value of $\eta$ in the above equation is set to be 0. Since $\psi$ is constant on the boundary, the term $\frac{\delta \psi}{\delta \xi}$ becomes 0. This results in the following equation which is a variable in $\xi$,

$$
\frac{\delta \psi}{\delta x}(\xi, 0) = \eta_x \frac{\delta \psi}{\delta \eta} \\
\frac{\delta \psi}{\delta y}(\xi, 0) = \eta_y \frac{\delta \psi}{\delta \eta}
$$

(5.22)
The flux is then integrated on the immersed boundary using the following equation,

$$Boundary\ Flux = \int_{C_0} \left(-\frac{\delta\psi}{\delta x}\ dy + \frac{\delta\psi}{\delta y}\ dx\right) \quad (5.23)$$
CHAPTER 6. FLOW PAST A CIRCULAR CYLINDER

In this chapter, the IBM developed in Chapter 5 will be assessed by applying that method to study the flow past a circular cylinder. This chapter is organised as follows. We start with the problem description, followed by the formulation and numerical method. In the last section of this chapter, the results are presented.

6.1 Problem Description

Flow over a circular cylinder has been chosen for assessing the new method because extensive literature is present for this case and lots of analytical and experimental data are available to compare with the solutions. A circular cylinder of diameter 0.1 m is used as shown in Figure 6.1.

6.2 Formulation

The flow over the circular cylinder will be studied by formulating the potential flow. For potential flow, the governing equation is,

\[ \nabla^2 \psi = 0; \] (6.1)

with the boundary conditions given by no penetration on the cylinder boundary i.e. \( \psi = \text{constant} \) on the boundary. The boundary conditions on the flow domain are as follows:
\[ Inlet: v = -\frac{\delta \psi}{\delta x} = 0 \]
\[ Outlet: v = -\frac{\delta \psi}{\delta x} = 0 \]
\[ Farfield Top: u = \frac{\delta \psi}{\delta y} = 1 \]
\[ Farfield bottom: u = \frac{\delta \psi}{\delta y} = 1 \] (6.2)

The flow domain is twelve times the diameter of the cylinder in length and width as shown. The cylinder is placed in the center of the flow domain. The domain size is selected to be large enough such that the boundary does not affect the flow over cylinder.

![Figure 6.1.: Figure showing the Computational domain for the flow over a 2D cylinder](image)
6.3 Numerical Method

The finite volume immersed boundary method for potential flow as shown in chapter 5 has been used. The cylinder is placed on a uniform Cartesian grid and the Cartesian cells are identified as solid, fluid and immersed cells. The fluid and solid cells are solved using Equations 5.7 and 5.9. The immersed cells are solved using Equations 5.10 and 5.11. The flux on the boundary face is calculated using the IBM1, IBM2 and PRESENT method as demonstrated in Chapter 5. When using PRESENT method, the boundary curve is parameterized using the following equation,

\[
x(\xi) = r \cos(2\pi \xi) \\
y(\xi) = r \sin(2\pi \xi)
\]  (6.3)

where \( \xi \) is a parameter which varies between \([0, 1]\) and \( r \) is the radius of the cylinder.

6.4 Results

6.4.1 Grid Refinement Studies

In this section, the solution behaviour is studied as the grid is refined. The performance of the three numerical methods are tested on four grids (31x31, 61x61, 121x121 and 241x241). The solutions on these grids are compared with the analytical solution in order to assess the accuracy with which the solutions are being calculated on each grid.

The grids used are Cartesian grids with uniform grid spacing. An example of the grid used (31x31) is shown Figure 6.2. Figure 6.3 shows the 4 grids used for this study. In these figures, the grid has been magnified near the immersed boundaries to highlight the grid quality.
Figure 6.2.: An example of the grid used for flow simulations
Figure 6.3.: Grids used for grid sensitivity studies
(Magnified near immersed boundary)

Figure 6.4 shows the grid sensitivity study for the Embedded Boundary Method of Johansen and Colella (referred to as IBM1 in this thesis). Here, the figures include a comparison for x-velocity, y-velocity, and Pressure Coefficient.
The legend of the graphs is shown separately for clarity. It can be observed that with increasing grid size, the numerical solution is marching towards the analytical solution. Although, it can also be observed that the solution on 121x121 grid is comparable or better than the solution on 241x241 grid. This seems to be because...
of the choice of points used to interpolate the boundary flux (the points on 121x121
grid overshoot).

Figure 6.5 shows the grid sensitivity study for IBM2. IBM2 is implemented
with using $\beta = 1$ in Equation 5.15 which is a safe value and does not result in
overshooting solutions on any grid. With increasing grid size, the solution is observed
to march towards the analytical solution.
Figure 6.5.: Grid Sensitivity Studies for the IBM2

Figure 6.6 shows the grid sensitivity study for the “PRESENT method” which uses Transfinite interpolation. The $\beta$ value, when making the stencil used for transfinite interpolation, is taken to be 1 to prevent any possible overshoots in the solution (similar to IBM2). From the figure, it is clear that with increasing grid size the solution is marching towards the analytical solution on the mesh.
Figure 6.6.: Grid Sensitivity Studies for the present method
All the numerical methods converge towards the analytical solution as the grid size is increased. This verifies the numerical methods used in this study.

6.4.2 Study of Solution Variation with $\beta$

In this section the effect of varying $\beta$ in Equation 5.15. This test is performed on two grids: a coarse grid (61x61) and a fine grid (121x121). For this test, the values of $\beta$ are varied in the range $[0.1, 2]$.

On the coarse grid it is observed that the solver diverges for values of $\beta$ less than 0.2 (Figure 6.8). With $\beta = 0.2$, the solution quality is very poor due to unpredictable overshoots and undershoots on various locations. The reason for this behaviour is believed to be the smaller distance of the probe from the immersed boundary which could result in the value at the probe point being interpolated from a solid cell. Because solid cells are not a part of the flow regime, this would result in unrealistic data to be interpolated on the probe points thereby causing instabilities in the solution. Next, choosing $\beta = 0.5$ provides a more stable solution. This value of $\beta$ ensures that the probe points are far enough so that the value at any probe point is not interpolated using the data on solid cells. $\beta = 0.75$ can be seen to predict better results than $\beta = 0.5$. Further decreasing the value of $\beta$ results in poorer prediction of solution as can be seen in the Figures 6.8. Another important point to note is that choosing $\beta = 1$ or higher would ensure that the two probe points used, are far enough from each other that they use difference bilinear interpolation stencils for getting the solutions. It is desirable that the two probe points take values from different interpolation stencils so that the information from a wider stencil is available. Therefore, $\beta$ values larger than 1 would be expected to provide a stable and smooth solution.

On the fine grid, it is observed that the solver does not converge for $\beta = 0.2$. Next, $\beta = 0.5$ provides a stable solution as expected. It also provides the best solution for this grid. $\beta = 1$ provides a better solution than $\beta = 0.75$ in this case. As $\beta$
value is increased further, the solution predictions become worse (but are stable as expected). For the purpose of this study, based on this numerical experiment, the value of $\beta$ is chosen as 1 for all the flow simulations. Although it may not provide the best prediction, it is expected to predict stable solution without any undershoots and overshoots. The same cannot be said for $\beta$ values less than 1 as is observed from the numerical study.
Figure 6.7.: Solution variation on 2-D cylinder with variation in $\beta$ for a 61x61 grid
Figure 6.8.: Solution variation on 2-D cylinder with variation in $\beta$ for a 121x121 grid
6.4.3 Accuracy Comparison

In this section the three numerical methods are compared on various grid sizes ranging from very coarse to very fine grids. Figures 6.9 and 6.10 show the comparison of the three numerical methods used on coarse grids. The PRESENT method is observed to predict better results when compared to the existing two methods on coarse grids. Moving further, the three methods are compared on finer grids as shown in Figure 6.11 and 6.12. On 121x121 grid, IBM1 appears to provide a comparable or better solution than the present method. But as discussed earlier, the IBM1 on 121x121 grid seems to over predict the result since it is even better than the solution on 241x241 grid. Lastly, on the 241x241 grid, the PRESENT method predicts better result than the two existing methods.
Figure 6.9.: Comparison of the 3 numerical methods on 31 x 31 grid
Figure 6.10.: Comparison of the 3 numerical methods on 61 x 61 grid
Figure 6.11.: Comparison of the 3 numerical methods on 121 x 121 grid
The relative error for the solutions on the four grids are presented in Figure 6.13. These errors are plotted for the pressure coefficient $C_p$. The relative error graph becomes meaningless at certain locations because the analytical solution for $C_p$ becomes zero or near zero, and this causes division by small numbers. Therefore, absolute error plots are shown in Figure 6.14 which are more meaningful to study.
For all grids, it can be observed that the present method predicts better solutions when compared to the existing methods.
Figure 6.13.: Relative Error for $C_p$ on different grid sizes
Figure 6.14.: Absolute Error for $C_p$ on different grid sizes
6.4.4 Performance Comparison

Table 6.1: Comparison of CPU time for the 3 methods on the 4 grid sizes used

<table>
<thead>
<tr>
<th>Method used</th>
<th>31 x 31</th>
<th>61 x 61</th>
<th>121 x 121</th>
<th>241 x 241</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM1</td>
<td>1.11</td>
<td>4.55</td>
<td>141.37</td>
<td>471.7</td>
</tr>
<tr>
<td>IBM2</td>
<td>0.815</td>
<td>4.67</td>
<td>106.54</td>
<td>451.8</td>
</tr>
<tr>
<td>PRESENT method</td>
<td>1.036</td>
<td>5.279</td>
<td>135.27</td>
<td>470.59</td>
</tr>
</tbody>
</table>
CHAPTER 7. SUMMARY AND FUTURE WORK

7.1 Summary

In this study, a new IBM method has been developed which implements the boundary conditions on the entire immersed boundary, thereby taking advantage of all the exact information available about the flow boundary conditions. For implementing the boundary conditions, a transfinite interpolation technique has been used. This method has been demonstrated by applying it to solve potential flow past a circular cylinder. The reason for choosing potential flow of a circular cylinder is the availability of the analytical solution for this problem.

Results from the newly developed method are compared with the results from two of the existing methods and the new method is found to predict better solutions on coarse as well as fine grids. The solutions are also observed to depend on the placement of the probe points used to calculate the boundary fluxes. A numerical study has been performed to decide upon the appropriate location of these points. Based on the numerical study and other observations, an appropriate technique for choosing the probe points has been suggested.

Additionally, since the new method implements the boundary condition on the entire boundary, there is no need for refining the grid due to excessive boundary curvature.

7.2 Future Work

Currently, work is being done to extend the current IBM based flow solver to an incompressible viscous flow solver. Future research includes,
• Finishing the validation for the incompressible viscous flow solver.

• Extending the flow solver to operate on non-uniform Cartesian grids.

• Extending the flow solver to solve 3-D flows.

• Improving the grid quality next to the immersed boundaries by employing automatic prismatic grid generation thus, extending the grid induced errors in the fluid domain away from the boundaries.
APPENDICES
APPENDIX A: IDENTIFICATION OF NODES WHEN EMPLOYING FDM WITH IBM

The definition of fluid, solid and ghost nodes is as follows:

**Fluid nodes** Nodes which are outside the solid boundary. The discretized flow equations are solved on these nodes.

**Solid nodes** Nodes which are inside the solid boundary. The flow equations do not apply to the solid domain.

**Ghost nodes** Nodes near the immersed boundary which are used to implement the boundary conditions.

![Figure A.1.: Figure showing concept of ray casting algorithm](image)

(a) Point inside the boundary  (b) Point outside the boundary

The nodes of the Cartesian grid are identified using the ray casting algorithm (refer Haines (1994)) which is one of the most popular algorithms when using immersed boundary methods. Ray casting algorithm is a technique which detects whether a point lies inside or outside of a closed polygon. Thus, applying this algorithm on each grid node on a Cartesian grid identifies if the grid nodes are inside or
outside of the immersed boundary. The algorithm to identify if a point \( P \) is inside or outside of a closed boundary surface is as follows:

- Project a ray from point \( P \) to a bounding box that fully contains the immersed boundary in an arbitrary direction
- Compute the number of times the ray intersects the boundary of the solid
- Decide if the point \( P \) is inside or outside of the boundary:
  - If the number of intersections are odd, then the point \( P \) is inside the boundary. (see Figure A.1(a))
  - If the number of intersections is zero or even, then the point \( P \) is outside the boundary. (see Figure A.1(b))
- Some exceptions to this odd-even rule exist (for e.g. see point \( P’ \) in Figure A.1(b)). In such cases, an additional set-up can be added which determines if the ray at the point of intersection is a tangent to the boundary. If the ray is a tangent, then the contribution of this intersection is counted as two intersections.
- Decide on fluid, solid and ghost points: Outside points are in the fluid domain and are called fluid points. Inside points are in the solid domain and are called solid points. To implement the boundary conditions, some solid points near immersed boundary are used as ghost points. Solid points, which have at least one neighbour (in x or y direction) as fluid point are called ghost points. Figure 2.2 shows the application of ray casting algorithm to identify the fluid, solid, and ghost nodes for a 2-D circular cylinder on a Cartesian grid.

Here, a general classification of nodes is shown for the purpose of explaining the methods clearly. There can be (and are) other methods which use a slightly different classification. For example, some higher order methods, when not using biased
differencing on the boundaries, may require two or more layers of ghost nodes which would give a different IBM grid (Anupindi, Delorme, Shetty, and Frankel (2013)).
APPENDIX B: IDENTIFICATION OF CELLS WHEN EMPLOYING FVM WITH IBM

The procedure for developing an IBM grid for finite volume follows the same steps which were followed in finite difference method until finding the inside and outside nodes. Once, the inside and outside nodes are determined, the cell types are defined using the following definitions:

**Fluid Cells** Cells whose nodes are all outside points.

**Solid Cells** Cells whose nodes are all inside points.

**Immersed Cells** Cells whose nodes include at least one of both inside and outside points. Immersed cells are the only cells which contain the immersed boundary.
LIST OF REFERENCES


