

APPLICATION OF THE FICTITIOUS DOMAIN METHOD TO FLOW PROBLEMS
WITH COMPLEX GEOMETRIES

A Dissertation

by

HUNG-CHIEH CHU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Yassin A. Hassan
Co-Chair of Committee,	Wolfgang Bangerth
Committee Members,	Anastasia Muliana
	Sevan Goenezen
Head of Department,	Andreas Polycarpou

May 2017

Major Subject: Mechanical Engineering

Copyright 2017 Hung-Chieh Chu

ABSTRACT

In this study, in order to address the immersed boundary condition, which was the critical issue regarding the fictitious domain method, two new strategies for addressing the numerical integral related to the immersed boundary condition were introduced. In the first strategy, the constraint was set to live everywhere, but only equaled the desired values in the area outside the needed domain. As to the second strategy, a boundary region was conceptually generated to replace the immersed boundary. An additional function, $k(x)$, was added as a weight function to validate this replacement. Both of these strategies transfer boundary integrals to domain integrals that all computations can be finished by using the mesh generated for the fictitious domain. In addition, in order to deal with large scale problems, a modified iterative algorithm was proposed.

Three different types of problems were studied to evaluate the capability of these two strategies. It is shown that both of these two strategies are capable of addressing problems with only one variable. However, the study of the Stokes problem indicates the second strategy is a superior choice to deal with problems with multiple variables. Finally, from the Navier-Stokes flow problem, it is concluded that the second strategy can solve large scale flow problems with complex geometries.

ACKNOWLEDGEMENTS

First, I would like to thank my chair advisor, Dr. Hassan, for his great help and guidance. As a department head of the nuclear engineering, he is a busy person. However, he always tried to give me some time from his tight schedule and reminded me some important issues.

Second, I would like to thank my two committees, Dr. Muliana and Dr. Goenezen. They not only joined my preliminary exam and dissertation defense, but also offered their valuable suggestions.

I would like to give my great appreciation to Dr. Bangerth, who was also the main supervisor of this research. In addition to mentoring me, he also gave me a lot help in many aspects. In addition, he taught me how to do correct things, which is the biggest gain I have gotten in the past four and a half years. Moreover, he is the kindest professor I have ever met. He is always friendly to his colleagues and students. It is sad that my poor English is not capable of showing my gratefulness to him.

Here, I would like to give my wish to my friend, Sam Wu. We met in this school and both got PhD degrees from the same department. During this process, we were friends sharing with our depressions and difficulties. Wish him and his families have great future.

Last, but not the least, I would like to mention the people that I am most grateful for, my parents and my wife. As the oldest son of my parents, spending many years to pursue the PhD degree should not be an appropriate choice. However, my dear parents

unconditionally supported me. I hope I can repay them in the near future. I also want to say thank you to my parents in law for letting their daughter marrying me. As to my wife Yi-Chin, if she did not marry me, she would enjoy a colorful life in Taiwan without suffering some bad things, such as the language barrier. However, she still made her decision to resign her job and came with me. Her accompany gave me a great help in my life. The most thankful thing she made is to deliver and take care of our adorable daughter, Livia. I cannot imagine a life without them.

CONTRIBUTORS AND FUNDING SOURCES

This work was supervised by a dissertation committee consisting of Professor Yassin A. Hassan of the Department of Nuclear Engineering (co-advisor), Professor Wolfgang Bangerth of the Department of the Mathematics (co-advisor), Associate Professor Anastasia Muliana of the Department of Mechanical Engineering and Assistant Professor Sevan Goenezen of the Department of Mechanical Engineering.

All work for the dissertation was completed independently by the student.

This work was made possible in part by National Science Foundation (NSF) under Grant Number 1148116. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the Division of Grants and Agreements (DGA).

NOMENCLATURE

ω	Original complex domain.
Ω	Fictitious domain.
γ	Immersed boundary.
Γ	Boundaries of Ω .
g_1	Immersed boundary condition.
u or \mathbf{u}	Original solutions valid in ω .
\tilde{u}	Solutions valid in Ω .
u_h	Approximate of \tilde{u} .
v	Test function of \tilde{u} .
v_h	Approximate of v .
λ	Lagrange multiplier.
λ_h	Approximate of λ .
μ	Test function of λ .
μ_h	Approximate of μ .
p	Pressure.
p_h	Approximate of p .
q	Test function of p .
q_h	Approximate of q .
ε	Penalty parameter.
r_i	Radius of inner obstacles.

$\bar{\gamma}$	Boundary region.
hf	Half width of the boundary region.
h	Mesh size.
dis	Distance between an arbitrary point and the reference point.
$dist(x, \bar{\gamma})$	Shortest distance measured from a quadrature point to γ .
$d(x)$	Distance between a quadrature point and the reference point.
σ	Standard deviation of Gaussian function.
η	Kinematic viscosity of the fluid.
$w_1(x)$	Weight function in Strategy 1.
$w_2(x)$	Weight function in Strategy 2..
ξ	Stopping criterion of the loop/inner loop.
τ	Stopping criterion of the outer loop.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
CONTRIBUTORS AND FUNDING SOURCES.....	v
NOMENCLATURE.....	vi
TABLE OF CONTENTS	viii
LIST OF FIGURES.....	xi
LIST OF TABLES	xv
CHAPTER I INTRODUCTION	1
1.1 Foreword	1
1.2 Fictitious Domain method.....	5
1.2.1 Penalty function.....	6
1.2.2 Lagrange multiplier	9
1.2.3 Immersed boundary method.....	15
1.3 Parallel computing	16
1.3.1 Standards regarding the parallelization	17
1.3.2 Combination of the FD method and the parallelization	19
1.4 C++ library deal.II.....	20
1.5 Outline of this dissertation	20
CHAPTER II POISSON PROBLEM	22
2.1 Introduction	22
2.2 Test case setting	22
2.3 Fictitious domain method implementation.....	24
2.3.1 Fictitious domain formulation.....	25
2.3.2 IB condition treatment: Strategy 1	29
2.3.3 IB condition treatment: Strategy 2	30
2.4 Numerical details	35
2.5 Experimental results.....	37
2.5.1 Results from Strategy 1	37

2.5.1.1	Solution profile from Strategy 1	38
2.5.1.2	Error analysis of Strategy 1	38
2.5.2	Results from Strategy 2	40
2.5.2.1	Solution profile from Strategy 2	41
2.5.2.2	Error analysis of Strategy 2	41
2.5.2.3	Irregular behavior for in narrow boundary	48
2.6	Summary	49
CHAPTER III STOKES FLOW PROBLEM		50
3.1	Introduction	50
3.2	Test case setting	51
3.3	Fictitious domain method implementation	55
3.3.1	Fictitious domain formulation	55
3.3.2	IB condition treatment: Strategy 1	57
3.3.3	IB condition treatment: Strategy 2	58
3.4	Iterative algorithm	60
3.4.1	Stopping criterion	67
3.4.2	Krylov subspace method	67
3.5	Numerical details	68
3.6	Experimental results	69
3.6.1	Results from Strategy 1	69
3.6.1.1	Solution profile from Strategy 1	70
3.6.1.2	Error analysis of Strategy 1	72
3.6.2	Results from Strategy 2	74
3.6.2.1	Solution profile from Strategy 2	74
3.6.2.2	Error analysis of Strategy 2	75
3.6.3	Influence from the loop part stopping criterion ξ	79
3.7	Summary	83
CHAPTER IV NAVIER-STOKES PROBLEM		84
4.1	Introduction	84
4.2	Code parallelization	85
4.3	Fictitious domain method implementation	88
4.4	Advection term linearization and Iterative algorithm modification	89
4.5	Numerical details	90
4.6	Experimental results	90
4.6.1	Test case 1: Parallel Poiseuille flow	91
4.6.1.1	Error analysis	91
4.6.1.2	Influence from the outer loop stopping criterion τ	93
4.6.1.3	Study regarding the tolerance of the solver	97
4.6.1.4	Efficiency of the code parallelization	98
4.6.2	Test case 2: 2D lid driven cavity flow multiple obstacles	100

4.6.3 Test case 3: 3D channel flow with multiple obstacles	104
4.7 Summary	109
CHAPTER V CONCLUSIONS AND RECOMMENDATIONS	111
5.1 Conclusions	111
5.2 Recommendations for future work.....	112
REFERENCES	114

LIST OF FIGURES

	Page
Figure 1 Additional block (black) added for connection in the grid embedding method (refer to Chu [7]).	3
Figure 2 PWR fuel assembly and spacer grid spring (from Kim et al. [8]).	4
Figure 3 (a) Mesh generated without using the FD method, the red circles point out places with low quality mesh; (b) mesh generated by applying the FD method.	5
Figure 4 Conceptual figure regarding the strategy used in Glowinski et al. [9], red spots represent the intersection point between the irregular boundary and the mesh.	12
Figure 5 Concept of the XFEM in the FD method. T is a bad element which has small intersection with the immersed boundary Γ_D , T' is a good element neighboring T , shape function of T' will be extended. (refer to [24]).	14
Figure 6 The approximate area for judging the bad and good element. Point A is a vertex of this cell, points B and C are intersection points between the immersed boundary (red curve line) and this mesh.	15
Figure 7 Flow domain for the Poisson problem.	24
Figure 8 (a) The original domain ω and its boundary γ , (b) Ω is the fictitious domain for the Poisson problem.	25
Figure 9 Mesh generated for the fictitious domain, Ω	26
Figure 10 Boundary region γ for the Poisson problem.	33
Figure 11 Conceptual plot regarding the Strategy 2, (a) Constant function, (b) Triangle function, (c) Gaussian function.	35
Figure 12 Mesh and obtained solution distribution of the Poisson problem obtained from the direct result and Strategy 1, (a) and (c) are meshes used in the direct simulation and Strategy 1, (b) and (d) are their corresponding results.	38

Figure 13 L_2 norm of the error and H^1 norm of the error of the Poisson problem obtained from Strategy 1.	40
Figure 14 Mesh and obtained solution distribution of the Poisson problem obtained from the direct result and Strategy 2 (with constant function), (b) and (d) are their corresponding results.	41
Figure 15 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.	43
Figure 16 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.	45
Figure 17 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.	47
Figure 18 An example to explain why the results are not stable when constant c is smaller than 1; the star symbols represent the positions of quadrature points.	49
Figure 19 Flow domain for the Stokes flow problem.	52
Figure 20 Fictitious domain for the Stokes flow problem.	55
Figure 21 Extension of the immersed boundary condition after applying Strategy 1.	58
Figure 22 Boundary region $\bar{\gamma}$ for the Stokes flow problem.	59
Figure 23 Mesh and obtained velocity profile of the Stokes flow problem obtained from the direct simulation.	70
Figure 24 Velocity profile of the Stokes flow problem obtained from Strategy 1.	71
Figure 25 Velocity profile without imposing the immersed boundary condition.	72
Figure 26 L_2 norm of the error and H^1 norm of the error of the Stokes flow problem obtained from Strategy 1 with $\xi = \mathbf{1} \cdot \mathbf{0}h$	73
Figure 27 Velocity profile of the Stokes flow problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.	74

Figure 28 L_2 norm of the error and H^1 semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.	76
Figure 29 L_2 norm of the error and H^1 semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.	77
Figure 30 L_2 norm of the error and H^1 semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.	79
Figure 31 L_2 norm of the error of the Stokes flow problem obtained from Strategy 1 with $\xi = 1.0h$ and $\xi = 0.25h$	80
Figure 32 L_2 norm of the error of the Stokes flow problem obtained from Strategy 2 with $\xi = 1.0h$ and $\xi = 0.25h$	82
Figure 33 L_2 norm of the error and H^1 semi-norm of Test case 1 obtained from the code for solving the NS system with the constant function used as the weight in the integrals over the boundary region.	92
Figure 34 Improvement from the smaller outer loop stopping criterion.	96
Figure 35 Speedup S_N and Efficiency E_P of our implementation.	99
Figure 36 Velocity vector and distribution of the lid driven cavity flow with an obstacle in the flow domain.	101
Figure 37 Velocity vector and distribution of the lid driven cavity flow with five obstacles in the flow domain.	102
Figure 38 Adjusted velocity distribution of the lid driven cavity flow with five obstacles in the flow domain.	103
Figure 39 Velocity vector and distribution of the lid driven cavity flow with fourteen obstacles in the flow domain.	104
Figure 40 Fictitious domain for Test case 3.	105
Figure 41 Velocity vector and distribution of the channel with an obstacle in the flow domain when the outer loop stopping criterion is 0.025.	106

Figure 42 Velocity vector and distribution of the channel with an obstacle in the flow domain when the outer loop stopping criterion is 0.01.....	106
Figure 43 Slice output for demonstrating a flow passing around two balls.....	107
Figure 44 Velocity vector and velocity distribution of the channel with two obstacles in the flow domain.....	108
Figure 45 Velocity vector and velocity distribution of the channel with two obstacles in the flow domain.....	109

LIST OF TABLES

	Page
Table 1 Results from Strategy 1	39
Table 2 Results from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.	42
Table 3 Results from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.	44
Table 4 Results from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.	46
Table 5 Results from Strategy 1 with $\xi = 1.0h$	73
Table 6 Results from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.	75
Table 7 Results from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.	77
Table 8 Results from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.	78
Table 9 L_2 norm of the error from Strategy 1 with different stopping criterion.	80
Table 10 L_2 norm of the error from Strategy 2 with different stopping criterion	82
Table 11 Results from the code for solving the NS system with the constant function used as the weight in the integrals over the boundary region.	92
Table 12 Needed inner iterations in each outer iteration step, inner iteration errors after each inner iteration step obtained under the setting with $c = 1.0$, $h = 1/128$ and $\xi = 1.0h$	95
Table 13 Comparison between results from different outer loop stopping criterion τ	95
Table 14 Results with different solver tolerance.....	98
Table 15 Efficiency of the code parallelization.	99

CHAPTER I

INTRODUCTION

1.1 Foreword

Due to the fast development of computer technology, computational fluid dynamics (CFD) has become a powerful tool for studying engineering problems. For people who work on this subject, problems with complex geometries are usually hard to address. Directly generating body-fitted meshes may not be a suitable choice since large numbers of cells need to be assigned to keep the smoothness of the grids used for describing irregular boundaries, which is computationally expensive.

The grid patching method is an alternative to address this difficulty. The application of this method enables its user to subdivide the whole flow domain into several sub-blocks whose shapes are easier for mesh generation (Lee and Rubbert [1]). With this treatment, the user can customize suitable grids for each sub-block. Compared to directly generating body-fitted mesh, this method reduces the difficulty regarding mesh generation, especially when there are multiple complex geometries. The drawback of this method is that it is error-prone if the node arrangement on the common boundary of two typed meshes loses conformity. The conformity means that adjacent cells that share the same edge also share its endpoints as their joint vertices (Shapira [2]). This confinement may cause memory waste while connecting a block with dense meshes to a block with coarse meshes.

The grid embedding method is another alternative for solving this puzzle. Similar to the grid patching method, this method also divides the physical domain into several smaller regions. However, in this method, two neighboring blocks no longer share a common boundary. Instead, these two meshes are overlapped. Communication among each region is now accomplished by interpolation at grid boundaries. This method not only keeps the advantage of the grid patching method, but also overcomes the limitation regarding the node arrangement on common boundaries. Details about its implementation can be seen in Benek et al. [3] and Benek et al. [4]. In addition to its positives, however, several downsides may trouble the user of this method. First, for communication among each sub-block, a detailed description regarding the interpolations points is needed, which is hard if there are multiple blocks overlapped at the same regions. Petersson [5] proposed a hole-cutting technique for solving this difficulty, which was efficient but also increased the complexity of this method. In addition, some modifications are necessary to prevent the interpolation error, like the one described in Hubbard and Chen [6]. In their research, the pressure/velocity coupling scheme was modified to conserve the continuity of the incompressible flow. Moreover, generating additional blocks for connecting two sub-blocks with very different shapes may be inevitable. An instance can be seen in Figure 2. In this instance, in order to solve a problem with a circular pipe (green colored part) partially connected to a U-shaped channel (red colored part), an additional block (black colored part) needs to be added. Setting this kind of structure not only needs user's experience but also additional memory. The resulting data structure also becomes much more complex.

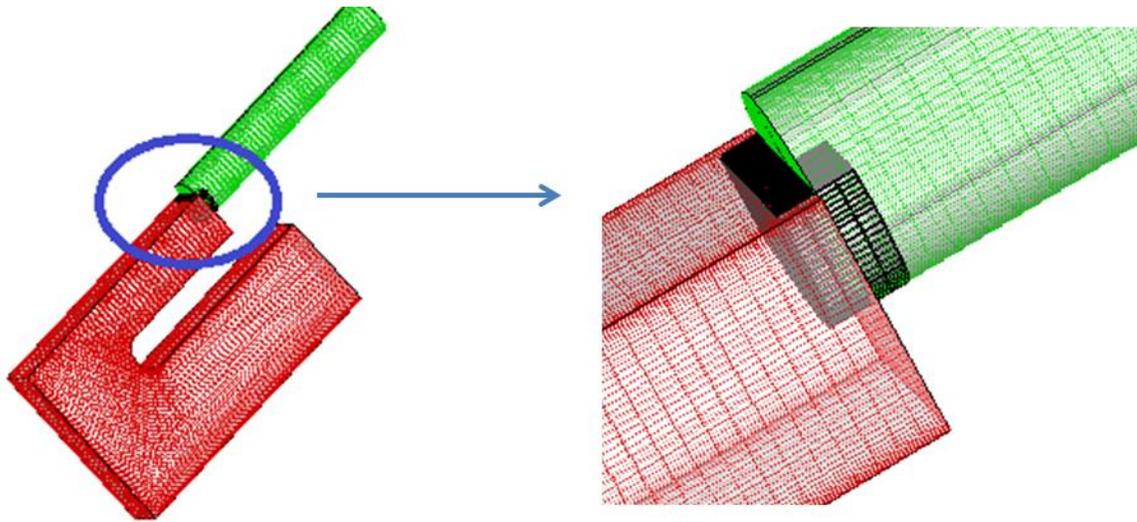


Figure 1 Additional block (black) added for connection in the grid embedding method (refer to Chu [7]).

Both of these two methods have the ability to address flow problems with complex geometries. However, they also have some drawbacks, which will be severe when the size of the problem is large.

Simulation regarding the Pressurized Water Reactors (PWRs) is an evident example. In PWRs, the fuel, Uranium dioxide (UO_2), is contained within fuel rods. These fuel rods will be further grouped to become fuel bundles (Figure 1(a)). A large reactor has 150~250 such assemblies. In addition to these fuel rods, there are many spacer grids which are used to guide flow (Figure 1(b)). Moreover, the springs and dimples in these grids further increase the complexity (Figure 1(c)). No matter if the user wants to apply the grid patching method or the grid embedding method, subdividing this domain is difficult. Large amounts of memory may be wasted due to node arrangement along common boundaries or be used to record interpolations points. In fact, even with

today's computers, trying to simulate an entire reactor core is still impossible. As a result, finding alternatives is needed.

In this research, in order to reduce the difficulties with simulating large scale flow problems with complex geometries, we study the Fictitious Domain method (FD method).

A brief introduction and literature review about the FD method and its related methods are provided in the next section. In addition, in order to solve large scale problems, parallelization is inevitable. Some fundamental knowledge regarding this issue is mentioned in Chapter 1.3. After that, we introduce the C++ software library, deal.II. All implementation in this study is accomplished by using this powerful tool. In the final part of this chapter, we provide an overview of this dissertation.

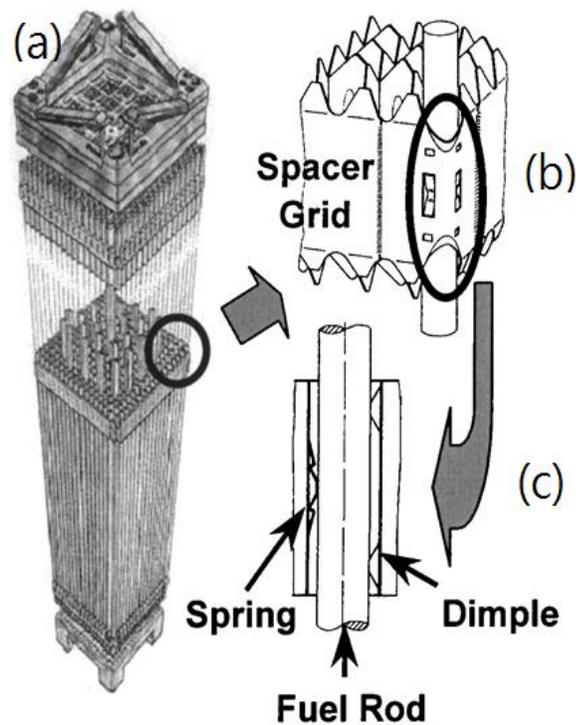


Figure 2 PWR fuel assembly and spacer grid spring (from Kim et al. [8]).

1.2 Fictitious Domain method

The FD method, also called the domain imbedding method (Glowinski et al. [9]), was originally proposed by Russian mathematician Saul'ev in 1963. The idea of this method is that whenever a problem needs to be solved on a domain with an irregular boundary, it might be useful to embed it into a larger domain of a simpler shape (Quarteroni and Valli [10]). After this operation, the user only needs to generate the mesh for the larger domain. Due to the simplicity of the larger domain (i.e., embedding domain), the difficulties regarding the mesh generation is diminished (see Figure 3).

The critical issue of the FD method lies in how to evaluate the influence from the desired boundary condition on the immersed boundary (IB). As mentioned above, while using the FD method, the mesh is generated for the fictitious domain, thus, the original irregular boundary becomes immersed. Therefore, it is not straightforward to impose any boundary condition on the IB. Next, we are going to introduce some approaches invented for addressing this difficulty.

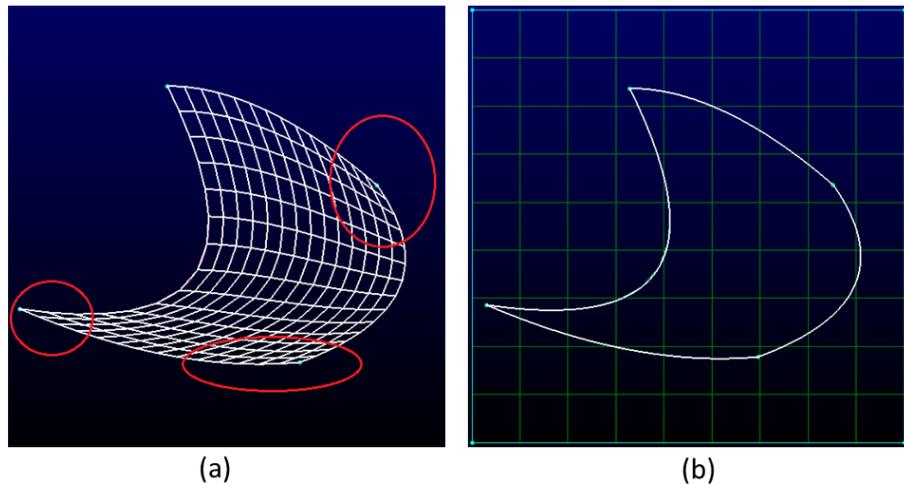


Figure 3 (a) Mesh generated without using the FD method, the red circles point out places with low quality mesh; (b) mesh generated by applying the FD method.

1.2.1 Penalty function

The use of a penalty function for imposing the boundary value constraint comes from the theory of constrained optimization problems. Solutions obtained from this approach is not only the minimum value or the maximum value of a function, but also satisfies desired constraints (Reddy [11]).

To get an insight of this approach, we first consider a constrained optimization problem whose domain has an irregular boundary (e.g., Figure 3(a)):

$$\text{minimize } J(u),$$

$$\text{subject to the constraint } G(u) = g_1,$$

with

$$J(u) = \frac{1}{2} \int_{\omega} |\nabla u|^2 dx - \int_{\omega} (f \cdot u) dx, \quad (1-1)$$

$$u = G(u) \quad \text{on } \gamma, \quad (1-2)$$

where γ represents the boundary of the domain ω on which we want to solve the PDE.

With a well-defined function space, the solution u can be proved to be the solution of the Poisson problem:

$$-\Delta u = f \quad \text{in } \omega, \quad (1-3)$$

$$u = g_1 \quad \text{on } \gamma. \quad (1-4)$$

Next, by applying the penalty parameter ε , a penalty functional for this problem can be obtained:

$$J_{\varepsilon}(u) = \frac{1}{2} \int_{\omega} |\nabla u|^2 dx - \int_{\omega} (f \cdot u) dx + \frac{1}{2\varepsilon} \int_{\gamma} (u - g_1) dx. \quad (1-5)$$

The variational principle states that the solution of this equation can be found by taking the variation of $J_\varepsilon(u)$ in all directions v and requiring them to be equal to zero. This yields the following weak formulation:

$$\int_{\omega} (\nabla v \cdot \nabla u) dx + \frac{1}{\varepsilon} \int_{\gamma} (v \cdot u) d\gamma = \int_{\omega} (v \cdot f) dx + \frac{1}{\varepsilon} \int_{\gamma} (v \cdot g_1) d\gamma, \quad (1-6)$$

for all test functions v in $H^1(\omega)$. We can also write its differential form as:

$$-\Delta u + \frac{1}{\varepsilon} \cdot u \cdot \delta|_{\gamma} = f + \frac{1}{\varepsilon} \cdot g_1 \cdot \delta|_{\gamma}. \quad (1-7)$$

The Dirac delta function, δ , in equation (1-7) equals infinity on the immersed boundary γ . Otherwise, it equals to zero.

In equation (1-6), the two integral terms involving penalty parameter ε are called penalty functions and are related to the constraint on the boundary γ . In theory, the use of the penalty function is capable of imposing the IB condition. However, since in the fictitious domain there is no definite expression about the position of the immersed boundary γ , it is hard to define where the penalty functions exist.

This puzzle can be solved by considering the basic concept of the FD method. While using this method, even though the needed domain (e.g., ω) is embedded in a larger domain (e.g., Ω), solutions in the domain ω are still what the user wants. Therefore, it is harmless to extend the penalty functions to be not only valid on the immersed boundary but also valid in the area outside the domain ω (i.e., $\Omega \setminus \omega$).

Thus, the following weak formulation for the fictitious domain Ω can be obtained:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \frac{1}{\varepsilon} \int_{\Omega \setminus \omega} (v \cdot \tilde{u}) dx = \int_{\Omega} (v \cdot \tilde{f}) dx + \frac{1}{\varepsilon} \int_{\Omega \setminus \omega} (v \cdot g_1) dx, \quad (1-8)$$

where

$$\tilde{f}|_{\omega} = f.$$

Similar to equation (1-6), this equation has to hold for all test functions in H^1 space. As to the solution \tilde{u} gained from this linear system, it should be identical to the one from original Poisson problem (i.e., equation (1-3) and equation (1-4)) in the domain ω .

In addition, according to equation (1-8), we know that the penalty functions are discontinuous in the domain Ω , since they are only valid in the domain $\Omega \setminus \omega$. As a result, they can be viewed as discontinuous terms.

There are two kinds of penalty function: L_2 penalty function and H^1 penalty function. The penalty functions shown in equation (1-8) belong to the L_2 penalty function. The H^1 penalty function is derived from the gradient of solutions. In this example, the weak form after applying the H^1 penalty function can be expressed as:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \frac{1}{\varepsilon} \int_{\Omega \setminus \omega} (\nabla v \cdot \nabla \tilde{u}) dx = \int_{\Omega} (v \cdot \tilde{f}) dx + \frac{1}{\varepsilon} \int_{\Omega \setminus \omega} (\nabla v \cdot \nabla g_1) dx. \quad (1-9)$$

The capability of these two penalty functions in the FD method were studied in Zhou and Saito [12] and Saito and Zhou [13] by solving the Poisson problem with the homogeneous boundary condition. Although these two penalty approaches have different discontinuous terms, numerical experimental results showed that both of these methods could save computational cost while dealing with problems with complex geometries. They also approximately kept first order accuracy in the L_2 error norm (i.e., $e \sim O(h)$, where e represents the error norm obtained from computing the difference between the approximate solution and the exact solution).

The big advantage of using this method is that there is no need to use two discretization grids or modify the numerical scheme near the immersed interface.

Therefore, it is simple to apply. Application regarding this method can be seen in Ramière et al. [14] and Zhu and Ma [15].

However, there is one thing the user needs to notice if he/she wants to apply this method: the existence of the lower bound to error norms. This lower bound is formed due to the application of the penalty parameter. For example, according to Saito and Zhou [13], while using the L_2 penalty function for the Poisson problem, the L_2 norm of the error is not only related to the mesh size h , but also related to the square root of the penalty parameter ε . Therefore, after some point, even if the user keeps on refining the mesh, the error will not decrease since it is bounded by $\sqrt{\varepsilon}$. Although the influence from this lower bound can be reduced by decreasing the value of the penalty parameter, the tradeoff of this operation is that the system will become more unstable since it increases the abrupt function changes in the domain (Bryan and Shibberu [16]). This feature may confine the development of the penalty function method in the FD method. This is also the main reason why we do not apply this strategy in our research.

1.2.2 Lagrange multiplier

This may be the most popular approach for implementing the FD method. To realize how this method works, let us consider the constrained optimization problem listed in equation (1-1) and equation (1-2). By applying the Lagrange multiplier method, we can get the Lagrangian functional for the complex domain ω (see Figure 3(a)):

$$L(u, \lambda) = \frac{1}{2} \int_{\omega} (\nabla u \cdot \nabla u) dx - \int_{\omega} (f \cdot u) dx + \int_{\gamma} \lambda \cdot (u - g_1) dx \quad \text{in } \omega, \quad (1-10)$$

where λ represents the Lagrange multiplier.

Next, we embed the domain ω in the domain Ω and extend the Lagrangian functional (equation (1-10)) to be valid in the whole domain Ω (Figure 3(b)):

$$L(\tilde{u}, \lambda) = \frac{1}{2} \int_{\Omega} (\nabla \tilde{u} \cdot \nabla \tilde{u}) dx - \int_{\Omega} (\tilde{f} \cdot \tilde{u}) dx + \int_{\gamma} \lambda \cdot (\tilde{u} - \tilde{g}) dx \quad \text{in } \Omega. \quad (1-11)$$

In this equation, the Lagrange multiplier λ is valid on the immersed boundary γ , which is immersed in the mesh generated for the fictitious domain. In addition, $\tilde{f}|_{\omega} = f$ and $\tilde{g}|_{\gamma} = g_1$.

Next, by applying the variational principle, we have the following weak formulation:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \int_{\gamma} (v \cdot \lambda) d\gamma = \int_{\Omega} (v \cdot \tilde{f}) dx \quad \forall v \in H^1(\Omega), \quad (1-12)$$

$$\int_{\gamma} (\mu \cdot \tilde{u}) d\gamma = \int_{\gamma} (\mu \cdot \tilde{g}) d\gamma \quad \forall \mu \in L_2(\Omega), \quad (1-13)$$

whose differential form is:

$$-\Delta \tilde{u} = \tilde{f} - \lambda \cdot \delta|_{\gamma} \quad \text{in } \Omega, \quad (1-14)$$

$$\tilde{u} = \tilde{g} \quad \text{on } \gamma. \quad (1-15)$$

The solution \tilde{u} obtained from the weak formulation system (equation (1-12) and equation (1-13)) should be identical to the desired solution in the domain ω (i.e., $\tilde{u}|_{\omega} = u$).

Note: Here, we omitted some details regarding the derivation of the linear system. Detailed description about this derivation can be seen in Chapter 2

Similar to the penalty function method, while applying the Lagrange multiplier method to implement the FD method, users first need to deal with an important issue:

how to compose a suitable solution space for the Lagrange multiplier, which lives on the IB.

To address this issue, Glowinski et al. [9], proposed a strategy which involves seeking intersection points between the original irregular boundary and the mesh generated for the fictitious domain. The product of this process was a set of additional points, which composed the solution space of the Lagrange multiplier (refer to Figure 4). According to their experiment results, their strategy worked very well. It was capable of dealing with the Stoke flow problem with Dirichlet boundary condition. This strategy was later proved to be valid not only to the Stokes flow problem but also to other flow problems (Glowinski et al. [17, 18]).

This so-called Lagrange multiplier fictitious domain method (LM/FD) was further developed to become the distributed Lagrange multiplier fictitious domain (DLM/FD) method, which was capable to simulate the flow around moving bodies, whose motions had been known in advance (Glowinski et al. [19] and Glowinski et al. [20]). Thereafter, Yu [21] extended this method to be able to simulate the motion and the deformation of a nonlinear elastic body in a flowing field.

The most impressive feature of this strategy is that its accuracy is comparable to the one from body-fitted meshes. For example, to a Poisson problem, if the user discretize the solution with Q1 element, then, the L^2 norm of the error obtained from this strategy approximately behave as $O(h^2)$ (Glowinski et al. [9]).

As a result, some experts tried to propose new strategies to apply the Lagrange multiplier. Burman and Hansbo [22] proposed a stabilized Lagrange multiplier method, which involves using “cut elements.” Cut elements were elements passed through by the immersed boundary. The Lagrange multipliers were defined as being element-wise constants in these cut elements. In addition, the jump of the multiplier over element faces was penalized to satisfy the inf-sup condition. The data structure of this strategy is much simpler, compared to the one in Glowinski et al. [9]. Thereafter, they proposed another related alternative which used the Nitsche method to apply the constraint on the immersed boundary (Burman and Hansbo [23]).

The downside of this strategy is the appearance of the interior penalty term for stabilizing the scheme. To get this term, evaluating the jump of the Lagrange multiplier in cut elements is necessary which complicates the computation.

The extended finite element (XFEM) method is another alternative to impose the immersed boundary condition. Similar to previous strategies, this strategy also starts with a search process. However, it not only searches elements passed through by the immersed boundary, but also divides all of them into two groups: bad elements and good elements. The former one only intersects with the immersed boundary in very small portions. The shape function of a good element will be extended to its neighboring bad element (refer to Figure 5). The advantage of this treatment is that it reduces the instability that comes from these bad elements. As a result, the Lagrange multiplier and primal variables can all be defined on a single mesh. Experimental results in Halinger and Renard [24] showed that this strategy could solve the Poisson problems with mixed

boundary conditions. Further extensions regarding the applications of this strategy in the Stokes problem and fluid structure interaction problem can be found in Court et al. [25] and Court et al. [26].

The flaw of this strategy is its complex search process, which involves judging bad elements and good elements. In 2D cases, it is not very hard to implement this work. For example, in Figure 6, instead of computing the area within the red curve, the user can approximate it by the green triangle surrounded by the points A, B and C. However, in 3D cases, the intersection points between the immersed boundary and mesh may be uncertain. It is hard to do the same thing in this situation.

In our research, we choose the Lagrange multiplier method to address the immersed boundary condition. However, considering the difficulty and drawbacks of the previous strategies, we would like to propose new idea to compute the resulting boundary integrals.

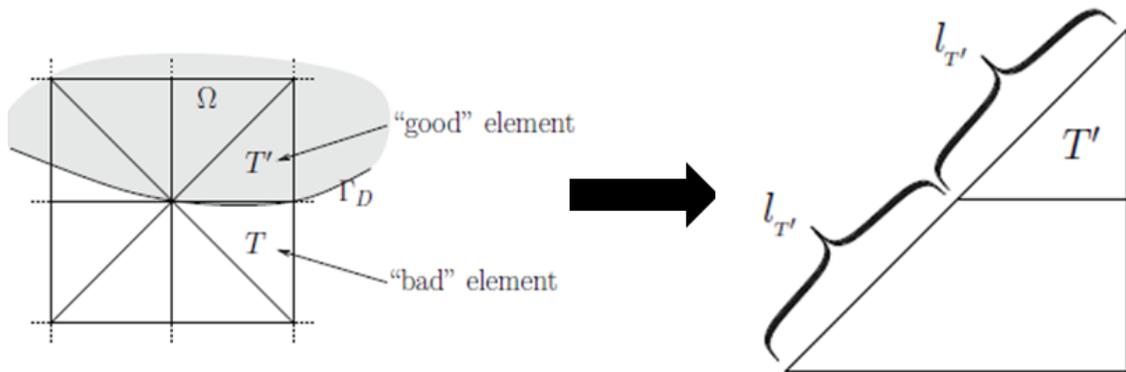


Figure 5 Concept of the XFEM in the FD method. T is a bad element which has small intersection with the immersed boundary Γ_D , T' is a good element neighboring T , shape function of T' will be extended. (refer to [24]).

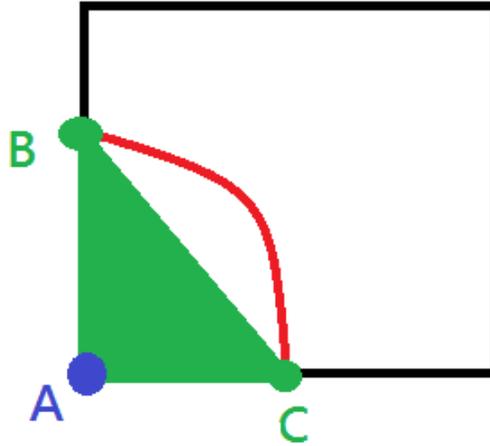


Figure 6 The approximate area for judging the bad and good element. Point A is a vertex of this cell, points B and C are intersection points between the immersed boundary (red curve line) and this mesh.

1.2.3 Immersed boundary method

The immersed boundary method (IB method) is a very popular method in fluid-structure interactions problems (FSI problem). While applying this method, interactions between the solid structure and the flowing fluid are expressed as the surface force exerts on the immersed boundary, which formulates the kinematic constraints. The Dirac delta function will be used to impose these constraints, which plays an identical role as the Lagrange multiplier in the LM/FD method. Therefore, some researchers view them as an identical method (Lai and Peskin [27], Peskin [28] and Jendoubi et al. [29]).

This method was shown to be valid in many applications, such as problems with a deformable body in the flow domain (Loon et al. [30]) or problems with a coolant passing around a high temperature circle (Mark et al. [31]).

As to the accuracy, the IB method is basically first order accurate, but the situation can be improved by applying some techniques, such as the RKPM (Wang and

Liu [32]). However, along with these additional treatments, the complexity of the programming also increases.

Here, we have to mention the reason why we do not choose the IB method in our research. Our concern results from the essence of the IB method. The invention of the IB method is for solving the FSI problem. As a result, there are at least two kinds of meshes in a computational domain, one is for the solid structure and the other is for the fluid. This situation exists even when the user only needs to solve a problem regarding a flow passing through a rigid, static body. To us, the generation of the mesh for the solid structure seems redundant. In addition, in some situation, in some situation, generating the mesh for the inner obstacle is as difficult as generating the body-fitted mesh for the fluid domain. Therefore, we do not choose this approach.

1.3 Parallel computing

When people started to do research in scientific computation, all accomplished codes could only be executed on a machine with one central processing unit (CPU). This kind of computation is called “serial computing.” The ability of serial computing is confined since the speed of the cpu and the available memory on a machine are limited. In order to address large scale engineering problems, study of parallel computing sprouted.

As a discipline, parallel computing is attractive due to following advantages: provides multiplicity of data paths, increases access to storage elements (memory and disk), scalable performance and lower cost (Grama et al. [33]). These features

significantly improve the efficiency of scientific computation. Nowadays, parallel computing is a necessary tool for large scale simulations.

In the remaining parts of this section, we first introduce main standards about communication in parallel computing, which is the most basic issue in this topic. After that, we discuss some research regarding the combination of the parallel computation and the FD method.

1.3.1 Standards regarding the parallelization

In parallel computing, programs and data reside on different processing elements (nodes or cores) which have to communicate with one another (Smith et al. [34]). Among numerous standards proposed for addressing it, Message Passing Interface (MPI) is the oldest one, which was developed by a group of researchers from academia and industry, including vendors. The objective of this standard is to establish a portable, efficient, and flexible standard, which can be used for writing message passing programs (Barney [35]). Since vendors are also its developers, MPI is available on all commercial parallel computers (Grama [33]). Usually, the MPI is applied to group many nodes (i.e., computers) and each node has its own memory and Central Processing Unit (CPU).

Open Multi-Processing (OpenMP) is another popular standard. It is an application programming interface (API) for programming shared-memory systems (Smith et al. [34]). The shared-memory system is a system coupling separate cores that access the same memory. Due to the appearance of multi-core processor, OpenMP can

be applied in a personal computer. A detailed description regarding this standard can be seen at <http://www.openmp.org/>.

When people started to develop MPI, each node/machine only had one processor with one core. However, over the past 10 years, multi-core processors have gradually replaced single core processors (Smith et al. [34]). Because of this improvement, while doing parallel computation on a super computer, experts now choose a hybrid strategy which applies MPI to pass messages among different nodes and OpenMP on each node.

In addition to MPI and OpenMP, which work on the main processor, scientists/engineers are also trying to do parallel computing on secondary processors, such as Graphics Processing Units (GPU). There are mainly two standards in this aspect: Open Computing Language (OpenCL) and Computer Unified Device Architecture (CUDA). The former one can be applied on any vendor's GPU. As to the latter one, it is invented by the technology company, NVIDIA, and executed on their products (Smith et al. [34]). Details regarding the implementation of these two can be seen at <http://www.khronos.org/opencvl> and www.openmp.org.

In addition to these four standards, there are some other standards, such as Pthreads. However, the studies concerning these standards are out of our scope.

Knowing these standards is very important to parallel computing, especially, when users try to apply external software packages. For example, the open source library, Trilinos only supports MPI (refer to <https://trilinos.org>). However, the other open source library, PETSc, supports MPI, OpenCL and CUDA (refer to <https://www.mcs.anl.gov/petsc>).

1.3.2 Combination of the FD method and the parallelization

The great advantage of the FD method is the use of very simple, structure meshes, which is very helpful for code parallelization since data load balance can be achieved easily under this circumstance.

Blasco et al. [36] developed a fictitious domain parallel numerical method for simulating the behaviors of rigid particles in an incompressible viscous Newtonian fluid. In their method, the standard they chose for parallelization was OpenMP. In addition, in order to enforce the rigidity of the particle, they modified the DLM/FD method of Glowinski et al. [19], by applying the fast computational techniques from Patankar and Sharma. Moreover, the parallel Simultaneous Directions Implicit method (SDI) was implemented to improve the numerical algorithms. Their results showed that their model is capable of addressing simpler problems, such as simulating two particles moving in a rectangle domain.

Thereafter, Wachs [37] developed a code named Parallel Efficient Library for Grains in Fluid Flow (PeliGRIFF). The purpose of this code is to simulate particulate flows by employing the DLM/FD method. In this code, MPI is used for parallel computing. Their results showed that this code has the capability to simulate a 2D case with 6,400 particles.

Here, we have to mention that in comparison with the literature regarding the FD method or parallel computing, research projects regarding the combination of these two were rare. However, this combination indeed has high potential for solving large scale engineering problems with complex geometries,

1.4 C++ library deal.II

In this research, the implementation was accomplished using the C++ library deal.II, which is an open source library to solve the partial differential equations via the finite element method (<http://www.dealii.org>). Unlike some other FEM libraries whose applicability is limited to some specific applications, due to the use of advanced object-oriented and data encapsulation techniques, deal.II has proven that it support many fields, such as mechanical engineering (Chueh et al. [38]), biology (Freyer et al. [39]) and geophysics (Kronbichler et al. [40]). In addition, deal.II provides friendly interfaces to extensive open source codes, such as UMFPACK (<http://faculty.cse.tamu.edu/davis/suitesparse.html>), BLAS, PETSc and Trilinos. Users can benefit from this while solving a linear system or doing large scale computation. More details regarding the capabilities of deal.II can be seen in Bangerth et al. [41].

1.5 Outline of this dissertation

The motivation of this research was to find some alternatives to solve large scale flow problems without causing too much burdens to the user and the computer. Our choice was the FD method. The Lagrange multiplier method was used to include the influence from the immersed boundary condition, the critical issue regarding this method. However, considered the drawbacks of all existing strategies, we proposed two new strategies for the implementation. The idea of these two strategies is to replace all boundary integrals in the resulting weak formulation by domain integrals that all computation can be accomplished by using the mesh generated for the fictitious domain.

Therefore, the objective of this research was to see whether these two strategies are able to fulfill our need.

In addition, instead of directly addressing the Navier-Stokes system by using the FD method, which needs to address too many difficulties at the same time, we solved them gradually by studying three different types of problems. The study of these three problems composed the main parts of this research. The advantage of this idea is that we only need to focus on limited things at a time.

In the second chapter, we present and test our strategies for the immersed boundary condition by studying a Poisson problem.

Next, based on the outcomes of the second chapter, we solve a steady Stokes flow problem in Chapter 3. In addition to investigating whether our strategies work well in problems with multiple variables, in this chapter, we propose a modified iterative algorithm which can address the desired problem.

The topic of the fourth chapter is large scale Navier-Stokes system. In this chapter, we first explain how we parallelize our code for solving the steady Stokes flow problem by applying the open source software libraries Trilinos. Next, we accomplish the code for solving the steady Navier-Stokes system by including the influence from the nonlinear advection term. Accuracy and capability of this implementation are evaluated by studying multiple cases.

Finally, we give our conclusion in the fifth chapter. Possible extension of our FD method is mentioned as well.

CHAPTER II

POISSON PROBLEM

2.1 Introduction

In this chapter, the fictitious domain method is used to solve a Poisson problem with Dirichlet/essential boundary conditions. The governing equation of this problem belongs to the class of elliptic partial differential equations (PDE). The feature of this type of PDE is that information propagates at infinite speed in all direction. Related applications can be seen in elastic bar problems or heat conduction problems.

In the following contents of this chapter, we first give an overview of the designed problem in Chapter 2.2. Then, we explain our strategies for imposing the immersed boundary condition in Chapter 2.3. Numerical details, such as the finite element classes of the approximate solutions and the solver for the resulting linear system, are described in Chapter 2.4. All obtained experiment results are demonstrated in Chapter 2.5. Final conclusion and discussion regarding this topic are shown in Chapter 2.6.

2.2 Test case setting

Code verification plays an important role in numerical simulation. Comparing numerical results to experimental data is one way to achieve this goal. However, it is not practical in some situations. As a results, some benchmarks were proposed, such as Kovasznay flow (Kovasznay [42]) and backward-facing step flow (Biswas et al. [43]).

Although these benchmarks are very useful, most of them have some limitations. For example, Kovasznay flow cannot be used to problems with curved boundaries. Therefore, the Method of Manufactured Solutions (MMS) was invented.

The standard procedure of the MMS is that the user first assigns an exact solution. This solution will be substituted in the governing equation of the problem to get the corresponding right hand side terms. This right hand side terms will be implemented in the code. Since the assigned solution can be treated as an exact solution to get correct values at each point within the domain, error estimations become easier. In addition, assigning a solution is not hard since it does not need to be realistic. More details regarding this method can be seen in Roache [44]. In our research, this method is extensively applied when we design our test cases.

Now, we go back to consider our designed test case of this chapter whose domain, ω , is a 2D circle. The boundary of this domain is denoted as γ (see Figure 7). What we want to find is the solution which satisfies equation (1-3) and equation (1-4):

$$\begin{aligned} -\Delta u &= f && \text{in } \omega, \\ u &= g_1 && \text{on } \gamma, \end{aligned}$$

Suitable function spaces for the right hand side terms of these two equations are: $f \in H^{-1}(\omega)$ and $g_1 \in H^{1/2}(\gamma)$. In addition, g_1 represents the Dirichlet boundary condition.

The solution u of this system is also the solution of the following variational problem: Find $u \in V_g$, which satisfies

$$a_\omega(v, u) = \langle v, f \rangle \quad \forall v \in H_0^1(\omega), \quad (2-1)$$

where

$$a_\omega(v, u) = \int_\omega (\nabla v \cdot \nabla u) dx \quad \forall v \in H_0^1(\omega),$$

$$\langle v, f \rangle = \int_\omega (v \cdot f) dx.$$

Here, V_g is the solution space and its definition is: $V_g = \{v \mid v \in H^1(\omega), v = g \text{ on } \gamma\}$.

With the concept of MMS, we first set the exact solution of this test case as:

$$u = \frac{dis^2}{r_i^2} \times g_1 \quad \text{in } \omega, \quad (2-2)$$

where r_i is the radius of the circle and dis represents the distance between any point within ω and the reference point, the latter one represents the center of the circle in this test case. Next, by substituting equation (2-2) in equation (1-3), we can get f :

$$f = -\frac{4}{r_i^2} \times g_1 \quad \text{in } \omega. \quad (2-3)$$

Here, for simplicity, we set: (1) position of the reference point = (0,0); (2) $r_i = 0.6$; (3) $g_1 = 6$.

Note: The domain ω as some other shape with more irregular boundary. The constraint g_1 can also be replaced by a complicated function. Settings here are decided just for simplicity.



Figure 7 Flow domain for the Poisson problem.

2.3 Fictitious domain method implementation

In this section, we first formulate the fictitious domain method and its corresponding linear PDE. Then, we describe how we address the constraint on the immersed boundary, which is the key point of this chapter.

2.3.1 Fictitious domain formulation

The embedding domain for this problem is a 2D square, which is denoted as $\Omega := (-1,1) \times (-1,1)$. The original domain ω is now embedded in the center of the square (Figure 8(b)). After this operation, the mesh we are going to generate in this computation looks like the one shown in Figure 9.

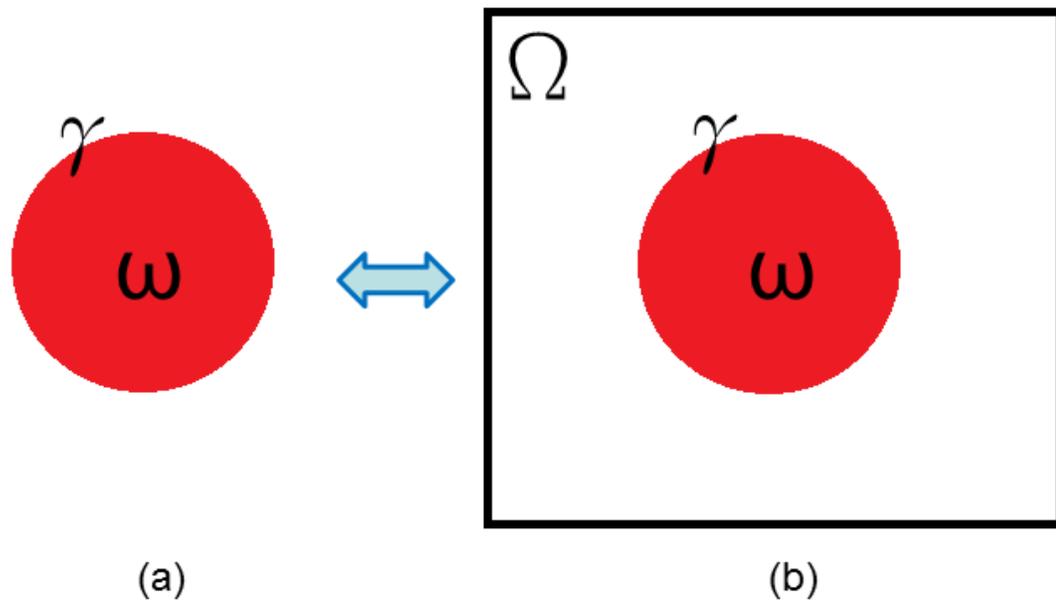


Figure 8 (a) The original domain ω and its boundary γ , (b) Ω is the fictitious domain for the Poisson problem.

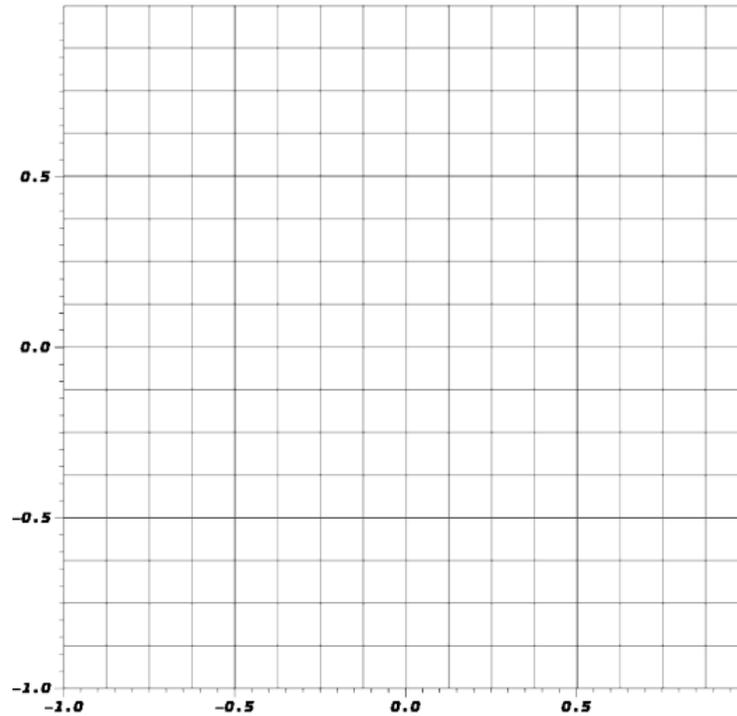


Figure 9 Mesh generated for the fictitious domain, Ω .

In order to place the corresponding equation on this fictitious domain, it is necessary to introduce the concept of the operator equation.

An abstract model for many problems in science and engineering can be written as the form of an operation equation:

$$Au = f, \tag{2-4}$$

where A is a linear or nonlinear operator. To the proceeding problem, the operator A equals to $-\Delta$, where Δ is Laplace operator.

With this concept, we can express the corresponding quadratic form as:

$$Q(u) \equiv \frac{1}{2}(Au, u) - (f, u) \tag{2-5}$$

and (\cdot, \cdot) represents an inner product.

According to the Minimum Functional Theorem (Reddy [11]), if w_0 is the solution of equation (1-3), it is also the one that minimize the quadratic functional, $Q(w)$ (i.e., $Q(w) \geq Q(w_0)$, for all w).

However, in addition to minimizing the quadratic function, the solution we are seeking also has to satisfy the boundary condition (equation (1-4)). Here we choose the Lagrange multiplier method to include the influence from this constraint. An introduction regarding the Lagrange multiplier method has been provided in Chapter 1.2.2.

With this concept, the proceeding Poisson problem becomes a saddle point problem: Find the solution pair (u_0, λ_0) that is a stationary point for the Lagrangian functional for ω :

$$L(u, \lambda) = \frac{1}{2} \int_{\omega} (\nabla u \cdot \nabla u) dx - \int_{\omega} (f \cdot u) dx + \int_{\gamma} \lambda \cdot (u - g_1) dx \quad \text{in } \omega.$$

In this equation, the first two terms on the right hand side represent the quadratic form of the Poisson equation (equation (1-3)). The last term is from the constraint of this problem.

Thus, we can get the Lagrangian functional for the fictitious domain Ω :

$$L(\tilde{u}, \lambda) = \frac{1}{2} \int_{\Omega} (\nabla \tilde{u} \cdot \nabla \tilde{u}) dx - \int_{\Omega} (\tilde{f} \cdot \tilde{u}) dx + \int_{\gamma} \lambda \cdot (\tilde{u} - \tilde{g}_1) dx \quad \text{in } \Omega.$$

where solution \tilde{u} is valid in the whole domain Ω and the Lagrange multiplier λ is valid on the boundary γ . In addition, in order to have $\tilde{u}|_{\omega} = u$ functions f and g need to be: $\tilde{f}|_{\omega} = f$ and $\tilde{g}_1|_{\gamma} = g_1$.

As mentioned above, the solution pair (\tilde{u}, λ) we are seeking is a stationary point of the Lagrangian functional (i.e., a saddle point). In order to get this point, we apply the variational principle to the Lagrangian functional:

$$\left. \frac{d}{dt} \right|_{t=0} L(\tilde{u} + tv, \lambda) = 0 \quad \text{for all } v, \mu,$$

$$\left. \frac{d}{dt} \right|_{t=0} L(\tilde{u}, \lambda + t\mu) = 0 \quad \text{for all } v, \mu.$$

Expanding these equations, we can get the weak formulation for the desired fictitious domain problem:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx - \int_{\Omega} (v \cdot \tilde{f}) dx + \int_{\gamma} (v \cdot \lambda) d\gamma = 0 \quad \text{in } \Omega, \quad (2-6)$$

$$\int_{\gamma} \mu \cdot (\tilde{u} - \tilde{g}_1) d\gamma = 0 \quad \text{on } \gamma, \quad (2-7)$$

whose differential form can also be written as:

$$-\Delta \tilde{u} = \tilde{f} - \lambda \cdot \delta|_{\gamma} \quad \text{in } \Omega, \quad (2-8)$$

$$\tilde{u} = \tilde{g}_1 \quad \text{on } \gamma, \quad (2-9)$$

where δ is the Dirac delta function which is only valid on γ .

In Chapter 1.2.2, we mentioned that in the weak formulation for the fictitious domain, the term related to the original irregular boundary γ (i.e., $\int_{\gamma} (v \cdot \lambda) d\gamma$) cannot be computed directly since γ becomes immersed in the mesh generated for the fictitious domain (see Figure 9). This is also the reason why people proposed some strategies to compute these boundary integrals, such as using intersection points or applying some regularization (see [9, 17, 18, 22-26]).

Different from other strategies, we would like to reduce the difficulty of this issue as much as possible, not only to the user, but also to computers. In the next portion, we will explain our strategies for addressing this boundary issue.

2.3.2 IB condition treatment: Strategy 1

In Strategy 1, we plan to divide the whole domain into two parts: ω and $\Omega \setminus \omega$, then, we extend the original boundary constraint g to be valid in the artificial area $\Omega \setminus \omega$. Since the area $\Omega \setminus \omega$ includes the immersed boundary γ , this idea automatically satisfies the constraint (equation (2-7) or equation (2-9)).

Thus, the weak formulation (equation (2-6) and equation (2-7)) can be adjusted as:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \int_{\Omega \setminus \omega} (v \cdot \lambda) dx = \int_{\Omega} (v \cdot \tilde{f}) dx \quad \text{in } \Omega,$$

$$\int_{\Omega \setminus \omega} (\mu \cdot \tilde{u}) dx = \int_{\Omega \setminus \omega} (v \cdot \tilde{g}_1) dx \quad \text{in } \Omega \setminus \omega$$

Next, by introducing the weight factor $w(x)$, we can get a new weak formulation for the fictitious domain:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \int_{\Omega} (w_1(x) \cdot v \cdot \lambda) dx = \int_{\Omega} (v \cdot \tilde{f}) dx, \quad (2-10)$$

$$\int_{\Omega} (w_1(x) \cdot \mu \cdot \tilde{u}) dx + \int_{\Omega} ((1 - w_1(x)) \cdot \mu \cdot \lambda) dx$$

$$= \int_{\Omega} (w_1(x) \cdot \mu \cdot \tilde{g}_1) dx + \int_{\Omega} ((1 - w_1(x)) \cdot \mu \cdot 0) dx, \quad (2-11)$$

where

$$w_1(x) = \begin{cases} 1, & x \in \Omega \setminus \omega \\ 0, & x \in \omega \end{cases},$$

$$\widetilde{g}_1|_{\Omega \setminus \omega} = g_1 \text{ and } \widetilde{g}_1|_{\omega} = 0.0,$$

$$\widetilde{f}|_{\omega} = f \text{ and } \widetilde{f}|_{\Omega \setminus \omega} = 0.0.$$

The big feature of system composed of equation (2-10) and equation (2-11) is that all terms are integrals over the fictitious domain Ω . Therefore, it is not necessary to have more than two types of mesh; all computation can be obtained directly without applying any additional changes in the region near the immersed boundary.

2.3.3 IB condition treatment: Strategy 2

The second strategy to deal with the boundary terms is to conceptually set a boundary region $\bar{\gamma}$, which contains the original boundary γ and its neighboring area. Then, we use this region to form the solution space for the multiplier. However, it is not enough to just replace $\int_{\gamma} dx$ with $\int_{\bar{\gamma}} dx$, since this action cannot correctly evaluate the influence from the immersed boundary condition. To validate this replacement, we need an additional weight function $k(x)$, which satisfies:

$$\int_{\gamma} s(x) dx \approx \int_{\bar{\gamma}} k(x) \cdot s(x) dx. \quad (2-12)$$

In this equation, the function $s(x)$ represents the original constraint which is only valid on the immersed boundary. Accordingly, we have the following expressions:

$$\int_{\bar{\gamma}} k(x) dx \approx \int_{\gamma} 1 dx. \quad (2-13)$$

A similar idea can be seen in Buffet and Penven [45].

Thereafter, by introducing the function $k(x)$, we can rewrite the original system (equation (2-6) and equation (2-7)) as:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \int_{\bar{\gamma}} (k(x) \cdot v \cdot \lambda) dx = \int_{\Omega} (v \cdot \tilde{f}) dx \quad \text{in } \Omega,$$

$$\int_{\bar{\gamma}} (k(x) \cdot \mu \cdot \tilde{u}) dx = \int_{\bar{\gamma}} (k(x) \cdot \mu \cdot \tilde{g}_1) dx \quad \text{in } \bar{\gamma}$$

where $\tilde{g}_1|_{\bar{\gamma}} = g_1$ and $\tilde{g}_1|_{\Omega \setminus \bar{\gamma}} = 0.0$. In addition, we set $\tilde{f}|_{\bar{\omega}} = f$ and $\tilde{f}|_{\Omega \setminus \bar{\omega} \setminus \bar{\gamma}} = 0.0$.

Finally, as in Strategy 1, we introduce the weight factor $w(x)$ and get the corresponding system:

$$\int_{\Omega} (\nabla v \cdot \nabla \tilde{u}) dx + \int_{\Omega} (w_2(x) \cdot k(x) \cdot v \cdot \lambda) dx = \int_{\Omega} (v \cdot \tilde{f}) dx, \quad (2-14)$$

$$\begin{aligned} & \int_{\Omega} (w_2(x) \cdot k(x) \cdot \mu \cdot \tilde{u}) dx + \int_{\Omega} ((1 - w_2(x)) \cdot \mu \cdot \lambda) dx \\ & = \int_{\Omega} (w_2(x) \cdot k(x) \cdot \mu \cdot \tilde{g}_1) dx + \int_{\Omega} ((1 - w_2(x)) \cdot \mu \cdot 0) dx, \end{aligned} \quad (2-15)$$

where the weight factor $w_2(x)$ is now equal to one in $\bar{\gamma}$ and equal to zero in the remaining areas.

Although all terms in this system look solvable, we still cannot compute the system directly, since we have not decided the size of the boundary region and the function $k(x)$.

Due to the setting of the boundary region $\bar{\gamma}$, there will be three conceptual divisions in computation:

1. $\bar{\omega} := \{x \in \omega: \text{dist}(x, \gamma) > hf\}$,
2. $\bar{\gamma} := \{x \in \Omega: \text{dist}(x, \gamma) \leq hf\}$,

$$3. \quad \Omega \setminus \bar{\omega} \setminus \bar{\gamma} := \{x \in \Omega \setminus \omega : \text{dist}(x, \gamma) > hf\}.$$

Here, the function $\text{dist}(x, \gamma)$ is the shortest distance measured from a chosen quadrature point to the immersed boundary γ . In this problem, we first computed the distance between the quadrature point and the center of the obstacle (black circle), $d(x)$, then, subtracted it by the radius of the circle, r_i . Parameter hf represents the half width of the boundary region. Therefore, the radii of the inner green circle and outer green circle could be expressed as $r_i = r_i \mp hf$ (refer to Figure 10). According to Figure 10, we knew that the domain $\bar{\omega}$ is smaller than the needed domain ω .

Here, we have to mention that the domain decomposition listed above is conceptual and only used in computation. In particular, we still need to include the influence in the region between the inner green circle and the black circle when we process the error computation.

The crucial issue regarding the size of the boundary region lies in the definition of the parameter hf . We think the suitable choice for this parameter is to connect it to the mesh size h . Thus, it can be expressed as:

$$hf = c \times h, \tag{2-16}$$

where the parameter c is an arbitrary constant decided by the user. In this chapter, it varies from 0.25 to 3.0.

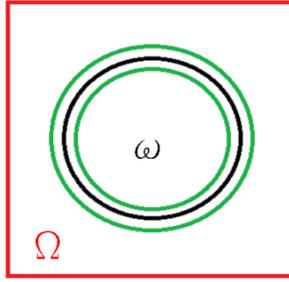


Figure 10 Boundary region $\bar{\gamma}$ for the Poisson problem.

After deciding the size of the boundary region, the next thing we need to do is to decide the $k(x)$ function. Theoretically speaking, there are countless options regarding this function. Among these options, we chose three functions in our implementation, which were constant function, triangle function and Gaussian function. They will be introduced in the following content.

The shape of the constant function looks like the one in Figure 11(a). After considering the basic requirement (i.e., equation (2-13), this function can be expressed as:

$$k(x) = A = \frac{1}{2 \times hf} \quad \text{in } \bar{\gamma}, \quad (2-17)$$

The second $k(x)$ is a triangular function, which can be seen in Figure 11(b). The peak value of it (i.e., B in Figure 11(b)) will appear at the center, where is coincidentally the position of the immersed boundary. The expression of this triangle function is:

$$k(x) = \left[1 - \frac{|d(x) - r_i|}{hf} \right] \times \frac{1}{hf} \quad \text{in } \bar{\gamma}, \quad (2-18)$$

where $d(x)$ is now the distance between a chosen quadrature point and the center of the circle ω . Comparing to the first assignment, this function should be more stable, since the triangle function is a continuous function.

Gaussian function is our third option, whose shape is a bell curve. Similar to the triangle function, Gaussian function is symmetric with respect to the center line (the red line in the Figure 11(c)). A reasonable choice regarding the position of this center line should be on the immersed boundary γ . Therefore, we have:

$$k(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(d(x)-r_i)^2}{2\sigma^2}\right) \quad \text{in } \bar{\gamma}, \quad (2-19)$$

where the parameter σ represents the standard deviation. In theory, in order to have the integral of this equation equal to one in $\bar{\gamma}$, we need to set σ much smaller than the width of the boundary region. Here, we connect it with the parameter hf :

$$\sigma = j \times hf. \quad (2-20)$$

The parameter j appearing in this equation is a positive number. In our implementation, we choose $j = \frac{1}{3}$, thus, the boundary region $\bar{\gamma}$ includes six standard deviations (i.e., the region between two green lines in Figure 11(c)). The advantage of this choice is that with this setting, the integral of the Gaussian function includes 99.7% of the area under it.

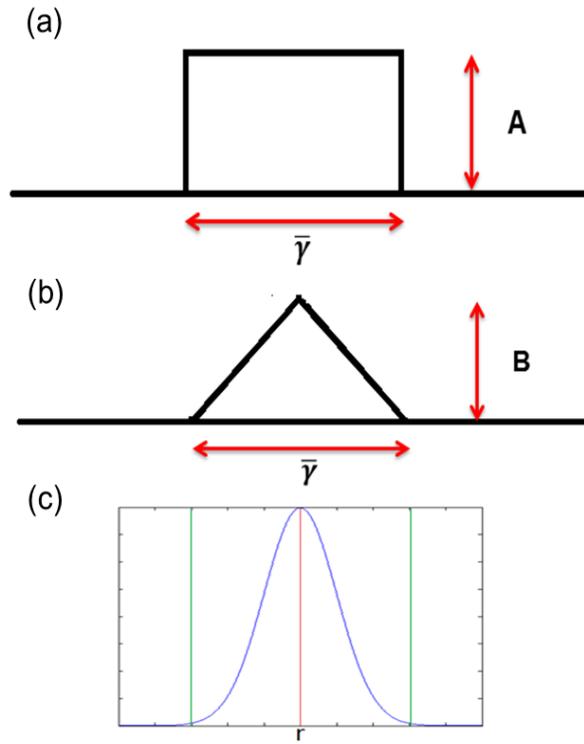


Figure 11 Conceptual plot regarding the Strategy 2, (a) Constant function, (b) Triangle function, (c) Gaussian function.

2.4 Numerical details

So far, we have described the test case we are going to solve in this chapter and its corresponding PDE after applying the FD method. In addition, we also described our strategies for immersed boundary condition implementation. In this section, we are going to talk about some numerical details regarding our approximate solutions.

First, we would like to define our finite element space. While using the FEM, the physical domain will be divided into several smaller elements, which can be denoted as K . Here, the default K in deal.II for 2D cases is quadrilateral. Next, we define a triangulation, which consists of these elements as T_h (i.e., $K \in T_h$). With these definitions, to our test problem, we can simply set the desired approximate solutions as:

$$u_h, \lambda_h \in Q_1(K),$$

where $Q_1 = \{p; p(x) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2, x \in K, \text{ where } a_i \in R\}$. This setting implies that there are four nodes for storing the approximate solutions in each cell.

Second, we would like to discuss the method for the numerical integration. In the finite element scheme, the matrix system we are going to solve is gained from the weak formulations. Therefore, how to deal with the integration is critical. In our implementation, Gaussian Quadrature rule is applied for this job. Instead of doing integration directly, this rule uses weighted sum of function values at specified points to get the solution (Sauer [46]). For example, if we want to get an approximate integration value of a function $f(x)$ in a 1-D domain, we instead evaluate the following equation:

$$\int_{-1}^1 f(x)dx \approx \sum_{i=1}^n C_i f(x_i), \quad (2-21)$$

where C_i are constants and can be gotten from Sauer [46] or many other books regarding numerical analysis. Parameter n is the number of quadrature points and x_i is the position of the quadrature point i .

In our code, we set the number of quadrature points to be equal to 3 in one direction, Thus, to this 2-D domain problem, we need to consider 9 quadrature points in each cell while doing numerical integration. These quadrature points are also used to evaluate the solution u , distance function $d(x)$ and weight function $w(x)$.

Since the objective of this chapter is to test our strategy for the immersed boundary condition, we do not want to put too much work on how to solve the saddle point problem composed by the variables, u and λ . Therefore, we use the sparse direct solver UMFPACK. UMFPACK is a set of routines for solving sparse linear systems,

$Ax=b$, using the Unsymmetric MultiFunctional method. A detailed description can be seen in Davis [47]. In deal.II, this solver can be accessed by applying the interface class `SparseDirectUMFPACK`.

The accuracy of our implementation is evaluated by computing the L_2 norm of the error and H^1 semi-norm norm of the error, which can be expressed as:

$$\|u - u_h\|_{L_2} = (\int |u - u_h|^2 dx)^{1/2} \quad \text{in } \omega, \quad (2-22)$$

$$|u - u_h|_{H^1} = (\int |\nabla(u - u_h)|^2 dx)^{1/2} \quad \text{in } \omega, \quad (2-23)$$

where u_h is the approximate solution obtained from our implementation and u can be obtained from equation (2-2)

2.5 Experimental results

In this section, numerical results obtained from our experiments are presented. Since we have introduced two strategies to impose the immersed boundary condition, we divide this section into two subsections. In each subsections, the solution profiles are demonstrated first. In order to examine whether these solution profiles are reasonable, we also show the result gained from the direct simulation, which is computed by using a body-fitted mesh. After that, we evaluate the L_2 norm of the error and the H^1 semi-norm of the error.

2.5.1 Results from Strategy 1

In Strategy 1, we let the constraint live outside the needed domain ω .

2.5.1.1 Solution profile from Strategy 1

In Figure 12, we demonstrate the meshes and solution profiles obtained from the direct simulation and Strategy 1. Here, we have to mention that what we are concerned with is the behavior in the inner circle (i.e., the original domain ω). According to Figure 12(d), we can see that the result from Strategy 1 only has variations in the inner area and this distribution is similar to the one from the direct simulation (Fig. 12(b)).

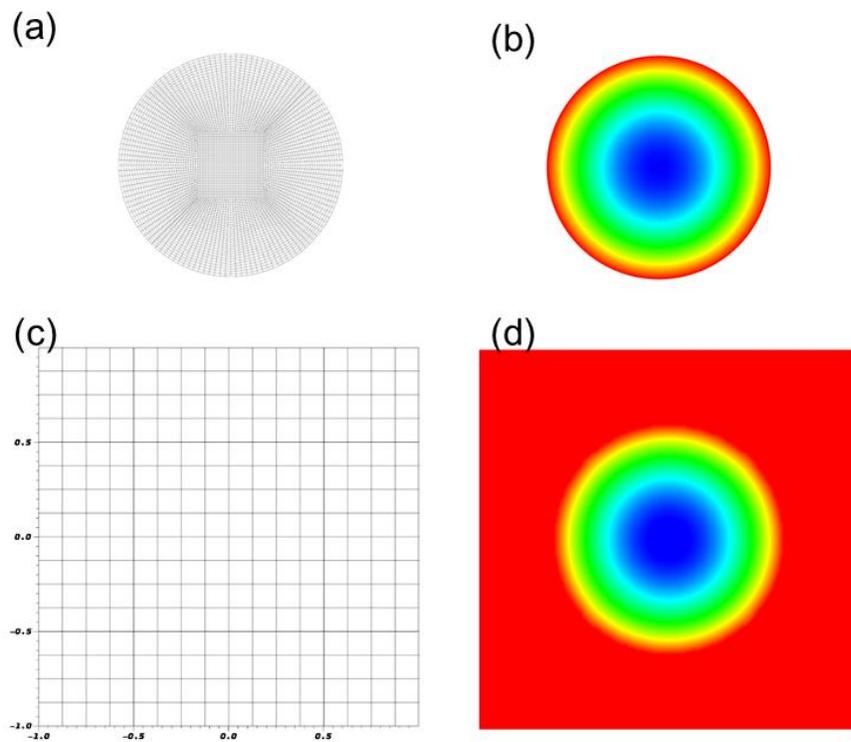


Figure 12 Mesh and obtained solution distribution of the Poisson problem obtained from the direct result and Strategy 1, (a) and (c) are meshes used in the direct simulation and Strategy 1, (b) and (d) are their corresponding results.

2.5.1.2 Error analysis of Strategy 1

The errors of this strategy are listed in Table 1.

Mesh size h	L_2 norm of the error	H^1 semi-norm of the error
$1/16$	0.994	1.15845
$1/32$	0.55701	0.73665
$1/64$	0.31595	0.42649
$1/128$	0.16292	0.23874
$1/256$	0.0859	0.17983
$1/512$	0.04279	0.14631

Table 1 Results of the Poisson problem obtained from Strategy 1.

According to this table, we draw Figure 13. For comparison, several reference lines are drawn, which are stated from the first data points. It is shown that the obtained L_2 norm of the error behaves linearly and has approximately first order accuracy, $O(h)$, which is comparable to the one from the use of the penalty function method. As to the H^1 semi-norm, it locates in the area between $O(h)$ and $O(h^{0.5})$.

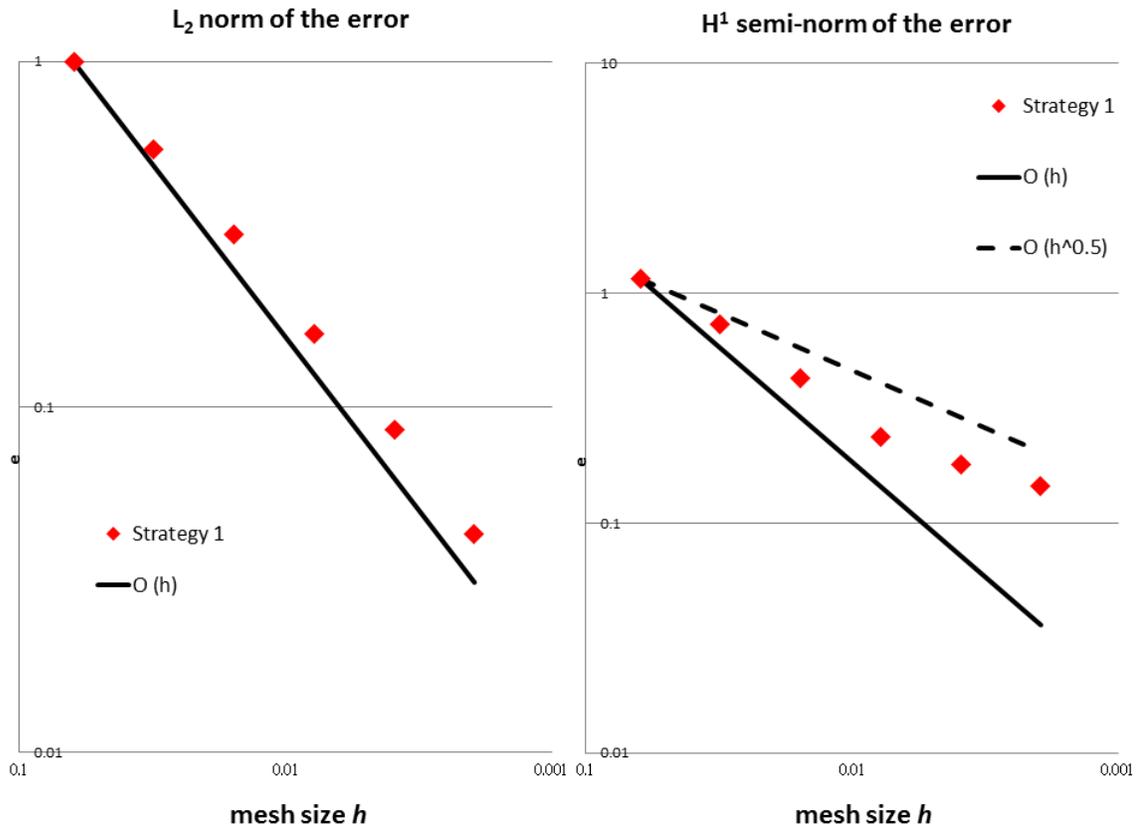


Figure 13 L₂ norm of the error and H¹ norm of the error of the Poisson problem obtained from Strategy 1.

2.5.2 Results from Strategy 2

In this strategy, we conceptually set a new boundary region $\bar{\gamma}$ to replace the original boundary γ . To validate this replacement, we need to add a $k(x)$ function which has to satisfy equation (2-13). In our experiment, three different functions are chosen and examined for this need, see Chapter 2.3.3. In addition, in order to test the influence from the width of the boundary region $\bar{\gamma}$, five different numbers regarding the constant c are also tested in our experiments (refer to equation (2-16)).

2.5.2.1 Solution profile from Strategy 2

Solution profile gained from Strategy 2 is shown in Figure 14. Likewise, we draw the result from the direct simulation for comparison. According to this figure, we can say that Strategy 2 can also get acceptable result in quality.

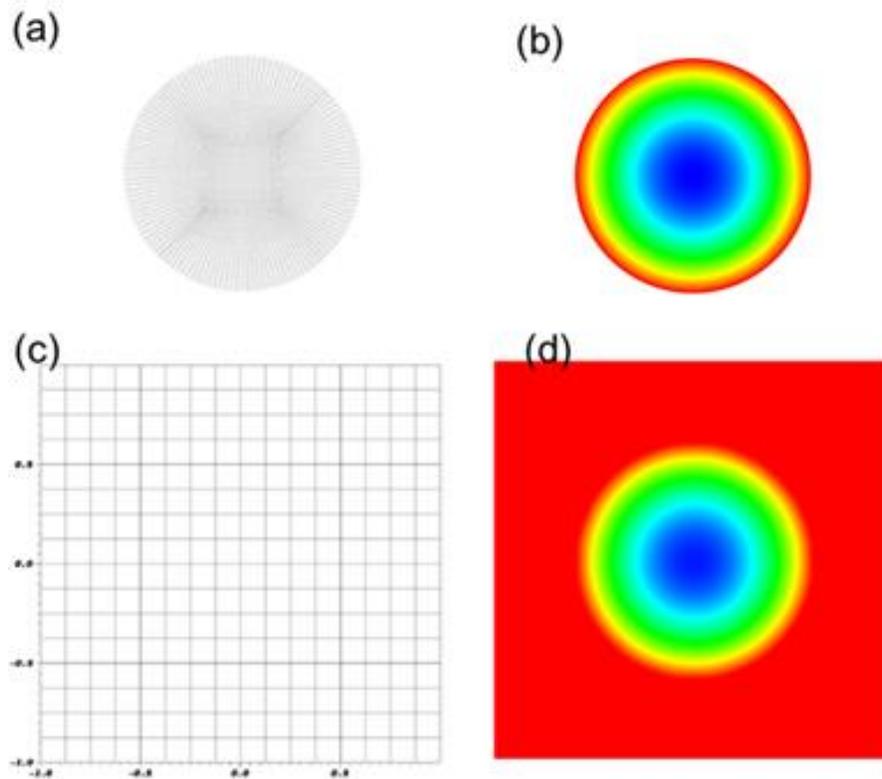


Figure 14 Mesh and obtained solution distribution of the Poisson problem obtained from the direct result and Strategy 2 (with constant function), (b) and (d) are their corresponding results.

2.5.2.2 Error analysis of Strategy 2

The first result we are going to demonstrate comes from the setting with $k(x)$ function as a constant function, whose shape and expression can be seen in Figure 11(a) and equation (2-17).

Table 2 shows the computed errors from the constant function.

L ₂ norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	0.56282	0.08733	0.62912	1.59754	2.33272
$1/32$	0.21926	0.0393	0.29205	0.87915	1.37365
$1/64$	0.09678	0.01857	0.15812	0.46038	0.73857
$1/128$	0.05886	0.00402	0.07789	0.22827	0.37762
$1/256$	0.03146	0.00216	0.03816	0.11539	0.1929
$1/512$	0.014	0.00096	0.01915	0.05865	0.0971
H ¹ semi-norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	3.27456	3.35045	5.3476	10.1537	12.1778
$1/32$	2.14259	2.18105	3.38812	7.68663	9.54891
$1/64$	1.53026	1.60983	2.53119	5.58738	7.03417
$1/128$	1.05158	1.04366	1.83221	3.94175	5.06716
$1/256$	0.77615	0.76309	1.26549	2.8037	3.63103
$1/512$	0.53379	0.52648	0.88783	2.00805	2.57778

Table 2 Results of the Poisson problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.

Accordingly, we can draw Figure 15. In the subplot regarding the obtained L₂ norm of the error, it is shown that when the width of the boundary region is large enough (in this case, it means $c \geq 1.0$), the corresponding data point distributions behave linearly and has approximately first order accuracy. However, even though the data point distribution from $c = 0.5$ does not behave linearly, it has the smallest errors. According to the two reference lines in this subplot, we know that the L₂ norm of the error from is

clearly better than first order accuracy. If we estimate its accuracy based on its first data point and the last data point, we can express the L_2 norm of the error as: $e \sim O(h^{1.3})$.

As to the subplot of the obtained H^1 semi-norm of the error, almost all data point distributions behave linearly and can be approximately expressed as $e \sim O(h^{0.5})$.

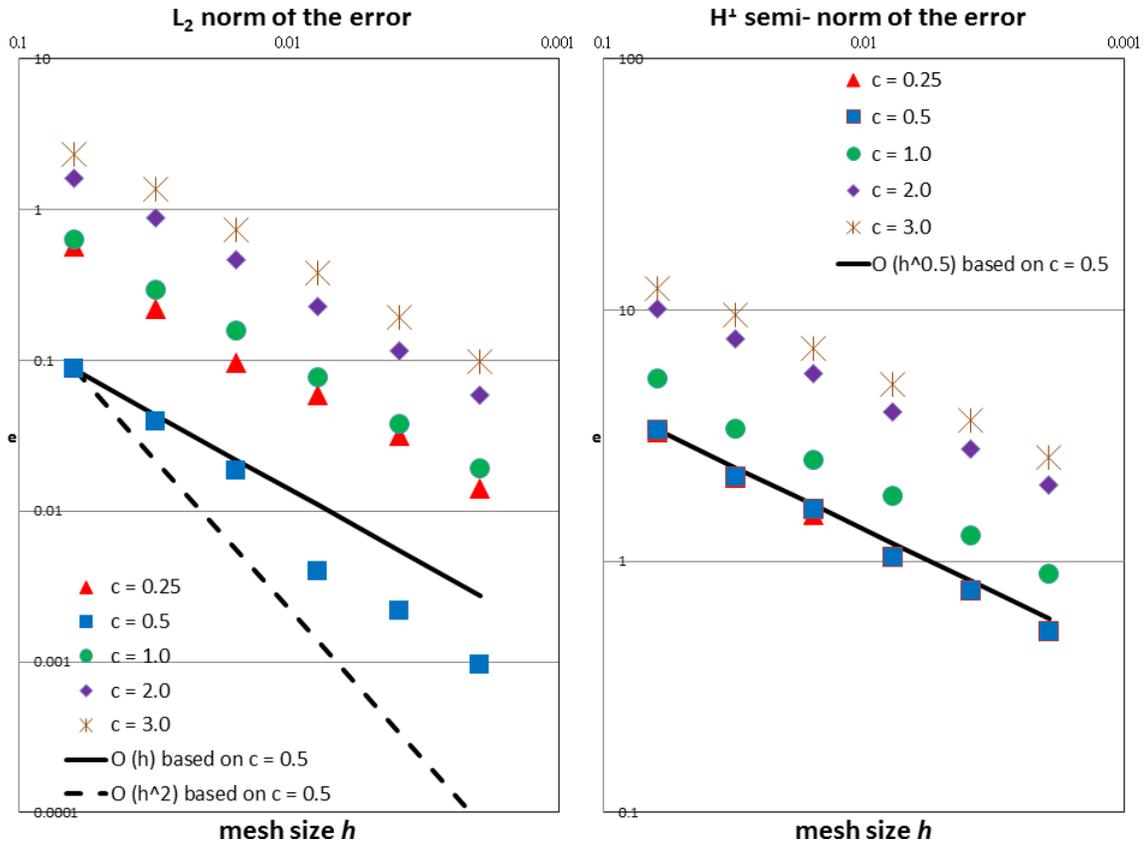


Figure 15 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.

The second $k(x)$ function we tested is a triangle function. Its shape and expression has been introduced in Figure 11(b) and equation (2-18). Obtained results from this function are listed in Table 3 and drawn in Figure 16

L ₂ norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	0.38736	0.07771	0.48416	1.42083	2.18398
$1/32$	0.17721	0.02653	0.21365	0.76089	1.24991
$1/64$	0.11558	0.00881	0.11292	0.39188	0.65389
$1/128$	0.06205	0.00371	0.05439	0.19384	0.3368
$1/256$	0.02724	0.00236	0.02668	0.09722	0.17052
$1/512$	0.01319	0.00055	0.01335	0.04948	0.08592
H ¹ semi-norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	3.1604	3.24853	4.58029	9.44169	11.7951
$1/32$	2.14332	2.1042	2.83314	7.08838	9.06271
$1/64$	1.57067	1.51153	2.10857	5.09848	6.59699
$1/128$	1.05344	1.00348	1.49199	3.61504	4.79562
$1/256$	0.76512	0.733	1.03668	2.56201	3.41514
$1/512$	0.53179	0.50814	0.72908	1.8334	2.42848

Table 3 Results of the Poisson problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.

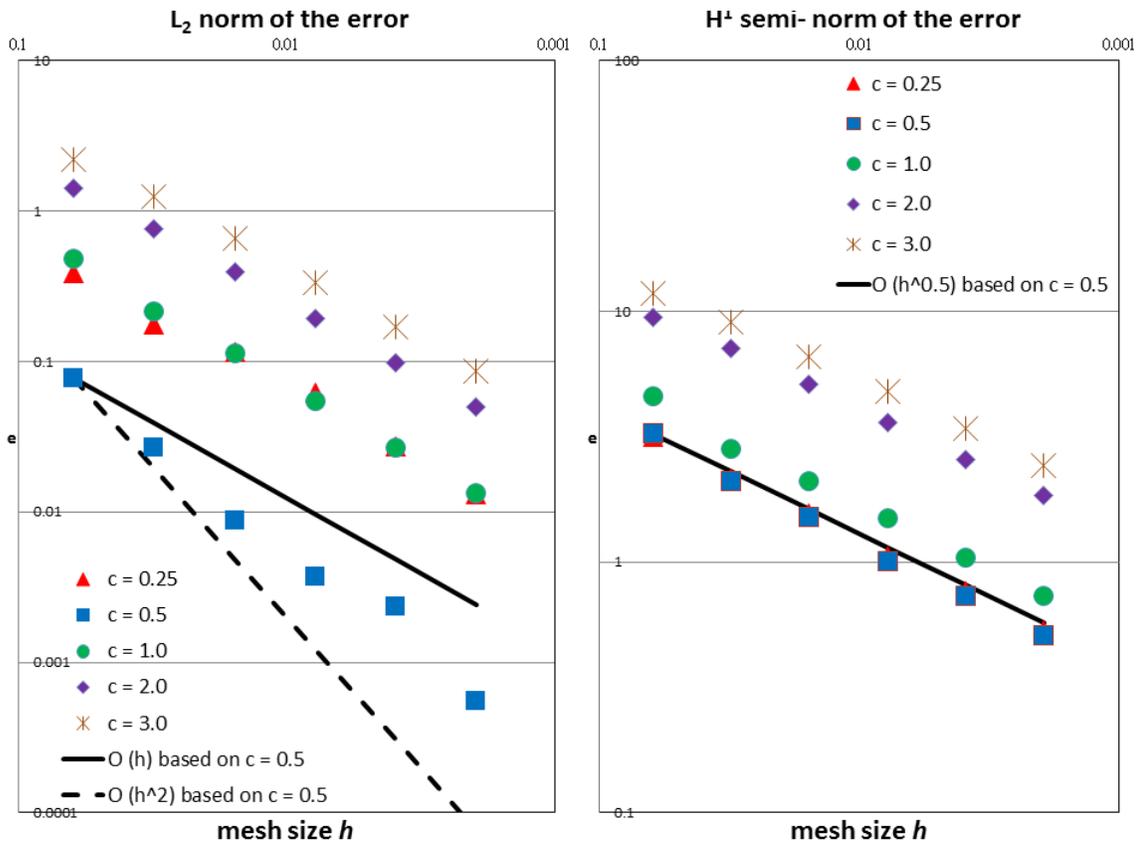


Figure 16 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.

Similar to the result from the constant function, in the subplot regarding the L_2 norm of the error, when $c \geq 1.0$, the obtained data point distributions behave linearly and have the first order accuracy. The smallest error still appears at $c = 0.5$, which looks more predicable than one from the similar setting with the constant function used as the weight in the integral. If we use the same way to estimate its accuracy, the L_2 norm of the error of this setting can be expressed as: $e \sim O(h^{1.4})$.

The subplot regarding the H^1 semi-norm of error is similar to the one in Figure 15, the accuracy of each settings has: $e \sim O(h^{0.54})$.

The Gaussian function is our third option for the function $k(x)$, which has been depicted in Figure 11(c) and equation (2-19). Corresponding results and figure are shown in Table 4 and drawn in Figure 17.

L ₂ norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	0.31232	0.06894	0.42033	1.30569	2.10923
$1/32$	0.17447	0.02258	0.17903	0.68595	1.15571
$1/64$	0.12812	0.00834	0.09329	0.34738	0.59777
$1/128$	0.06579	0.00467	0.04435	0.17211	0.31179
$1/256$	0.02623	0.00232	0.02168	0.08593	0.15677
$1/512$	0.01278	0.00065	0.01084	0.04372	0.07894
H ¹ semi-norm of the error					
Mesh size h	$c = 0.25$	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 3.0$
$1/16$	3.17329	3.21736	4.20694	8.91837	11.5767
$1/32$	2.1465	2.10763	2.58505	6.64084	8.69088
$1/64$	1.68342	1.49033	1.91003	4.72612	6.32215
$1/128$	1.12393	0.9954	1.33233	3.37059	4.62552
$1/256$	0.78604	0.728	0.92834	2.38849	3.28785
$1/512$	0.54383	0.50533	0.65519	1.70637	2.3373

Table 4 Results of the Poisson problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.

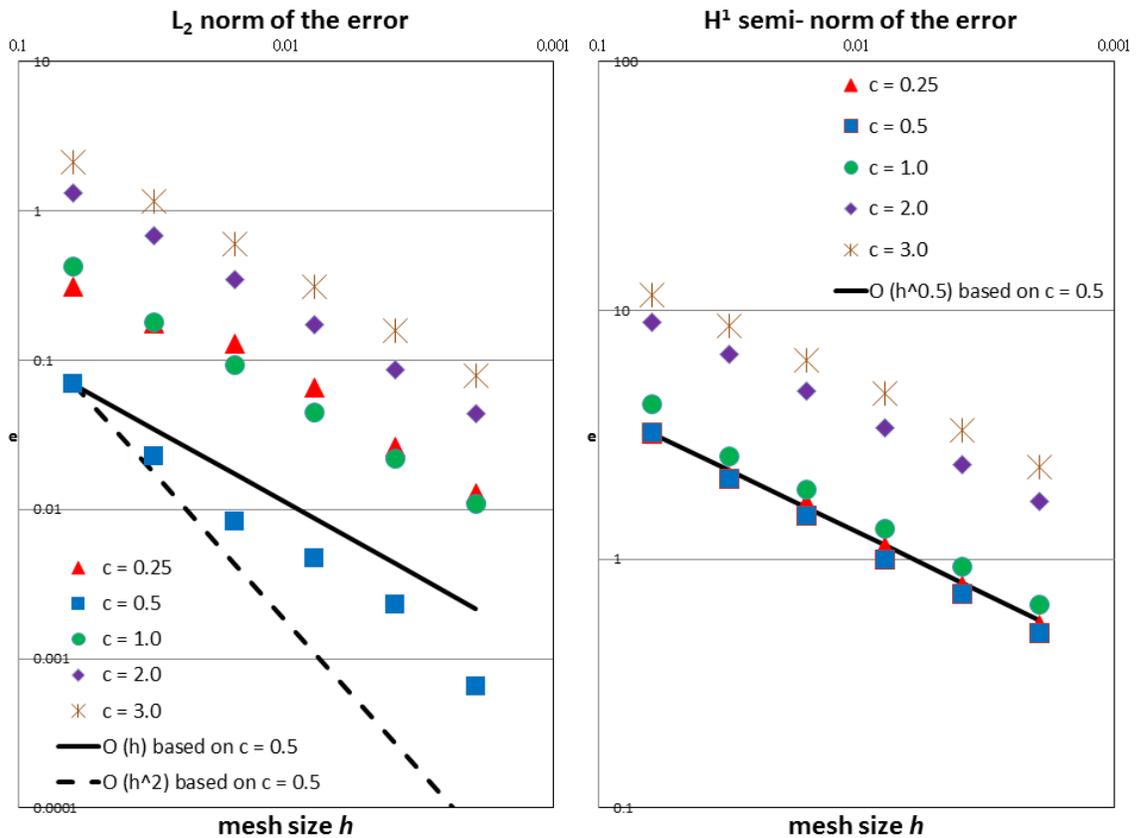


Figure 17 L_2 norm of the error and H^1 semi-norm of the error of the Poisson problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.

It is not surprising that results from the Gaussian function are similar to what we have from the previous two functions. We can still get predictable L_2 norm of the error from the setting with wider width, but have smaller errors from the one with $c = 0.5$. If we still use its first data point and the last data point to evaluate its accuracy, the L_2 norm of the error of this setting can be expressed as: $e \sim O(h^{1.35})$, which is slightly better than the one from the constant function, but slightly worse than one from the triangle function. In addition, all obtained H^1 semi-norm of the errors are still close to the reference line which represents $e \sim O(h^{0.53})$.

2.5.2.3 Irregular behavior for in narrow boundary

In all figures regarding the L_2 norm of the error from Strategy 2, it is easy to see that data point distributions are not linear when the width of the boundary region $\bar{\gamma}$ is small ($c < 1.0$). For example, in Figure 17, the first four data points from $c = 0.25$ oscillate.

To explain this phenomenon, we consider a 1D domain (refer to Figure 18). In this domain, what we want to find is an approximation of an integral in an interval with width equals to one element/mesh. The methodology for the numerical integration is the Gaussian Quadrature rule with three quadrature points, similar to what we do in our experiments. If the mesh is customized for this integration, like the body-fitted mesh, there is no difficulty in the computation since this interval is exactly located in an element (i.e., the red rectangle in Figure 18). We will have enough quadrature points to get information in this region.

However, if the mesh is not customized, like the one generated for the fictitious domain, it is possible that the number of quadrature points in an element is not correct (refer to the green rectangle in Figure 18). If so, error and instability appear.

This situation should also occur in the results with large constant c . Comparing to the settings with smaller constant c , however, the settings with large constant c have larger boundary region $\bar{\gamma}$. Therefore, it contains more quadrature points. The lack/addition of quadrature point certainly causes less influence to them. This is why their data point distributions are close to linear.

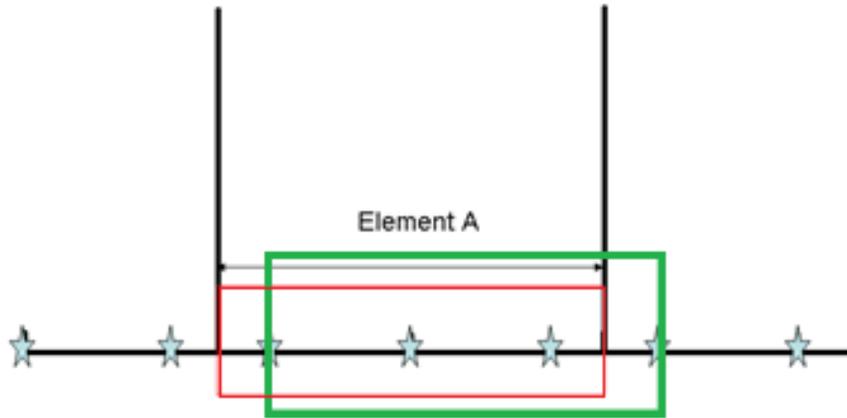


Figure 18 An example to explain why the results are not stable when constant c is smaller than 1; the star symbols represent the positions of quadrature points.

2.6 Summary

According to our experimental results, we can say that both Strategy 1 and Strategy 2 are capable of addressing the immersed boundary condition. The former one is an attractive choice if the user does not want to spend too much time on coding. While applying this strategy, he/she only needs to divide the whole domain into two parts: ω and $\Omega \setminus \omega$, then, impose the original constraint in $\Omega \setminus \omega$. Comparing to Strategy 2, which needs to set the boundary region $\bar{\gamma}$ and add additional $k(x)$ function, Strategy 1 is much simpler not only in concept but also in implementation.

On the other hand, if one wants to get more accurate solution, he/she should choose Strategy 2. Our experimental results imply that Strategy 1 may only have first order accuracy in its L_2 norm of the error. This number is not bad and is comparable to the one from the use of the penalty function method. However, it is still worse than Strategy 2, whose L_2 norm of the error can be obviously better than first order accuracy with suitable settings.

CHAPTER III

STOKES FLOW PROBLEM

3.1 Introduction

The objective of this chapter is to solve the Stokes flow problem by applying the fictitious domain method. Stokes flow is flow with very low Reynolds number ($Re \ll 1$). This kind of flow usually has high viscosity or very small velocity, which makes the advective inertial force become small in comparison with the viscous force. The high viscous force usually leads to slow fluid motions. Flows with such low Reynolds number are frequently called Stokes flow or creeping flow (refer to Panton [48]). Applications regarding Stokes flow can be seen in oil-lubricating bearings, aerosols, and the flow of lava.

From a mathematical point of view, the momentum equation of the Stokes flow problem can be viewed as a simplified Navier-Stokes equation. The only difference between these two lies in the nonlinear advection term. As a result, solving the Stokes problem can be viewed as a prerequisite for solving the Navier-Stokes system.

In this chapter, in addition to examining whether our strategies for addressing the immersed boundary condition is still valid in the problem whose desired solution is a vector with multiple components, we would like to find a new algorithm for solving the resulting system. Recall that in the previous chapter, we simply chose the solver UMFPACK to solve the discretized system. Although it is useful, it may not be practical

for large scale problems, since a lot of computational resources are needed. Therefore, we plan to find a suitable iterative algorithm for the proceeding Stokes flow problem.

The structure of this chapter is similar to the previous one. In the following sections of this chapter, we first introduce our designed test case in Chapter 3.2. Next, we explain how we implement the FD method. Numerical details about our code are described in Chapter 3.4. The main issue of this section lies in the iterative algorithm, which is based on the one mentioned in [17]. However, due to different treatments regarding the immersed boundary condition, significant modifications are needed. Chapter 3.5 is the section where we demonstrate our obtained experimental results. The last part of this chapter is Chapter 3.6, where we give an overall discussion.

3.2 Test case setting

In this section, we will first introduce the geometry of our test case. Then, we describe the governing equation and the corresponding weak formulations. After that, we assign an analytical solution, which can be used for error computations.

The test case is two rectangular domains with two fully developed Poiseuille flows. The flowing directions of these them are identical, both from left to right. The length and the width of these two rectangle domains are denoted as L and W . As to the distance between these two flows, it is denoted as Δy (see Figure 19).

The y coordinates of the top boundary and the bottom boundary of the Poiseuille flow (1) are denoted as y_1 and y_2 . The y coordinates of the top boundary and the bottom boundary of Poiseuille flow (2) are denoted as y_3 and y_4 .

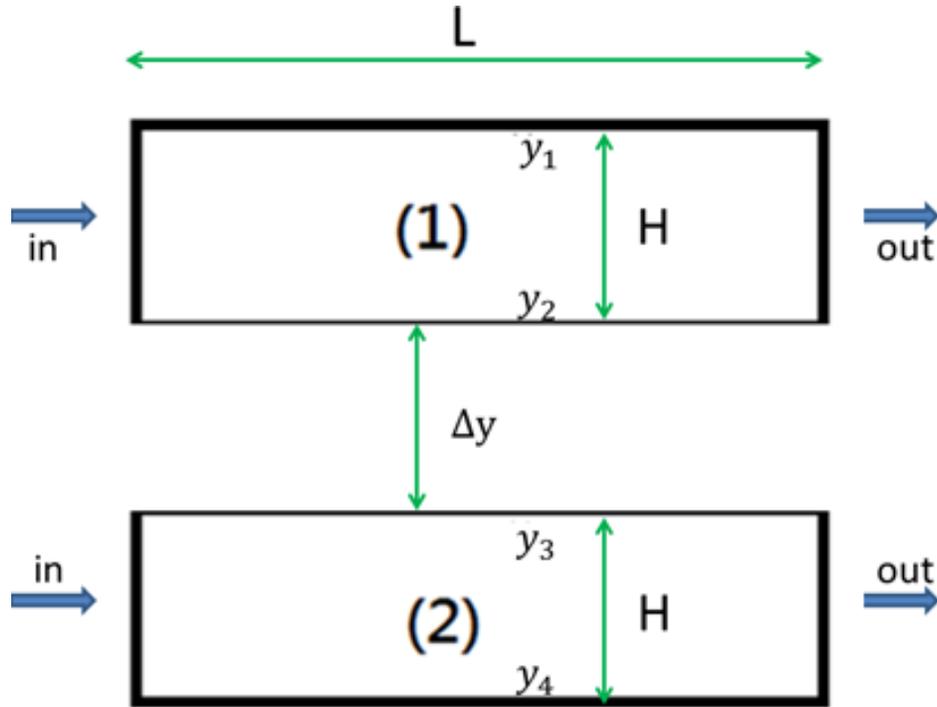


Figure 19 Flow domain for the Stokes flow problem.

We assume that the fluid velocities of these two Poiseuille flow are very slow.

Thus, we can express their governing equation as:

$$-\text{div}[2\eta \cdot \varepsilon(\mathbf{u})] + \nabla p = 0 \quad \text{in } \omega, \quad (3-1)$$

$$-\text{div } \mathbf{u} = 0 \quad \text{in } \omega, \quad (3-2)$$

where

$$\varepsilon(\mathbf{u}) = \frac{1}{2} [(\nabla \mathbf{u}) + (\nabla \mathbf{u})^T],$$

$$\mathbf{u} = 0.0 \text{ at } \Gamma.$$

In these equations, the domain ω represents the flow domain of these two Poiseuille flows. Γ defines the four horizontal boundaries of these two fluid domains. No slip boundary conditions are imposed on them. η is the dynamic viscosity of the fluid.

The solution pair (u, p) of equation (3-1) and (3-2) is also the solution for the following problem: Find $\mathbf{u} \in H^1(\omega)$, $p \in L^2(\omega)$, which satisfy:

$$2\eta(\varepsilon(v), \varepsilon(\mathbf{u}))_{\omega} - (\operatorname{div} v, p)_{\omega} = \langle v, 0 \rangle_{\omega} \quad \forall v \in H_0^1(\omega), \quad (3-3)$$

$$-(q, \operatorname{div} \mathbf{u})_{\omega} = \langle v, 0 \rangle_{\omega} \quad \forall q \in L_0^2(\omega), \quad (3-4)$$

where

$$(\varepsilon(v), \varepsilon(\mathbf{u}))_{\omega} = \int_{\omega} \varepsilon(v) \cdot \varepsilon(\mathbf{u}) \, dx,$$

$$(\operatorname{div} v, p)_{\omega} = \int_{\omega} \operatorname{div} v \cdot p \, dx,$$

$$\langle v, f \rangle_{\omega} = \int_{\omega} v \cdot f \, dx,$$

$$(q, \operatorname{div} \mathbf{u})_{\omega} = \int_{\omega} q \cdot \operatorname{div} \mathbf{u} \, dx,$$

$$\langle v, G \rangle_{\omega} = \int_{\omega} q \cdot G \, dx,$$

Functions v and q in equation (3-3) and (3-4) are test functions for the solutions \mathbf{u} and p .

Since we assume that these two flows are both fully developed, according to equation (3-1) and (3-2), we can get the velocity of the Poiseuille flow (1):

$$\mathbf{u} = u_1 \hat{i} + u_2 \hat{j} \quad (3-5)$$

where

$$u_1 = \frac{1}{2\eta} \frac{dp}{dx} (y - y_1)(y - y_2),$$

$$u_2 = 0.0.$$

Likewise, the velocity components of Poiseuille flow (2) can be obtained:

$$\mathbf{u} = u_1 \hat{i} + u_2 \hat{j}, \quad (3-6)$$

where

$$u_1 = \frac{1}{2\eta} \frac{dp}{dx} (y - y_3)(y - y_4),$$

$$u_2 = 0.0.$$

For simplicity, we further set the pressure gradient, $\frac{dp}{dx}$ as:

$$\frac{dp}{dx} = -8\eta.$$

Thus, we can simplify equation (3-5) and (3-6) as:

$$u_1 = -4(y - y_1)(y - y_2), \quad u_2 = 0.0, \quad \text{for Poiseuille flow (1).}$$

$$u_1 = -4(y - y_3)(y - y_4), \quad u_2 = 0.0, \quad \text{for Poiseuille flow (2).}$$

To complete these elaboration, we set y_1, y_2, y_3 and y_4 as 1.0, 0.3, -0.3 and -1.0. The reason why we set the position of the immersed boundary at ± 0.3 is that with this setting, the immersed boundary will not coincidentally lie on the mesh lines. Thus, our test results will be more meaningful.

Thus, we have:

$$u_1 = -4(y - 1.0)(y - 0.3), \quad u_2 = 0.0, \quad \text{for Poiseuille flow (1).} \quad (3-7)$$

$$u_1 = -4(y - (-0.3))(y - (-1.0)), \quad u_2 = 0.0, \quad \text{for Poiseuille flow (2).} \quad (3-8)$$

If we define the Reynolds number as: $\frac{U_{max} D_h}{\eta}$, where D_h is the hydraulic diameter, the resulting Reynolds number of our assigned solution is 0.01, which should be small enough to ignore the influence from the advection term in the Navier-Stokes equation.

3.3 Fictitious domain method implementation

In codes that use the body-fitted mesh, the linear system mentioned in Chapter 3.2 is enough to solve the Stokes flow problem since the necessary boundary conditions are imposed directly. However, if we want to solve the problem by using the FD method, significant modifications regarding the linear system are needed. In this section, we describe how we implement this method.

3.3.1 Fictitious domain formulation

Here, we first set the distance between these two Poiseuille flows to be equal to 0.6. Next, just like what we have done in Chapter II, we embed these two flows in a larger square domain Ω , which can be expressed as $\Omega := (-1, -1) \times (1, 1)$ (see Figure 20). Since this domain is also a square domain, the mesh generated for this domain is identical to the one in Figure 9.

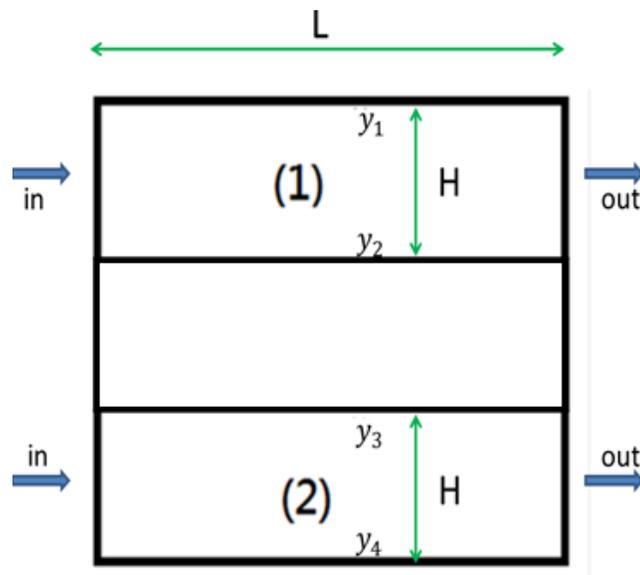


Figure 20 Fictitious domain for the Stokes flow problem.

By considering the Minimum Functional Theorem and adding the Lagrange multiplier, we can have the Lagrangian functional for the needed domain ω :

$$L(\mathbf{u}, p, \lambda) = \frac{1}{2} \int_{\omega} 2\eta |\nabla \varepsilon|^2 dx - \int_{\omega} p \nabla \mathbf{u} dx - \int_{\omega} (F \cdot \mathbf{u}) dx + \int_{\Gamma} \lambda \cdot (\mathbf{u} - g_1) dx \quad \text{in } \omega. \quad (3-9)$$

g_1 represents the no-slip boundary condition that needs to be imposed on the horizontal boundaries of these two flows.

Next, we can define the Lagrangian functional for the fictitious domain Ω based on equation (3-9):

$$L(\tilde{\mathbf{u}}, \tilde{p}, \lambda) = \frac{1}{2} \int_{\Omega} 2\eta |\nabla \tilde{\varepsilon}|^2 dx - \int_{\Omega} \tilde{p} \nabla \tilde{\mathbf{u}} dx - \int_{\Omega} (\tilde{F} \cdot \tilde{\mathbf{u}}) dx + \int_{\gamma} \lambda \cdot (\tilde{\mathbf{u}} - \tilde{g}_1) dx \quad \text{in } \Omega, \quad (3-10)$$

where the solutions $\tilde{\mathbf{u}}$ and \tilde{p} are valid in the whole domain Ω and γ represents the bottom surface of Poiseuille flow (1) and the top surface of Poiseuille flow (2). These two boundaries are the immersed boundaries in this test case. As to the constraints imposed on the top surface of Poiseuille flow (1) and bottom surface of Poiseuille flow (2), which are also no-slip boundary condition, they will be assigned directly. In addition, $\tilde{F}|_{\omega} = F = 0.0$ and $\tilde{g}_1|_{\gamma} = g_1 = 0.0$. For simplicity, we set $\tilde{F}|_{\Omega \setminus \omega} = 0.0$.

Finally, we can apply the variation principle to the Lagrangian functional L (equation (3-10)) and get the weak formulations for the fictitious domain:

$$2\eta(\varepsilon(v), \varepsilon(\tilde{\mathbf{u}}))_{\Omega} - (\nabla v, \tilde{p})_{\Omega} = (v, \tilde{F})_{\Omega} - (v, \lambda)_{\gamma}, \quad (3-11)$$

$$-(q, \nabla \tilde{\mathbf{u}})_{\Omega} = 0, \quad (3-12)$$

$$(\mu, \tilde{\mathbf{u}})_{\gamma} = (\mu, \tilde{g}_1)_{\gamma}. \quad (3-13)$$

These three equations are desired weak formulation we are going to use for solving our test case. Again, addressing terms related to the immersed boundary plays an important role. How treat the immersed boundary condition on γ will be the topic of the next two sub-sections.

3.3.2 IB condition treatment: Strategy 1

In our strategy, we let the desired constraint live everywhere but only equal to the immersed boundary condition in the area $\Omega \setminus \omega$. As a result, the term related to the immersed boundary can be expressed as:

$$\int_{\gamma} a \, dx \Rightarrow \int_{\Omega \setminus \omega} a \, dx + \int_{\omega} 0 \, dx.$$

With the addition of the weight factor, $w_1(x)$, the expression becomes:

$$\int_{\gamma} a \, dx \Rightarrow \int_{\Omega} w_1(x) \cdot a \, dx,$$

where

$$w_1(x) = \begin{cases} 1, & x \in \Omega \setminus \omega \\ 0, & x \in \omega \end{cases}$$

In this test case, when the y coordinate of any point is within the interval, $[-0.3, 0.3]$, the weight factor $w(x)$ equals to 1 (the area occupied by blue tilt lines in Figure 21).

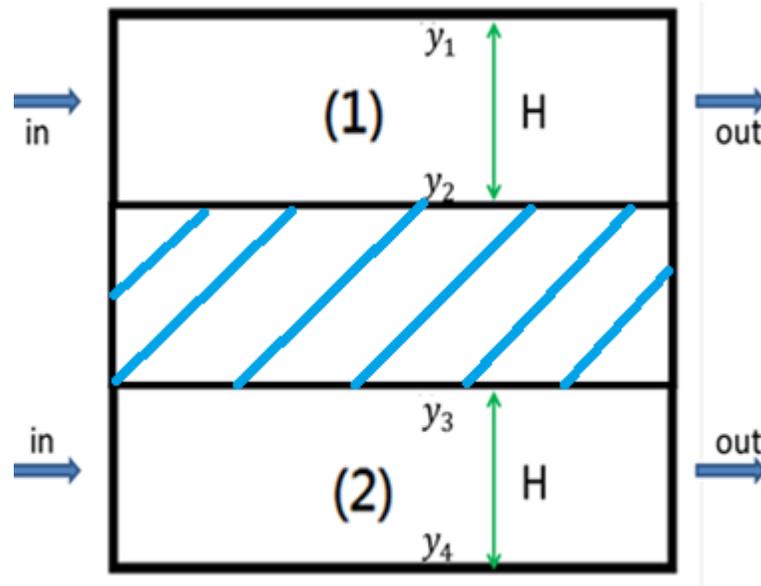


Figure 21 Extension of the immersed boundary condition after applying Strategy 1.

3.3.3 IB condition treatment: Strategy 2

The concept of Strategy 2 is to replace the original immersed boundary by a boundary region $\bar{\gamma}$:

$$\bar{\gamma}: y_{ib} - hf \leq y \leq y_{ib} + hf.$$

In this expression, the parameter y_{ib} represents the y coordinate of the immersed boundary. To Poiseuille flow (1), it equals to 0.3 (i.e., y_2 in Figure 22). As to Poiseuille flow (2), it equals to -0.3 (i.e., y_3 in Figure 21). hf still represents the half width of the boundary region, which can be obtained from equation (2-16). Therefore, the boundary region $\bar{\gamma}$ looks like two regions in Figure 22.

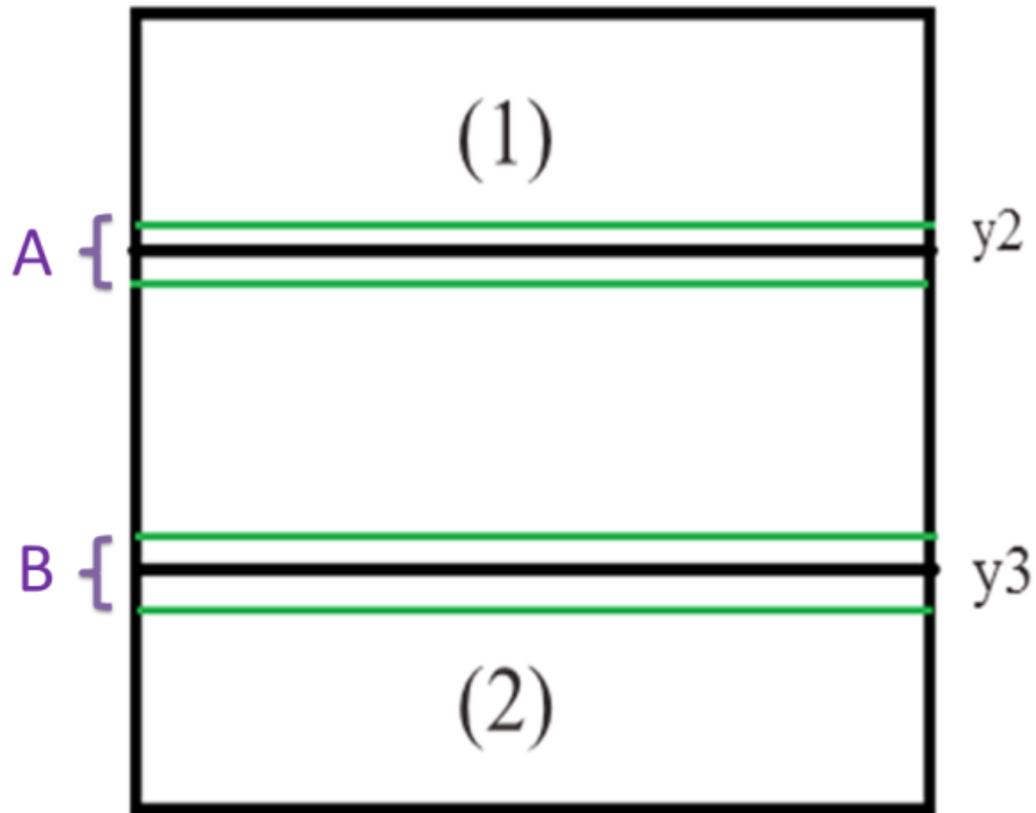


Figure 22 Boundary region $\bar{\gamma}$ for the Stokes flow problem.

Just like we mentioned in the previous chapter, in order to make this replacement reasonable, in addition to applying a weight function $w(x)$:

$$w_2(x) = \begin{cases} 0, & \text{in } \Omega \setminus \bar{\gamma} \\ 1, & \text{in } \bar{\gamma} \end{cases},$$

another weight function $k(x)$ needs to be introduced as well. In this test case, we still apply and test three different $k(x)$ functions: constant function, triangle function and Gaussian function. A thorough explanation of these three functions can be founded in Chapter 2.3.3.

3.4 Iterative algorithm

In theory, the proceeding Stokes flow problem is a saddle point problem. The characteristics of the saddle point problem are indefinite (refer to Benzi et al. [49]). These features make saddle point problems hard to solver.

There are mainly two approaches for solving saddle point problems: segregated methods and coupled methods (refer to [49]). Since the main goal of the previous chapter is to see whether our strategies are feasible, we directly apply the direct solver UMFPACK. This belongs to the coupled methods.

Although it works well, it is still a direct solver. This means that it may need more operations and more computer resources. For large scale problems, like our ultimate goal, segregated methods are more appropriate. This is why we want to find an iterative algorithm to serve as our solver for solving the resulting matrix systems.

According to [49], the most representative approach among segregated methods is the Schur complement reduction method. To explain it, let us first consider a saddle point problem that can be expressed as:

$$\begin{bmatrix} M & B^T \\ B & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ G \end{bmatrix}.$$

In the pure Stokes flow problem (without considering Lagrange multiplier), each block can be represented as: $M = -\Delta$, $B^T = \nabla$, $B = -\nabla$, $C = 0$, $x = \mathbf{u}$, $y = p$, $f = F$ and $G = 0$. This system can also be written as:

$$M\mathbf{u} + B^T p = F, \tag{3-14}$$

$$B\mathbf{u} = 0.0. \tag{3-15}$$

By multiplying the first equation by BM^{-1} , we have:

$$BM^{-1}M\mathbf{u} + BM^{-1}B^T\mathbf{p} = BM^{-1}\mathbf{F},$$

Obviously, $BM^{-1}M\mathbf{u} = B\mathbf{u} = 0.0$. We can further define the Schur complement S as $S = BM^{-1}B^T$. Thus, equation (3-14) can be expressed as:

$$S\mathbf{p} = BM^{-1}\mathbf{F}. \quad (3-16)$$

After getting the solution \mathbf{p} from this equation, we substitute \mathbf{p} in the equation (3-14) to get the following equation:

$$M\mathbf{u} = \mathbf{F} - B^T\mathbf{p}. \quad (3-17)$$

This is the one used to get the solution \mathbf{u} of the saddle point problem. In addition, stationary iteration methods, such as the Jacobi method and the Gauss-Seidel method, or Krylov subspace method, such as CG and GMRES can be used to solve equation (3-16) and equation (3-17) respectively.

In general, the system composed of equation (3-16) and equation (3-17) will be solved iteratively by using the Uzawa iteration. While using this approach, the user will first guess \mathbf{p}^0 , then, apply it to solve \mathbf{u}^0 . Then, these \mathbf{u}^0 and \mathbf{p}^0 will be used compute the residual, \mathbf{r}^0 such as:

$$\mathbf{r}^0 = \mathbf{F} - M\mathbf{u}^0 - B^T\mathbf{p}^0.$$

If \mathbf{r}^0 is smaller than the desired value, we can say we get the converged solution. Otherwise, based on obtained \mathbf{u}^0 and \mathbf{p}^0 , the whole process will be re-executed to get new solutions \mathbf{u}^{n+1} and \mathbf{p}^{n+1} . Just like we mentioned, the whole process will stop until the new solution is converged. For us, the situation is more difficult, since we need to consider an additional condition: the immersed boundary condition.

The corresponding iterative algorithm used for solving our test case is a variant of the Uzawa/conjugated gradient algorithm. This algorithm was originally proposed to solve transient flow problems (refer to Glowinski [50] and Glowinski and Pironneau [51]). In Glowinski et al. [9, 17], it was proved to be capable of solving flow problems with immersed boundary. A complete description regarding this algorithm can be seen in Glowinski and Le Tallec [52]. However, since we choose different treatments to address the immersed boundary condition, some modifications are needed.

To make the whole algorithm clear, we write down all steps of our algorithm:

I. Initial part

1. Initiate λ^0 as 0.0
2. Obtain initial \tilde{u}^0 and \tilde{p}^0 by solving the Stokes flow system of the initial part:

$$\begin{aligned} 2\nu(\varepsilon(v), \varepsilon(\tilde{u}^0))_{\Omega} - (\nabla v, \tilde{p}^0)_{\Omega} &= (v, \tilde{F})_{\Omega} - (v, \lambda^0)_{\gamma}, \\ -(q, \nabla \tilde{u}^0)_{\Omega} &= 0, \end{aligned}$$

where $\tilde{F} = 0.0$. In addition, we need to impose the no-slip boundary condition on the top surface and the bottom surface of the fictitious domain Ω . As to the left surface and the right surface of Ω , we impose the needed input and exit conditions, which equal to equation (3-7) and equation (3-8). Here, we have to mention that the inlet and exit boundary condition have nonzero value only in the area between y_1 and y_2 and the area y_3 and y_4 (see Figure 19).

The modification we need to make is to the term $(v, \lambda^0)_{\gamma}$. In Strategy 1, it becomes:

$$(v, \lambda^0)_{\gamma} \Rightarrow (w_1(x) \cdot v, \lambda^0)_{\Omega},$$

where

$$w_1(x) = \begin{cases} 1, & x \in \Omega \setminus \omega \\ 0, & x \in \omega \end{cases}.$$

If we apply Strategy 2, this term can be changed to:

$$(v, \lambda^0)_\gamma \Rightarrow (w_2(x) \cdot k(x) \cdot v, \lambda^0)_\Omega,$$

where

$$w_2(x) = \begin{cases} 1, & x \in \Omega \setminus \bar{\gamma} \\ 0, & x \in \bar{\gamma} \end{cases}.$$

3. Compute g^0

This parameter represents the difference between the solution \tilde{u}^0 and the desired immersed boundary condition (i.e., no-slip boundary condition). Similar to the previous step, if we use Strategy 1, we have:

$$\begin{aligned} & (w_1(x) \cdot \mu, g^0)_\Omega + ([1 - w_1(x)] \cdot \mu, g^0)_\Omega \\ &= (w_1(x) \cdot \mu, (\tilde{u}^0 - 0.0))_\Omega + ([1 - w_1(x)] \cdot \mu, 0)_\Omega. \end{aligned}$$

For Strategy 2

$$\begin{aligned} & (w_2(x) \cdot k(x) \cdot \mu, g^0)_\Omega + ([1 - w_2(x)] \cdot \mu, g^0)_\Omega \\ &= (w_2(x) \cdot k(x) \cdot \mu, (\tilde{u}^0 - 0.0))_\Omega + ([1 - w_2(x)] \cdot \mu, 0)_\Omega. \end{aligned}$$

4. Set $W^0 = g^0$

W^0 is the initial descent direction.

II. Loop part (for $n \geq 0$, where n is the iteration number for the loop part)

5. Get intermediate solution \bar{u}^n and \bar{p}^n from:

$$\begin{aligned} & 2\eta(\varepsilon(v), \varepsilon(\bar{u}^n))_\Omega - (\nabla v, \bar{P}^n)_\Omega = (v, W^n)_\gamma, \\ & -(q, \nabla \bar{u}^n)_\Omega = 0. \end{aligned}$$

Similarly, the term on the right hand side of the first equation (i.e., $(v, W^n)_\gamma$) needs to be adjusted depending on the strategy we would like to apply. While choosing Strategy 1, it becomes:

$$(w_1(x) \cdot v, W^n)_\Omega.$$

For Strategy 2, we use

$$(w_2(x) \cdot k(x) \cdot v, W^n)_\Omega.$$

6. Compute the parameter ρ_n using:

$$\rho_n = \frac{(w_1(x) \cdot |g^n|, |g^n|)_\Omega}{(w_1(x) \cdot \bar{u}^n, W^n)_\Omega} \quad \text{for strategy 1,}$$

or

$$\rho_n = \frac{(w_2(x) \cdot k(x) \cdot |g^n|, |g^n|)_\Omega}{(w_2(x) \cdot k(x) \cdot \bar{u}^n, W^n)_\Omega} \quad \text{for Strategy 2.}$$

In the original papers (refer to [9, 17, 18]), the ρ_n was computed on the immersed boundary.

7. renew λ , $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{p}}$:

$$\lambda^{n+1} = \lambda^n - \rho_n W^n,$$

$$\tilde{\mathbf{u}}^{n+1} = \tilde{\mathbf{u}}^n - \rho_n \bar{\mathbf{u}}^n,$$

$$\tilde{\mathbf{p}}^{n+1} = \tilde{\mathbf{p}}^n - \rho_n \bar{\mathbf{p}}^n.$$

8. Renew g^n

Likewise, the renewal of this parameter still depends on the chosen strategy.

In Strategy 1:

$$\begin{aligned} & (w_1(x) \cdot \mu, g^{n+1})_\Omega + ([1 - w_1(x)] \cdot \mu, g^{n+1})_\Omega \\ &= (w_1(x) \cdot \mu, g^n)_\Omega - \rho^n (w_1(x) \cdot \mu, \bar{u}^n) + ([1 - w_1(x)] \cdot \mu, 0.0)_\Omega. \end{aligned}$$

In Strategy 2:

$$\begin{aligned} & (w_2(x) \cdot k(x) \cdot \mu, g^{n+1})_\Omega + ([1 - w_2(x)] \cdot \mu, g^{n+1})_\Omega \\ &= (w_2(x) \cdot k(x) \cdot \mu, g^n)_\Omega - \rho^n (w_2(x) \cdot k(x) \cdot \mu, \bar{u}^n) + ([1 - w_2(x)] \cdot \mu, 0.0)_\Omega. \end{aligned}$$

9. Renew W^{n+1}

The new descent direction W^{n+1} is renewed by using:

$$W^{n+1} = g^{n+1} + r_n W^n,$$

where

$$r_n = \frac{(w_1(x) \cdot |g^{n+1}|, |g^{n+1}|)_\Omega}{(w_1(x) \cdot |g^n|, |g^n|)_\Omega} \quad \text{for Strategy 1.}$$

$$r_n = \frac{(w_2(x) \cdot k(x) \cdot |g^{n+1}|, |g^{n+1}|)_\Omega}{(w_2(x) \cdot k(x) \cdot |g^n|, |g^n|)_\Omega} \quad \text{for Strategy 2.}$$

10. Compute the iteration error of the loop part

The original equation in [9, 17, 18] for this computation was:

$$\frac{(|g^{n+1}|, |g^{n+1}|)_\gamma}{(|g^0|, |g^0|)_\gamma}.$$

It is straightforward to transform it as the following equation, if we choose Strategy

2:

$$\frac{(w_2(x) \cdot k(x) \cdot |g^{n+1}|, |g^{n+1}|)_\Omega}{(w_2(x) \cdot k(x) \cdot |g^0|, |g^0|)_\Omega}.$$

However, to Strategy 1, we may need to take an additional action: confine the range regarding this computation. Recall in Strategy 1, we extend the constraint to live everywhere but equals to the desired constraint in the area outside the needed

domain (i.e., $\Omega \setminus \omega$). If so, then, we should evaluate the loop error in the same area, like:

$$\frac{(w_1(x) \cdot |g^{n+1}|, |g^{n+1}|)_{\Omega}}{(w_1(x) \cdot |g^0|, |g^0|)_{\Omega}}.$$

As a result, we decide to narrow down the area about this equation when we apply Strategy 1 to the region near the immersed boundary, such as:

$$y \in (0.3, 0.3 + hf) \quad \text{for Poiseuille flow (1),}$$

$$y \in (-0.3, -0.3 - hf) \quad \text{for Poiseuille flow (2).}$$

The product of this step will be used as a criterion to decide whether the obtained solution satisfies the immersed boundary condition. If it is equal or smaller than the stopping criterion, ξ , we can stop the loop and say we got what we needed. Otherwise, we need to restart to compute all steps in the loop part (i.e., step-5 to step-10).

One critical issue about this iterative algorithm lies on how to choose the tolerance ξ . In [9], the authors set it to be 10^{-7} . However, we do not think this is appropriate to our situation. This is because in our iterative algorithm, the iteration error is evaluated in a region, not a series of points. As a result, the iteration error should be intrinsically influenced by the discretization error. In the next subsection, we are going to discuss this issue.

3.4.1 Stopping criterion

In our iterative algorithm, the iteration error in our iterative algorithm is evaluated in a region, whether we apply Strategy 1 or Strategy 2. Therefore, an appropriate choice for this is to connect ξ with the mesh size h , such as:

$$\xi = \frac{d \times h}{L}, \quad (3-18)$$

where d represents an arbitrary constant which can be decided by the user, and L is the characteristic length for the fictitious domain. This equation can be further simplified as:

$$\xi = e \times h, \quad (3-19)$$

where

$$e = \frac{d}{L}.$$

The influence from this issue will be further discussed in Chapter 3.6.3.

3.4.2 Krylov subspace method

Previously, we mentioned that the individual linear system after applying Schur complement reduction method (e.g., equation (3-16) or equation (3-17)) can be solved by applying stationary iteration methods or Krylov subspace methods. Among these two categories, we choose the latter one since it is more efficient.

The choice of Krylov subspace methods highly depends on the characteristics of the coefficient matrix. For example, to a linear system, $Ax = b$, if A is a symmetric positive definite matrix, the conjugated gradient method (CG method) can solve it fast.

In addition to considering the features of the system matrix, to get better efficiency, while using this method, we should consider preconditioning. Choosing a

good preconditioner can speed up the convergence of the solver. To the system, $Ax = b$, if we apply a left preconditioner, it can be written as:

$$PAx = Pb.$$

If P is the inverse matrix, A^{-1} , this system can be solved directly. Unfortunately, most of time, it is very expensive to get A^{-1} . Therefore, a practical way to address this issue is to compose a preconditioner which approximates A^{-1} . Saad [53] is a good reference to get a thorough understanding regarding this issue.

3.5 Numerical details

In this test case, the weak formulation is discretized by using the Finite Element Method (FEM). In order to satisfy the LBB condition (refer to Girault and Raviart [54]), we compute the solutions \mathbf{u}_h and p_h with Q2 and Q1 element respectively. As to the Lagrange multiplier, since the constraint we are going to add on the immersed boundary is related to the velocity, we discretize the Lagrange multiplier λ_h with Q2 elements as well. The integration computation is still evaluated by applying Gaussian Quadrature.

We would like to use a CG solver to solve all resulting linear systems regarding solutions u_h and p_h . Therefore, we simplify the complex preconditioning issue as much as possible. Let us consider the Schur complement of equation (3-16), which can be expressed as:

$$S = BA^{-1}B \approx -div(-div2\eta\nabla^s)^{-1}\nabla.$$

It is spectrally equivalent to the identity matrix, I . As a result, it is easy to precondition.

According to Heister [55], \tilde{S} can be approximated as:

$$S^{-1} \approx -\eta M_p^{-1} = \tilde{S}^{-1}$$

where M_p is the mass matrix in the pressure space.

In contrast, the block A is equivalent to $-\text{div}2\eta\nabla^s$, which is close to the Laplace operator. Therefore, getting A^{-1} is a little complicated. To address this problem, we use a direct sparse LU decomposition of the matrix as preconditioner. This preconditioner can be obtained by calling the solver UMFPACK. Details about all the preconditioning techniques used in this test case can be seen in the tutorial step-22 of deal.II (see <http://www.dealii.org>).

3.6 Experimental results

In this section, we are going to present our numerical results. The structure is similar to Chapter 2.5. Since we have two different strategies for addressing the immersed boundary condition, we have two subsections. In each subsection, the solution profile is shown first. Then, L_2 norm of the error and H^1 semi-norm of the error are demonstrated to evaluate the accuracy.

3.6.1 Results from Strategy 1

The use of this strategy extends the constraint on the immersed boundary to be valid in the area outside the needed domain. In this test case, this represents the inner block of the fictitious domain (see Figure 20).

3.6.1.1 Solution profile from Strategy 1

Again, to have some idea about the physical phenomenon, we directly simulate one of the two Poiseuille flows in our test case by modifying the code described in the deal.II tutorial step-22. The mesh and the obtained velocity profile of the direct simulation are shown in Figure 23. Since the flow is assumed to be fully developed, it is not surprising that the velocity distribution looks stratified.

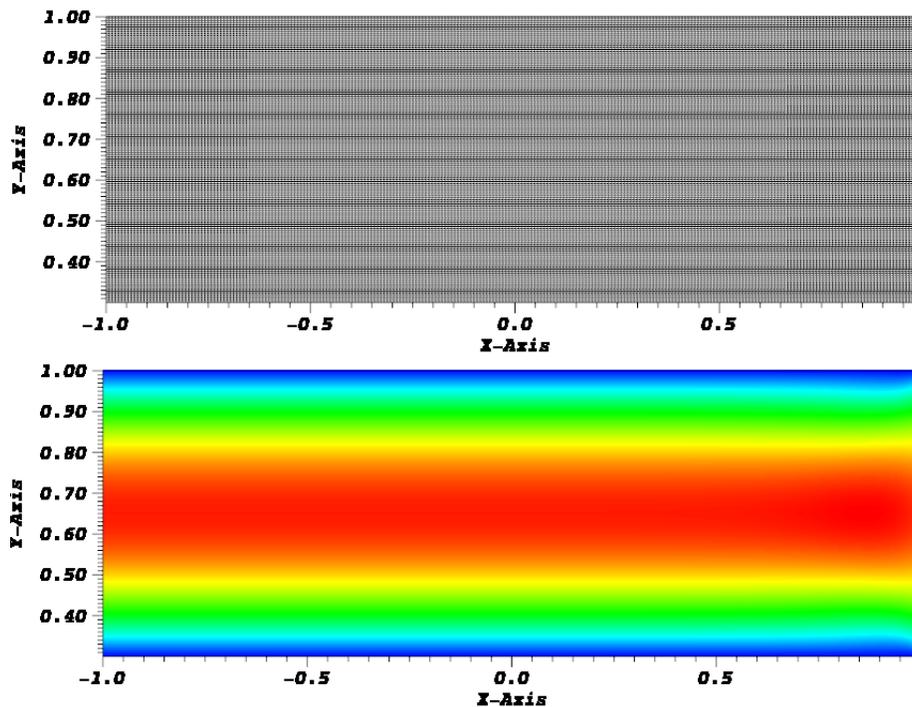


Figure 23 Mesh and obtained velocity profile of the Stokes flow problem obtained from the direct simulation.

The result shown in Figure 24 is the velocity profile gained from our implementation. The top portion of this figure represents the flow domain (1) and the bottom portion of this figure represents the flow domain (2). We can clearly see that the velocity distributions in these two regions are similar to what we have in Figure 23.

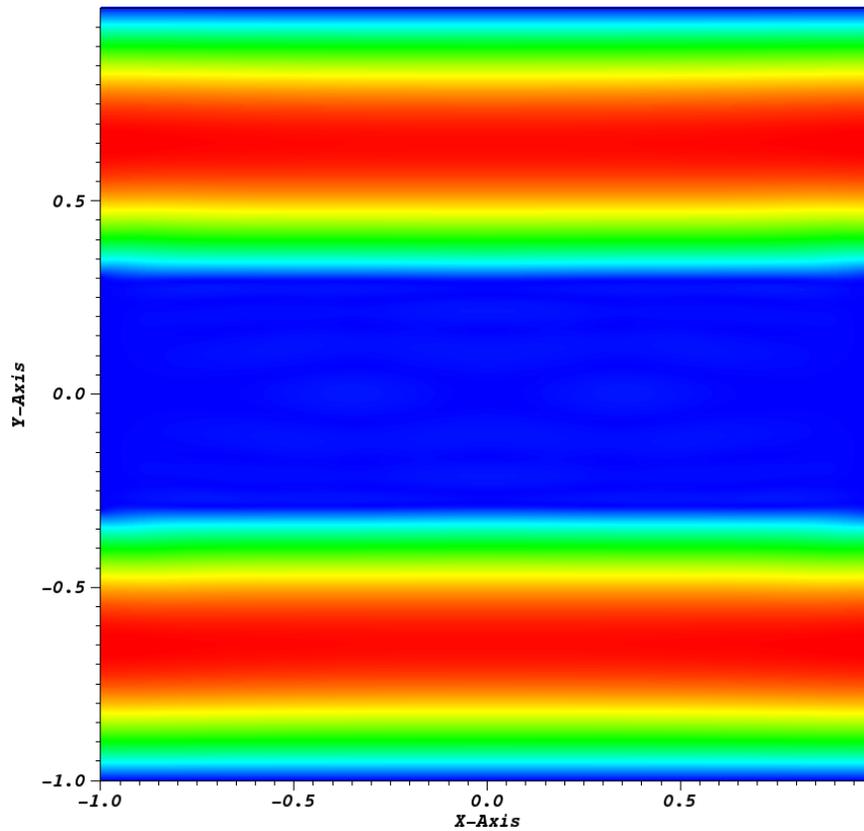


Figure 24 Velocity profile of the Stokes flow problem obtained from Strategy 1.

In addition, since our initial guess regarding the Lagrange multiplier (i. e., λ^0) is zero (see step-1 in Chapter 3.4), the solution obtained from the initial part is identical to the one from the Stokes system without considering the immersed boundary condition. This corresponding result is shown in Figure 25. In this figure, we see that the two Poiseuille flows merge in the center portion, which is reasonable.

According to these three figures, we can say that the result from Strategy 1 is qualitatively correct.

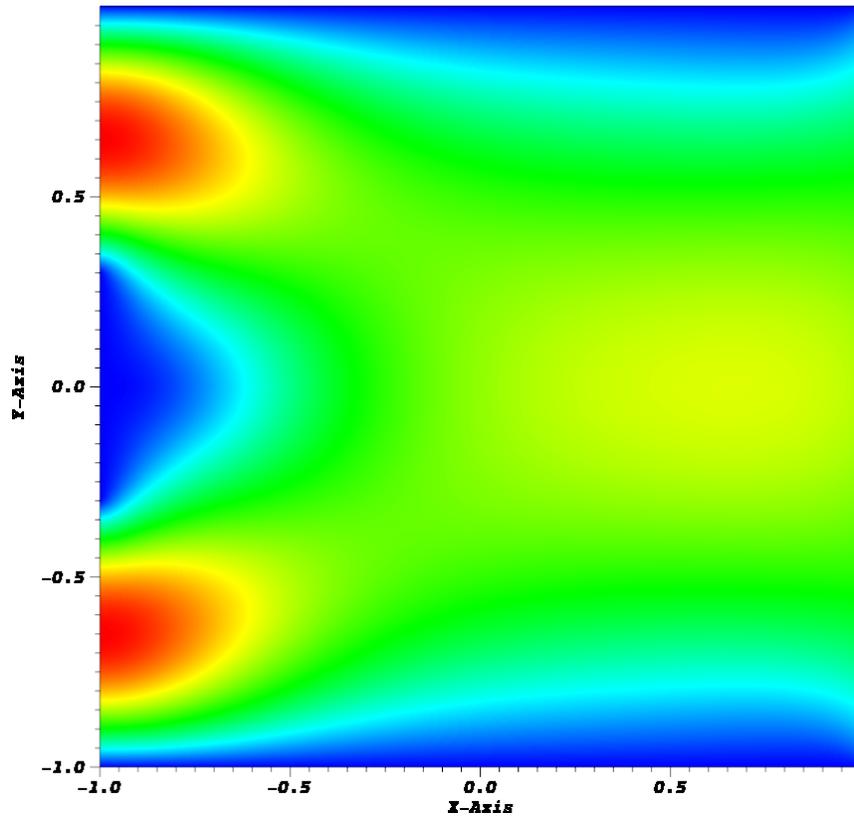


Figure 25 Velocity profile without imposing the immersed boundary condition.

3.6.1.2 Error analysis of Strategy 1

In addition to confining the range regarding iteration error computation (refer to step-9 in Chapter 3.4), we also need to decide the stopping criterion, ξ . As mentioned in Chapter 3.4.1, it should be related to the mesh size, h . This is why we define equation (3-22):

$$\xi = e \times h.$$

Here, we simply set e as 1.0.

Results obtained from this strategy are listed and drawn in Table 5 and Figure 26.

It is obvious that results from this strategy with $\xi = 1.0h$ are not ideal, no matter in L2

norm of the error or H^1 semi-norm of the error. Both of these two data point distributions behave stepwise. Even worse, the error does not necessarily decrease with every mesh refinement step.

Mesh size h	L_2 norm of the error	H^1 semi-norm of the error
$1/16$	0.04841	0.46751
$1/32$	0.03798	0.40623
$1/64$	0.03828	0.40212
$1/128$	0.02254	0.23044
$1/256$	0.0228	0.23216

Table 5 Results of the Stokes flow problem obtained from Strategy 1 with $\xi = 1.0h$.

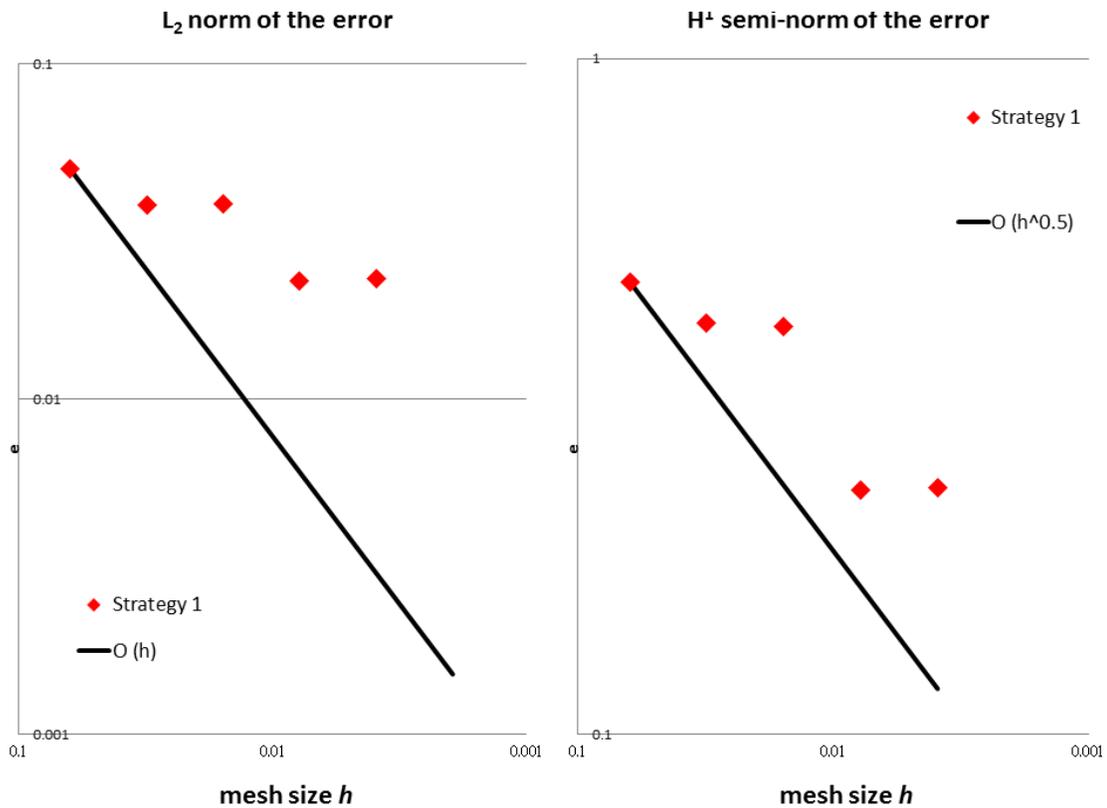


Figure 26 L_2 norm of the error and H^1 norm of the error of the Stokes flow problem obtained from Strategy 1 with $\xi = 1.0h$.

3.6.2 Results from Strategy 2

In this Strategy, we replace the immersed boundary γ by the boundary region $\bar{\gamma}$. A $k(x)$ function is added to validate this replacement. In our research, we choose three different functions to play this role.

The results from this strategy will be clearly affected by the width of the boundary. This idea is proved in the previous chapter. As a result, in the proceeding test case, we still choose different width of the boundary region in our experiment, except we reduce the number regarding the constant c from five (0.25, 0.5, 1.0, 2.0 and 3.0) to three (0.5, 1.0 and 2.0).

As to the stopping criterion of the loop part, similar to Chapter 3.6.1, we simply set $\xi = 1.0h$.

3.6.2.1 Solution profile from Strategy 2

Comparison between Figure 27 and Figure 23, we can say that the velocity profile obtained from this strategy is also acceptable.

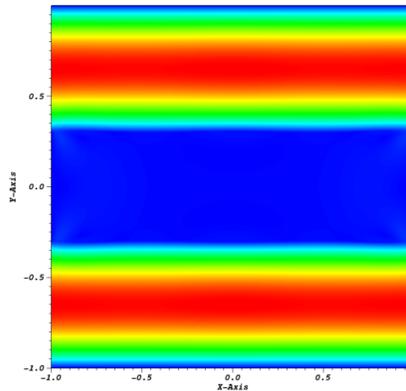


Figure 27 Velocity profile of the Stokes flow problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.

3.6.2.2 Error analysis of Strategy 2

In this portion, we first demonstrate results from the constant function in Table 6 and Figure 28.

It is shown that when the width of the boundary region is small (e.g., $c \leq 1.0$), the data point distribution in the coarse mesh is not ideal (red colored numbers in Table 6). In addition, distributions from these two settings do not behave ideally either (see Figure 28).

Better results can be obtained with $c = 2.0$. Its L_2 norm of the error has more than 1st order accuracy. If we compute it based on the first data point and the last data point, we can express it as: $e \sim O(h^{1.2})$. It is surprising that the H^1 semi-norm of the error of this setting can be shown as $e \sim O(h^{0.95})$, which is close to the first order accuracy.

Mesh size h	$\xi = 1.0 \times h$					
	L_2 norm of the error			H^1 semi-norm of the error		
	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 0.5$	$c = 1.0$	$c = 2.0$
$1/16$	0.05268	0.05081	0.1979	0.43544	0.57611	2.78541
$1/32$	0.05729	0.04613	0.09708	0.45888	0.67182	1.59115
$1/64$	0.03233	0.03344	0.03773	0.31828	0.39246	0.67032
$1/128$	0.00644	0.00665	0.01298	0.14818	0.15687	0.28139
$1/256$	0.006	0.006	0.00731	0.12274	0.14295	0.19611

Table 6 Results of the Stokes flow problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.

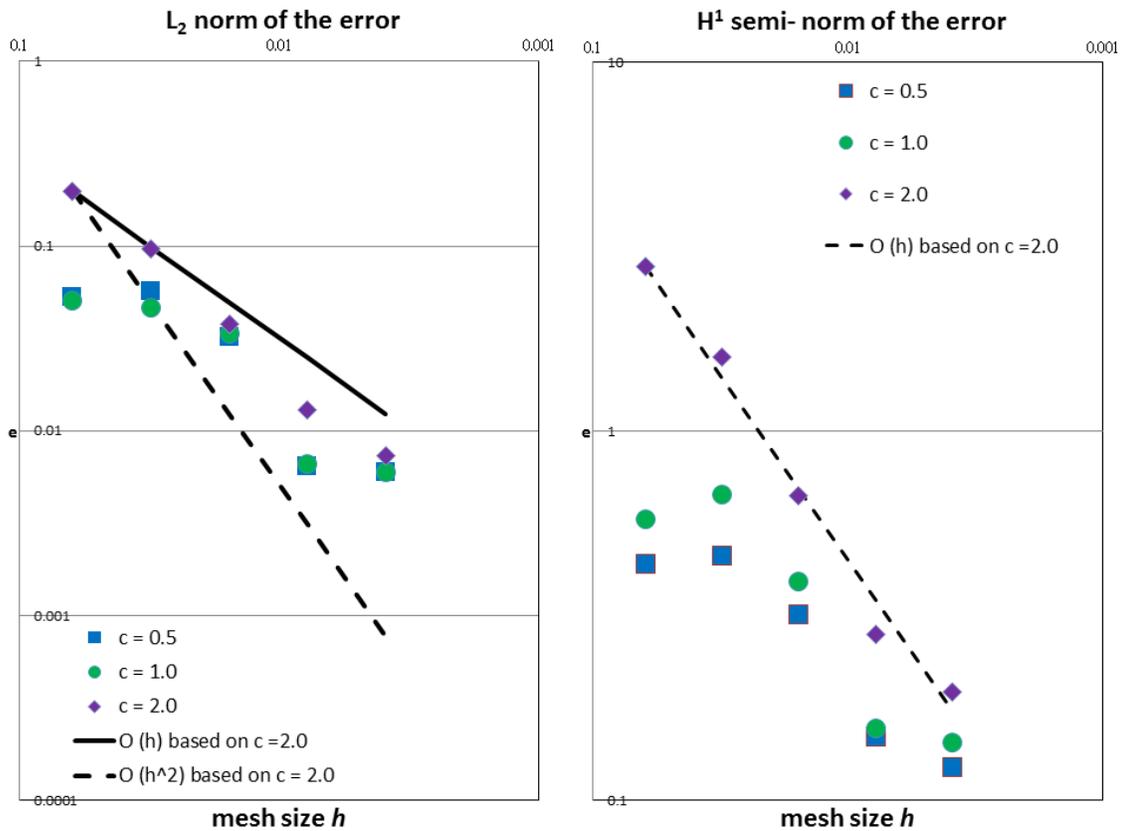


Figure 28 L₂ norm of the error and H¹ semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the constant function used as the weight in the integrals over the boundary region.

Table 7 and Figure 29 are results for the Triangle function for $k(x)$. Similar to what we have from the constant function, better results are obtained when the width of the boundary region is wide ($c = 2.0$). Its L₂ norm of the error can also be expressed as: $e \sim O(h^{1.18})$. As to its H¹ semi-norm of the error, it is $e \sim O(h^{0.94})$.

Mesh size h	$\xi = 1.0 \times h$					
	L_2 norm of the error			H^1 semi-norm of the error		
	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 0.5$	$c = 1.0$	$c = 2.0$
$1/16$	0.05467	0.05287	0.1712	0.45236	0.51191	2.24162
$1/32$	0.03255	0.03606	0.06599	0.35955	0.46914	1.06861
$1/64$	0.03239	0.03206	0.02473	0.3068	0.32692	0.43676
$1/128$	0.00599	0.00647	0.01001	0.13282	0.14356	0.21848
$1/256$	0.00603	0.00598	0.00655	0.12613	0.13406	0.16507

Table 7 Results of the Stokes flow problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.

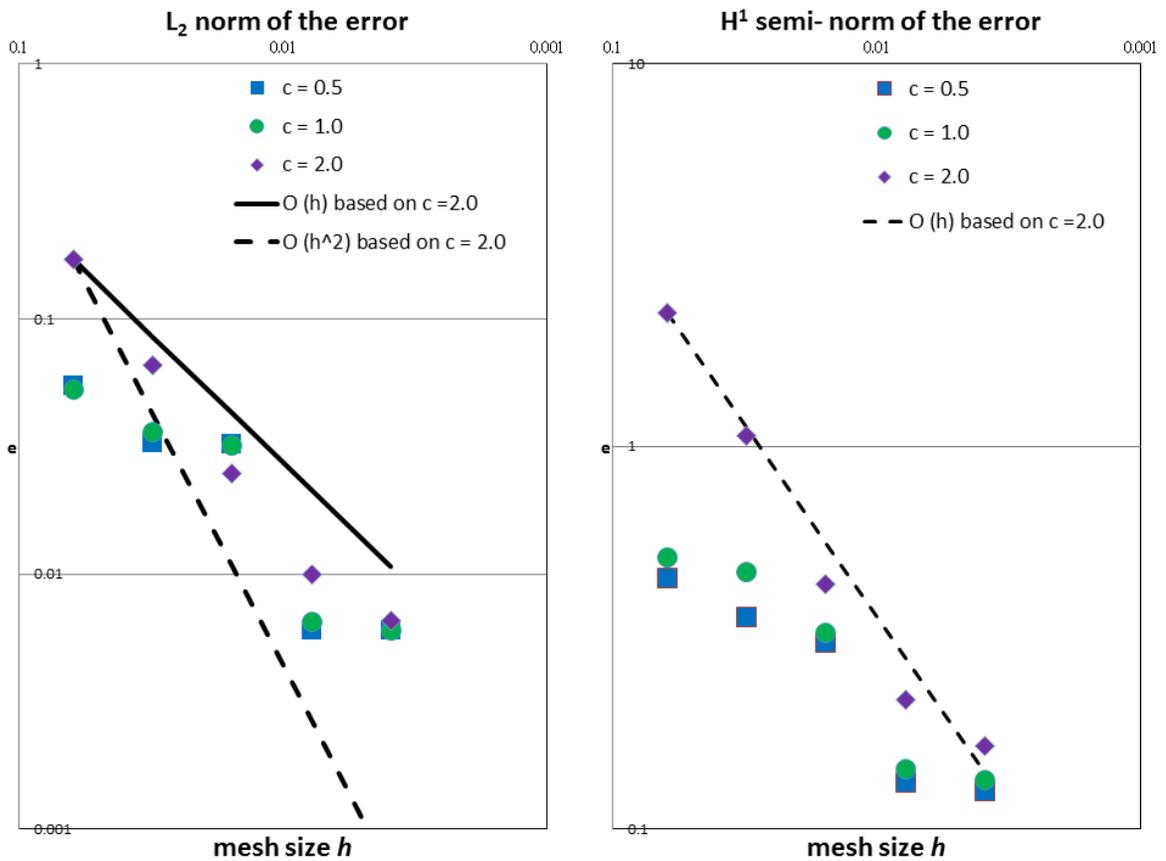


Figure 29 L_2 norm of the error and H^1 semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the triangle function used as the weight in the integrals over the boundary region.

Likewise, we use the same way to show results for the Gaussian function for $k(x)$. According to Table 8 and Figure 30, we know that under this criterion ($\xi = 1.0h$), results from the Gaussian function are worse than those from the previous two choices. Even when we set the width of the boundary region wide, such as $c = 2.0$, the obtained data point distribution is still not ideal.

Here, we still approximate errors from $c = 2.0$ based the first data point and the last data point. Thus, we can express its L_2 norm of the error as $e \sim O(h^{0.83})$. As to the H^1 semi-norm of the error, it is $e \sim O(h^{0.52})$.

We may be able to get better relation if we eliminate the first data point (the one from $h = 1/16$). However, even if we take this action, the result is still not good enough.

Mesh size h	$\xi = 1.0 \times h$					
	L_2 norm of the error			H^1 semi-norm of the error		
	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 0.5$	$c = 1.0$	$c = 2.0$
$1/16$	0.0572	0.05239	0.06152	0.4806	0.46561	0.62681
$1/32$	0.03226	0.03375	0.04805	0.33521	0.4215	0.63377
$1/64$	0.03246	0.03198	0.01854	0.29374	0.31478	0.34838
$1/128$	0.00603	0.00618	0.0083	0.13639	0.13329	0.18429
$1/256$	0.00607	0.00594	0.0062	0.13196	0.12748	0.14991

Table 8 Results of the Stokes flow problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.

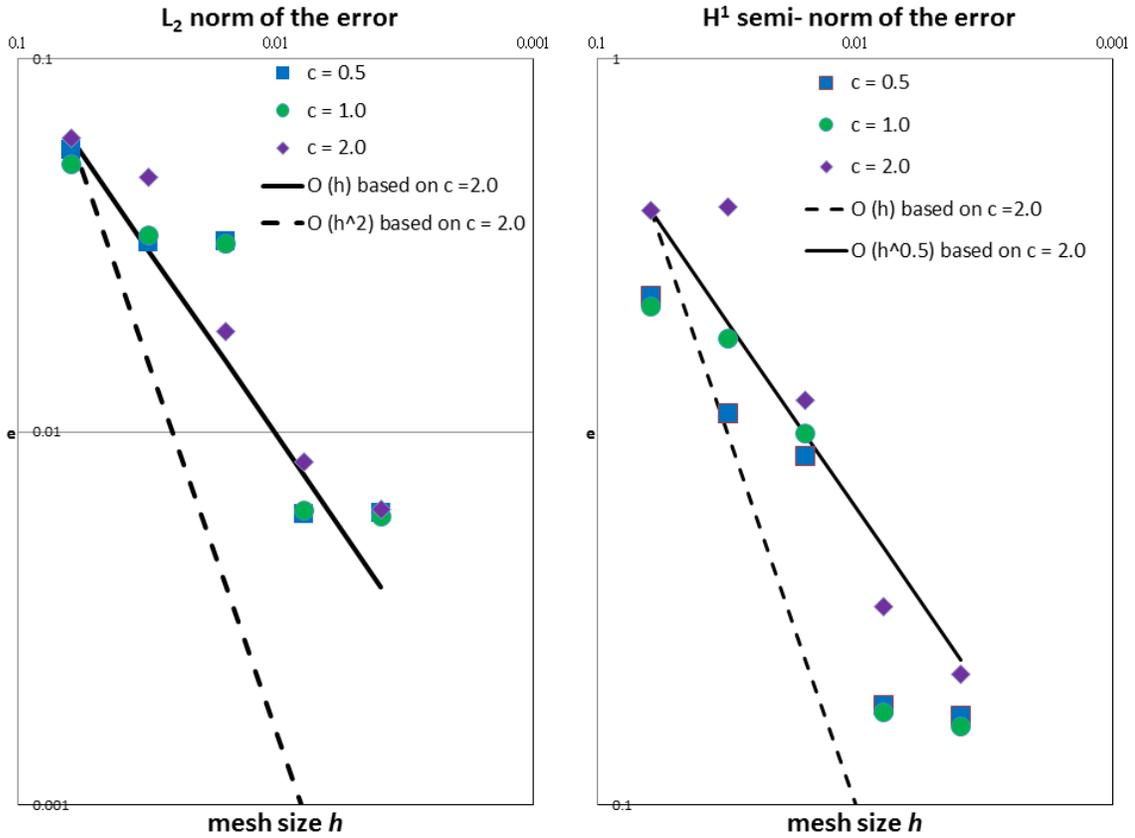


Figure 30 L_2 norm of the error and H^1 semi-norm of the error of the Stokes flow problem obtained from Strategy 2 with the Gaussian function used as the weight in the integrals over the boundary region.

3.6.3 Influence from the loop part stopping criterion ξ

Until this moment, all experimental results regarding this test case show that no matter what strategy we use, we can see stepwise data point distributions. Comparing to Strategy 1, Strategy 2 may have more room to improve this situation. By enlarging the width of the boundary region $\bar{\gamma}$, more ideal results can be obtained (see Figure 28 and Figure 29). However, for the Gaussian function, even if we take this action, results are still not good. As a result, in this subsection, we would like to see whether we can get some improvements from adjusting the stopping criterion.

Here, we first show L_2 norm of the error from Strategy 1 with $\xi = 0.25h$ in Table 9 and Figure 31. Original results from $\xi = 1.0h$ are also depicted in this figure for comparison. Apparently, the result from the smaller stopping criterion is better than the original one. However, it is still not good enough.

Mesh size h	$\xi = 1.0 \times h$		$\xi = 0.25 \times h$	
	iteration numbers	L_2 norm of the error	iteration numbers	L_2 norm of the error
$1/16$	2	0.04841	2	0.04841
$1/32$	2	0.03798	4	0.02699
$1/64$	2	0.03828	4	0.02078
$1/128$	4	0.02254	7	0.013
$1/256$	4	0.0228	7	0.01266

Table 9 L_2 norm of the error of the Stokes flow problem from Strategy 1 with different stopping criterion.

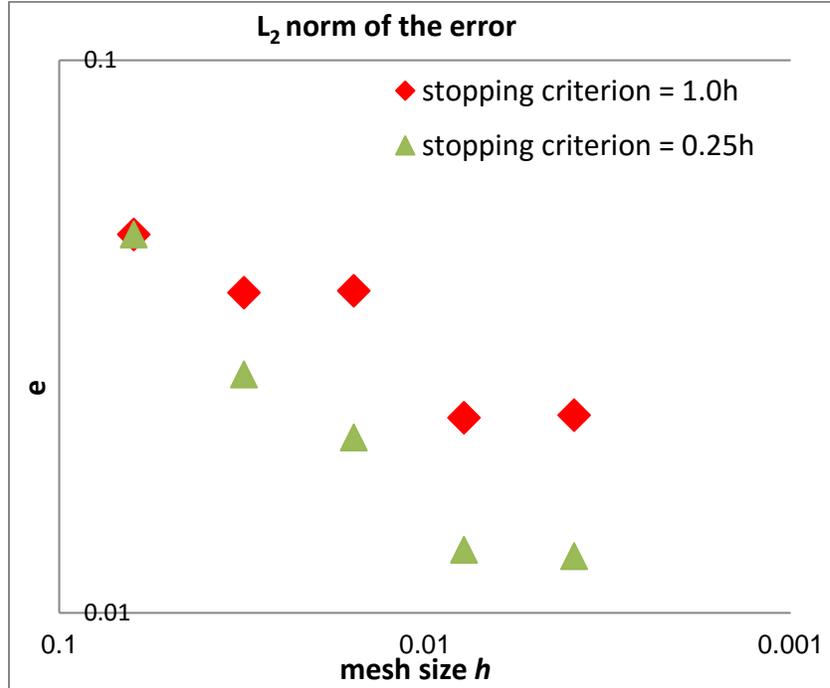


Figure 31 L_2 norm of the error of the Stokes flow problem obtained from Strategy 1 with $\xi = 1.0h$ and $\xi = 0.25h$.

Next, we demonstrate L_2 norm of the error from Strategy 2 with $\xi = 0.25h$. The $k(x)$ used for this test is the Gaussian function, whose half width of the boundary region $\bar{\gamma}$ is $2.0h$. Likewise, original results from the same setting with $\xi = 1.0h$ are also shown for comparison in Table 10 and Figure 32.

It is easy to realize that results from smaller stopping criterion are much better than those from the original setting. Its L_2 norm of the error can be approximate as $e \sim O(h^{1.3})$, which is significantly better than the first order accuracy.

There are two issues we need to mention about Table 10:

- [1] There is no data regarding $\xi = 0.25h$ at $h = 1/16$. The reason why we do not fill any information in this blank is that we cannot get solution satisfying this stopping criterion in 50 iteration steps. Since the implementation finished in this chapter will be treated as a prototype of the code for solving the Navier-Stokes problem, it is not valuable to spend too much time on solving the Stokes flow problem. More detailed description regarding this issue will be further explained in the next chapter.
- [2] Although we can get much better results from the smaller stopping criterion, the program needs more iteration to satisfy it. However, running more iteration implies that we may need more time to get converged solution. For example, when the mesh size is $1/256$, it takes 50 minutes to get the solution satisfying $\xi = 1.0h$. However, the one with $\xi = 0.25h$ needs 75 minutes. If we further decrease ξ as $h^{1.4}$, we are not able to get the solution satisfying this criterion in 40 iteration steps, which takes more than 7 hrs. It is expected that this situation will become more severe if we keep decreasing the stopping criterion.

Mesh size h	$\xi = 1.0 \times h$		$\xi = 0.25 \times h$	
	iteration numbers	L_2 norm of the error	iteration numbers	L_2 norm of the error
$1/16$	1	0.06152		
$1/32$	2	0.04805	5	0.07032
$1/64$	3	0.01854	5	0.02981
$1/128$	3	0.0083	6	0.01218
$1/256$	3	0.0062	4	0.00451

Table 10 L_2 norm of the error of the Stokes flow problem obtained from Strategy 2 with different stopping criterion.

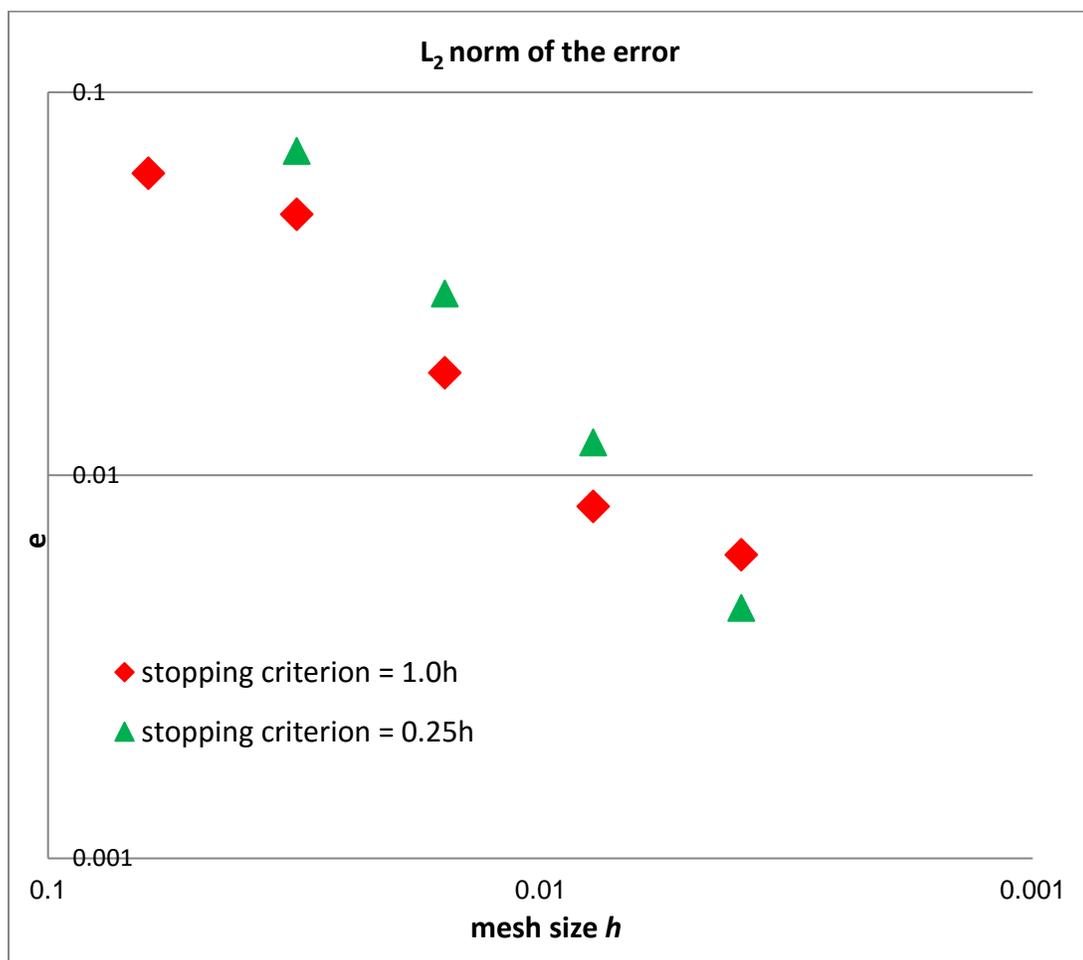


Figure 32 L_2 norm of the error of the Stokes flow problem obtained from Strategy 2 with $\xi = 1.0h$ and $\xi = 0.25h$.

3.7 Summary

Several things can be obtained from our experimental results:

- [1] Both of our two strategies for addressing the immersed boundary condition are also applicable to the Stokes flow problem.
- [2] Although our Strategy 1, which extends the constraint to be valid everywhere but equals to the desired value only in the area outside the needed domain, can get converged solution, its results are not ideal, even if we decrease the stopping criterion.
- [3] As to our second strategy, which replaces the immersed boundary by a boundary region, it is also capable of solving the Stokes flow problem. In addition, with some adjustments, we can get good results. As a result, we will choose it to be our strategy to address our ultimate topic: the Navier-Stokes flow problem.

CHAPTER IV
NAVIER-STOKES PROBLEM

4.1 Introduction

So far, we have shown that:

1. Our strategies for imposing the immersed boundary conditions are not only valid in the scalar-valued problem, but also practical in problems described by partial differential equations with multiple solution variables.
2. The modified iterative algorithm mentioned in Chapter 3 can solve the Stokes flow fictitious domain problem.

The next thing we need to do is to solve the Navier-Stokes problem, which is the ultimate goal of this research.

The governing equations for the incompressible Navier-Stokes problem are:

$$\frac{\partial \mathbf{u}}{\partial t} - \nabla \cdot (2\eta \boldsymbol{\varepsilon}(\mathbf{u})) + (\mathbf{u} \cdot \nabla) \mathbf{u} + \nabla P = f, \quad (4-1)$$

$$-\nabla \cdot \mathbf{u} = 0. \quad (4-2)$$

With the assumption that the flow is steady, the time derivative, $\frac{\partial \mathbf{u}}{\partial t}$, can be removed. Furthermore, if we assume the viscosity of the fluid is large or the fluid velocity is very small, the advection term $(\mathbf{u} \cdot \nabla) \mathbf{u}$, can be ignored and the flow becomes the Stokes flow. A discussion regarding the Stokes flow problem can be found in Chapter 3.

There are five main difficulties regarding solving large scale Navier-Stokes flow problems with complex geometries by applying the FD method:

- [1] The solution has multiple components

- [2] The solution needs to satisfy the incompressibility condition (e.g., equation (4.2)).
- [3] The solution needs to be close or equal to the desired constraint on the immersed boundary.
- [4] The advection term brings the nonlinearity (i.e., $(\mathbf{u} \cdot \nabla)\mathbf{u}$ in equation (4-1)).
- [5] The code needs to be parallelized for addressing large scale problems.

The first three items have been considered in the previous two chapters. The remaining two will be studied in this chapter.

The structure of this chapter is different from Chapter 2 or Chapter 3. In the next section, we first describe how we parallelize the code, including introducing a preconditioning technique for parallel computations. Thereafter, in Chapter 4.3, we show the corresponding weak formulation for this problem. How to linearize the advection term is explained in Chapter 4.4. Due to the appearance of this term, our iterative algorithm needs to be modified, which is also described in section 4.4. Numerical details about our implementation are mentioned in Chapter 4.5. Multiple test cases are shown in Chapter 4.6, which not only evaluate the accuracy of our code but also investigate its capability. A brief summary about this topic is provided in Chapter 4.7.

4.2 Code parallelization

Solving problems by using multiple processors is important in scientific computation, especially when the size of the problem is large. With appropriate methods, the capability of a code can be significantly increased. Many aspects of parallelizing are still subjects of current research.

While doing parallel computation, a job needs to be divided into several segments and be executed on different machines. Balancing the loading of each machine and ensure communication between them is difficult. This is also the reason why scientists/experts developed many standards to address this issue. In fact, this is a topic hard enough to be studied independently.

Fortunately, we can benefit from the friendly interface offered by deal.II. Via the interface, in our implementation, we apply Trilinos to be our library for dealing with this issue. Trilinos is a collection of open source libraries developed at Sandia National Laboratories (<https://trilinos.org>).

However, we need to address preconditioning. In the previous chapter, we proposed our iterative algorithm for solving the resulting matrix system. Then, we simply used the direct solver UMFPACK to play this role. To get better efficiency, we would like to apply a preconditioning technique to replace it.

The way we are going to use to precondition our resulting matrix system is called right preconditioning. To explain this, we first consider a linear system, $Ax = b$. While doing the right preconditioning, we first solve $AP^{-1}y = b$, then get x from $x = P^{-1}y$, where P represents the preconditioner. In case of the saddle point problem mentioned in Chapter 3.4, a block-triangular preconditioner can be used as an effective P^{-1} :

$$P^{-1} := \begin{pmatrix} \tilde{M} & B^T \\ 0 & \tilde{S} \end{pmatrix}^{-1} = \begin{pmatrix} \tilde{M} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \tilde{M} & B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \tilde{S}^{-1} \end{pmatrix}.$$

where \tilde{M} and \tilde{S} are approximations of the M block matrix and approximation of the Schur complement. According to Murphy et al. [56] and Silvester and Wathen [57], using this preconditioner leads to two or three distinct eigenvalues. As a result, the resulting linear system can be solved effectively.

The approximation of the Schur complement \tilde{S} can still be obtained by using the same method described in Chapter 3.5. However, here, it is not suitable for us to use the same way to get \tilde{M} , as what we have done in Chapter 3. In theory, the block matrix M corresponds to the symmetric Laplacian. Directly inverting the matrix M is very expensive. Even though the use of UMPACK could get good preconditioner, considered the truth that UMFPACK is a direct solver, which needs more computer resource and operations, we should better find some other ways to be in charge of this issue.

In this problem, to address this issue, we approximate this block as:

$$\tilde{M} = \begin{pmatrix} M_s & 0 \\ 0 & M_s \end{pmatrix},$$

where M_s is the Laplace matrix $M_s = (\nabla v, 2\eta \nabla v)$. Addressing this approximation is less expensive than dealing with the original one $(\varepsilon(v), \eta \varepsilon(v))$.

A more detailed description regarding implementation of this preconditioning technique in deal.II can be found in [55] and Kronbichler et al. [58].

4.3 Fictitious domain method implementation

The governing equations of the steady Navier-Stokes system are:

$$-\nabla \cdot (2\eta\varepsilon(\mathbf{u})) + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla P = f \quad \text{in } \omega, \quad (4-3)$$

$$-\nabla \cdot \mathbf{u} = 0 \quad \text{in } \omega. \quad (4-4)$$

$$\mathbf{u} = 0.0 \quad \text{at } \Gamma. \quad (4-5)$$

It is not hard to realize that the difference between this system and the one mentioned in Chapter 3 only lies in the appearance of the advection term, $(\mathbf{u} \cdot \nabla)\mathbf{u}$. If we simply treat this term as an addition, according to what we have from the previous chapter, we can express the corresponding weak formulation in the fictitious domain as:

$$2\eta(\varepsilon(v), \varepsilon(\tilde{\mathbf{u}}))_{\Omega} + (v, (\tilde{\mathbf{u}} \cdot \nabla)\tilde{\mathbf{u}})_{\Omega} - (\nabla v, \tilde{p})_{\Omega} = (v, \tilde{F})_{\Omega} - (v, \lambda)_{\gamma}, \quad (4-6)$$

$$-(q, \nabla \tilde{\mathbf{u}})_{\Omega} = 0, \quad (4-7)$$

$$(\mu, \tilde{\mathbf{u}})_{\gamma} = (\mu, 0)_{\gamma}. \quad (4-8)$$

An explanation for the parameters used in these equations can be found in Chapter 3.3. In this system, all terms related to the immersed boundary γ are addressed by our Strategy 2. The resulting matrix system is also solved by the iterative algorithm mentioned in the previous chapter. However, since the advection term is nonlinear, we need to find a way to linearize it. The iterative algorithm also needs to be adjusted. We are going to explain these two issues when we talk about the numerical details of our implementation in the next section.

4.4 Advection term linearization and Iterative algorithm modification

In our research, we choose a common way to address it: apply a Picard's iteration.

As a result, the advection term becomes:

$$(\tilde{\mathbf{u}}^{k-1} \cdot \nabla) \tilde{\mathbf{u}}^k,$$

where n represents the iteration number of the outer loop, which will be introduced later.

With this treatment, the whole resulting system is no longer nonlinear.

Next, we would like to explain how we modify the iterative algorithm mentioned in Chapter 3.4:

- (1). The linearized advection term needs to be included in step-2 and step 5.
- (2). We need to add an outer loop. Each outer loop includes an initial part and a loop part, which have been described in Chapter 3.4. Due to this action, An additional step, step-11, needs to be performed after the loop part II. In this step, we compute the following equation:

$$\frac{\|\mathbf{u}^{k-1} - \mathbf{u}^k\|_{L_1}}{\|\mathbf{u}^{k-1}\|_{L_1}}. \quad (4-9)$$

The factor of this equation represents the difference between the result obtained from the previous outer iteration, $k - 1$, and the new result from the outer iteration k .

Once the product of this step is smaller than the desired standard, τ , the final solution can be viewed as not only satisfying the governing equation and immersed boundary condition, but also converged to a fixed value.

Here, we have to mention that the outer loop needs to be executed in at least two times. This is because in the beginning, the linearized advection term will be initialized as zero (i.e., $\tilde{u}^0 = 0.0$). This term starts to cause significant influence after one outer iteration step.

To briefly sum up, at every outer iteration step, the code will first get the initial solution from the initial part. This initial solution will be modified in the loop part (it will be denoted as inner loop in the following content) for satisfying the immersed boundary condition. The obtained solution from the loop part will be checked to see whether it satisfies the desired outer loop criterion τ . If the answer is yes, then, we can say we have a converged solution.

4.5 Numerical details

In our code for solving the steady Navier-Stokes problem, to satisfy the LBB condition (see [54]), we still discretize the solutions \mathbf{u}_h , p_h and λ_h with Q2, Q1 and Q2 elements respectively. Integration is still addressed by Gaussian Quadrature.

4.6 Experimental results

We already mentioned that multiple test cases are examined to evaluate the accuracy and capability of our implementation. Results from these tests are going to be demonstrated in this section.

4.6.1 Test case 1: Parallel Poiseuille flow

Our first test case is the fully developed parallel Poiseuille flow, which has been described in Chapter 3. Although the advection term does not cause significant influence to its exact solution, due to the use of the FEM for discretization, the approximate solution will be not identical to the exact solution. Therefore, it is still an acceptable example. Moreover, due to its simplicity, the accuracy of our code can be evaluated easily. In addition, we use the Reynolds number of 100.

The weak formulation used in this test case is composed of equation (4-6), equation (4-7) and equation (4-8). As to the right hand side term $(v, \tilde{F})_{\Omega}$ of equation (4-6), it equals to 0 here. The strategy used for addressing the immersed boundary related terms is Strategy 2. According to what we have done in the previous two chapters, we know that experimental results from the three different $k(x)$ functions are similar. There may be no need to keep on testing all of them. Therefore, in this test, we only set the constant function as our $k(x)$ function.

4.6.1.1 Error analysis

Since the obtained solution profile is identical to Figure 27, we will directly talk about the obtained errors from our implementation. Here, the stopping criterion for the inner loop (i.e., loop part), ξ , is set as $1.0h$. As to the stopping criterion of the outer loop, we set it as $\tau = 0.025$. In addition, the constant c shown in the following table is still related to the width of the boundary region $\bar{\gamma}$ ($hf = c \times h$, where h represents the mesh size).

Mesh size h	$\xi = 1.0 \times h$ and $\tau = 0.025$					
	L_2 norm of the error			H^1 semi-norm of the error		
	$c = 0.5$	$c = 1.0$	$c = 2.0$	$c = 0.5$	$c = 1.0$	$c = 2.0$
$1/16$	0.02439	0.06066		0.55159	1.31233	
$1/32$	0.02078	0.02711	0.08116	0.35849	0.84887	1.65903
$1/64$	0.01118	0.01605	0.03588	0.24929	0.40909	0.79961
$1/128$	0.00558	0.00625	0.01403	0.1469	0.1692	0.33146
$1/256$	0.00511	0.00541	0.00725	0.12483	0.14946	0.21558

Table 11 Results of Test case 1 obtained from the code for solving the NS system with the constant function used as the weight in the integrals over the boundary region.

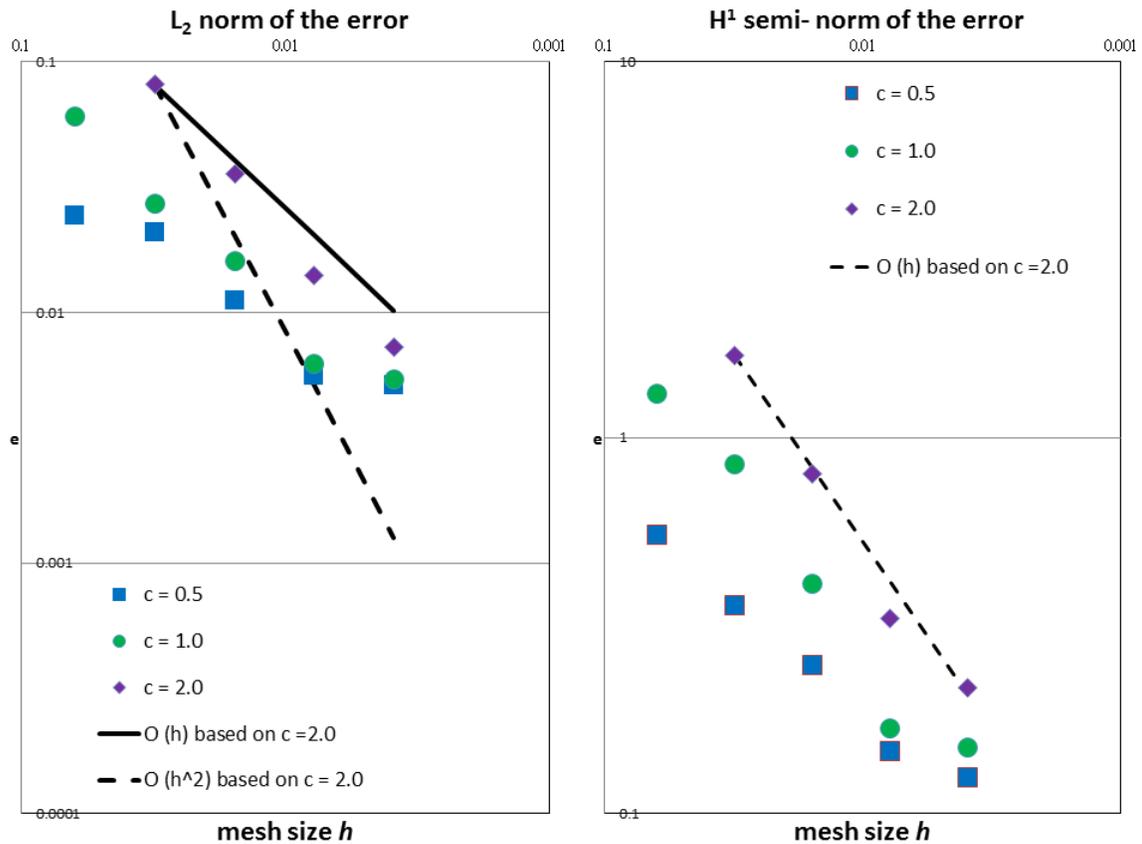


Figure 33 L_2 norm of the error and H^1 semi-norm of Test case 1 obtained from the code for solving the NS system with the constant function used as the weight in the integrals over the boundary region.

It is easy to see that when the width of the boundary region is small ($c \leq 1.0$), the obtained relationships are not ideal. The L_2 norms of the error from these settings do not significantly decrease with the mesh refinement when h is small.

In contrast, if the width is larger ($c \geq 2.0$), the data points behave in a more regular way. In this figure, all reference lines are drawn based on the first data point of $c = 2.0$. According to these lines, we know that the L_2 norm of the error obtained from $c = 2.0$ is slightly better than the 1st order accuracy ($e \sim O(h^{1.16})$) and the H^1 semi-norm of the error is close to 1st order accuracy ($e \sim O(h^{0.98})$).

In addition, we also notice that when the width of the boundary region is large ($c = 2.0$), we cannot get converged solution at coarse meshes. The reason to explain this is that since we replace the immersed boundary by the boundary region, the iteration error, which is related to the difference between the approximate solution and the desired constraints in the immersed boundary, is connected to the discretization error. In essence, errors in the wider boundary region are larger than those in the smaller boundary region since the former one contains larger area. As a result, it is hard or impossible to get converged solutions under this setting. Fortunately, this influence can be diminished with the mesh refinement.

4.6.1.2 Influence from the outer loop stopping criterion τ

According to Table 11 and Figure 11, we know that when the width of the boundary region $\bar{\gamma}$ is narrow (i.e., $c \leq 1.0$), results from $h = 1/128$ and $h = 1/256$ do not have big difference. Similar situation happened by the time we studied the Stokes

flow problem. At that time, we mentioned that we can get some improvement by decreasing the inner loop stopping criterion ξ (refer to Table 9). The tradeoff is that the program then needs more time to get solutions satisfying the smaller ξ .

To get insight of this issue, we first need to consider the three possible error resource in our implementation:

- A. Discretization error
- B. Errors from the immersed boundary condition
- C. Errors due to the tolerance of solvers, including the influence from the inner loop part, as well as the outer Picard iteration.

If one of them is bounded, then, the total error will not decrease with mesh refinement.

Influence from ξ belongs to item C. In theory, if we can get a solution which is close to the desired constraint in $\bar{\gamma}$, the solution in the needed domain will be indirectly improved as well. This is also the reason why this operation is valid in the previous chapter. However, we have to mention that the inner loop part can only be set to improve the initial solution obtained in the initial part. If the initial solution is too far from the correct answer, the program may need a lot of iterations to get an acceptable solution. This kind of situation happened when we tested the influence from the inner loop stopping criterion in Chapter 3.6.3 (refer to Table 10 with $h = 1/16$).

In Table 12, we output the needed outer iterations, needed inner iterations in each outer iteration step, inner iteration errors after each inner iteration step obtained under the setting with $c = 1.0$, $h = 1/128$ and $\xi = 1.0h$ in Table 12. It is shown that the computed inner iteration error after the first outer iteration step is much smaller than the

desired value $\xi = 1.0h = 0.0078125$. Therefore, in the 2nd outer iteration step, the loop part only needs one inner iteration step since the obtained solution after the first outer iteration step is intrinsically smaller than the designed ξ in $\bar{\gamma}$.

$c = 1.0, h = 1/128$ and $\xi = 1.0 \times h$		
Outer iteration step	Needed inner iterations	Computed inner iteration error
1	3	0.00188
2	1	6.26336×10^{-5}

Table 12 Needed inner iterations in each outer iteration step, inner iteration errors after each inner iteration step obtained under the setting with $\mathbf{c} = \mathbf{1.0}$, $\mathbf{h} = 1/128$ and $\xi = \mathbf{1.0h}$.

Therefore, what we should do is to see whether we can get the improvement from adjusting the outer loop stopping criterion τ . Table 13 shows the needed outer iterations, L_2 norm of the error and H^1 semi-norm of the error obtained under the setting with smaller outer loop stopping criterion τ . The original result with the same parameters but different τ is also shown for the comparison.

$c = 1.0$ and $\xi = 1.0 \times h$						
$\tau = 0.025$				$\tau = 0.005$		
Mesh size h	outer iterations	L_2 norm of the error	H^1 semi- norm of the error	outer iterations	L_2 norm of the error	H^1 semi- norm of the error
$1/256$	2	0.00541	0.14946	10	0.00271	0.12951

Table 13 Comparison between results from different outer loop stopping criterion τ .

To make this improvement clear, we put the L_2 norm of the error from the original setting and the new L_2 norm of the error obtained from the smaller outer loop stopping criterion ($\tau = 0.005$) in Figure 34. Apparently, if we replace the last data point by the new result, the data point distribution becomes linear. Thus, the L_2 norm of the

error from this setting ($c = 1.0$) can be improved from $e \sim O(h^{0.87})$ to $e \sim O(h^{1.11})$, which is comparable to the one from $c = 2.0$ (refer to Table 12). As to its H^1 semi-norm of the error becomes $e \sim O(h^{0.84})$.

However, similar to decreasing the inner loop stopping criterion ξ , decreasing the outer loop stopping criterion τ also takes more iterations and time. If we use the original outer loop stopping criterion (i.e., $\tau = 0.025$) to compute the setting with $c = 1.0$ and $h = 1/256$, the program needs 2.83×10^3 second to get converged solution. But, if we use the smaller outer loop stopping criterion (i.e., $\tau = 0.005$), the spending time enormously increases to 2.08×10^4 second.

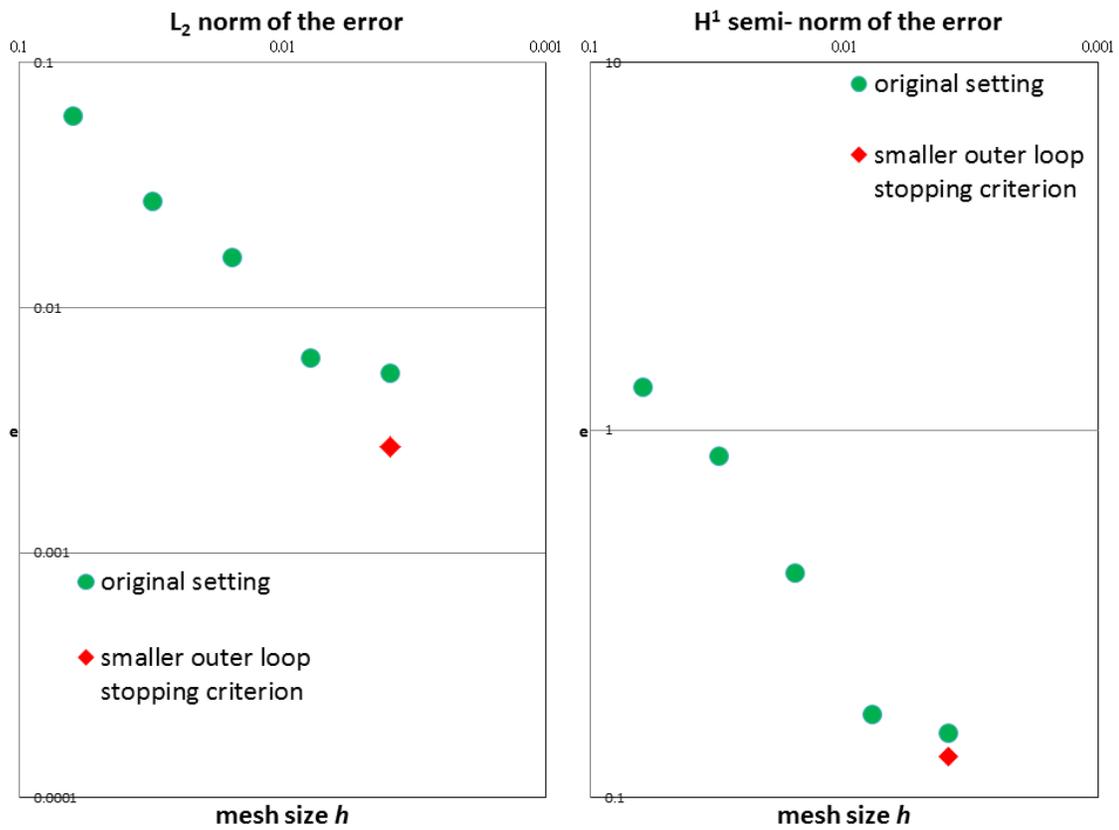


Figure 34 Improvement from the smaller outer loop stopping criterion.

4.6.1.3 Study regarding the tolerance of the solver

Recall that in our implementation, we need to solve multiple matrix systems for the solution components \mathbf{u} and p or G . For the need of parallelization, all of them are solved by applying iterative Krylov subspace solvers, such as CG, GMRES or AMG. The use of these iterative solvers implies that we need to set a tolerance for the solver to tell them when to stop the computation. For example, to a linear system, $Ax = b$, its solver tolerance can be expressed as:

$$\text{Solver tolerance} = \phi \times b,$$

where ϕ is a coefficient decided by the user.

It is not hard to realize that if we set the tolerance as a smaller value (i.e., $\phi \ll 1$), we can get more accurate solutions, compared to those with large tolerances. However, this also implies that the solver will need more time and iteration steps. In real engineering applications, how fast we can get the results is of course very important.

To study this issue, we did an experiment with the following settings: the $k(x)$ function used in the boundary region $\bar{\gamma}$ is a constant function whose width is $2h$, where h is the mesh size and equals to $1/64$. In addition, the outer loop stopping criterion, $\varepsilon = 0.025$. Coefficient for the solver tolerance, ϕ , varies from $10^{-3} \sim 10^{-6}$. The corresponding results are listed in the Table 14.

According to the table, we see that every time we divide the constant ϕ by 10, the wall clock time becomes at least 50% longer than the previous one. The comparison between $\phi = 10^{-3}$ and $\phi = 10^{-6}$ even shows that the latter setting needs approximate by 6 times longer than the former one. In our opinion, it is not necessary to spend at least

triple time for only improving 0.4% accuracy. As a result, in our implementation, we choose $\phi = 10^{-3}$.

	$\phi = 10^{-3}$	$\phi = 10^{-4}$	$\phi = 10^{-5}$	$\phi = 10^{-6}$
Processors	24	24	24	24
Wall clock time (s)	198	608	961	1.36×10^3
L ₂ norm of the error	0.01605	0.01601	0.01601	0.01601

Table 14 Results with different solver tolerance.

But, we also want to mention that it is possible that the code needs very small tolerance to address the case with very strong advection. Although we can get converged results fast by setting larger solver tolerance, the obtained error is also larger than the one with smaller tolerance. It seems likely that the required tolerance is also tied to the strength of advection. One may surmise that the tolerance for the iteration will have to be tightened for large Reynolds number problems

4.6.1.4 Efficiency of the code parallelization

To address large scale problems, we parallelize our code using the way described in Chapter 4.2. In this section, we would like to study the efficiency of the parallelization. Related settings regarding this test are: $c = 2.0$, $\xi = 1.0h$ and $h = 1/64$. Wall clock time from using different processor numbers are demonstrated in Table 15 and Figure 35.

The speed up is computed by using:

$$S_N = \frac{T_1}{T_N},$$

where T_N is the run-time executed on N processors. The efficiency, E_P , is evaluated by using:

$$E_P = \frac{S_N}{N}.$$

Processor numbers	1	2	4	8
Wall clock time (s)	2.08×10^3	1.23×10^3	624	384
Speedup (S_N)	1	1.691	3.333	5.417
Efficiency (E_P)	1	0.845	0.833	0.677
Processor numbers	16	32		
Wall clock time (s)	247	187		
Speedup (S_N)	8.421	11.123		
Efficiency (E_P)	0.526	0.348		

Table 15 Efficiency of the code parallelization.

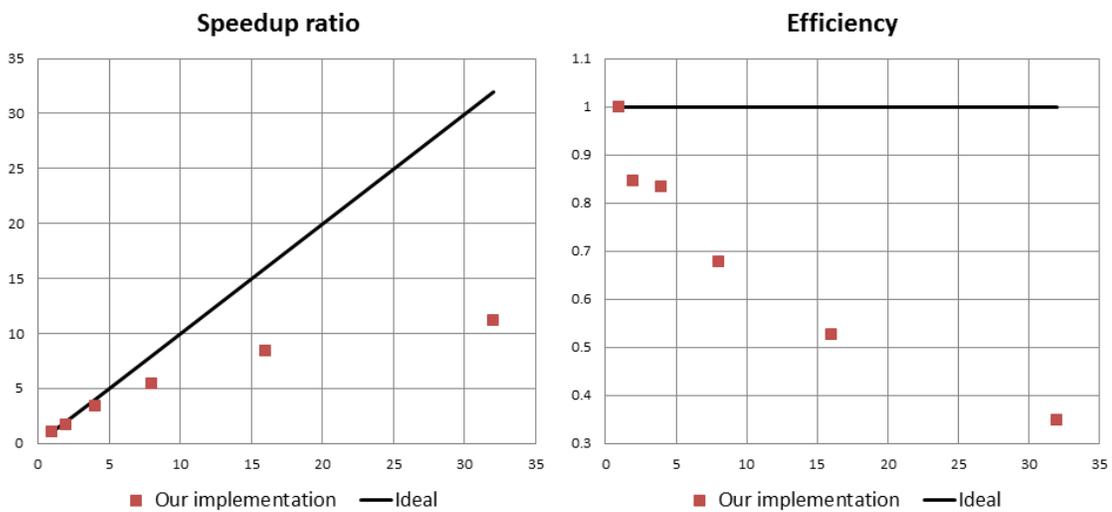


Figure 35 Speedup S_N and Efficiency E_P of our implementation.

It is shown that the speedup and efficiency of our implementation are close to the two ideal lines when N is small. However, these two decreases when N is large. This may be because even when we use multiple processors, some process in the algorithm still cannot get benefits from this operation.

In addition, if the loading is not that heavy, then, using too many processors to do it is a waste. According to [55, 58], the good speedup can be obtained if each processor is in charge of around 100,000 degree of freedoms (i.e., 100,000 dofs/processor). In this experiment, the dof is about 820000. Therefore, the speedup after 8 processors is not ideal.

Overall speaking, our test shows that our parallelized code works well and can decrease the wall clock time. However, in order to get good efficiency, when we try to run a job, we should consider a suitable processor number used for the execution.

4.6.2 Test case 2: 2D lid driven cavity flow multiple obstacles

According to our first test cases, we know that our implementation is capable of solving the steady Navier-Stokes system. In this section, we are going to use it to simulate the lid driven cavity flow with multiple obstacles.

Lid driven cavity flow is well-known benchmark for evaluating whether the result from a code is accurate (refer to Ghia et al. [59]). We choose it as our test case since its boundary condition is easy to impose.

In this test, we define the fictitious domain Ω as $\Omega := (-1,1)^2$, which can be seen in Figure 9 or Figure 36(a). The moving boundary condition is imposed on the top

surface of Ω , which can be expressed as $\mathbf{u} = 2\hat{i} + 0\hat{j}$. In addition, the no-slip boundary condition is imposed on the other three surfaces. For simplicity, we assume all obstacles are circles, whose surfaces will be viewed as the immersed boundary and addressed by Strategy 2 with the constant function used as the weight in the integrals over the boundary region.

I. One obstacle with $Re = 160$

In this portion, we assume that there is a single circular obstacle in the flow domain. The location of its center is $(0,0)$, which is also the center of the fictitious domain Ω . Its radius equals 0.5.

The following figure shows the obtained results from this setting with $Re = 160$. It is clear that our FD method successfully describes the inner object.

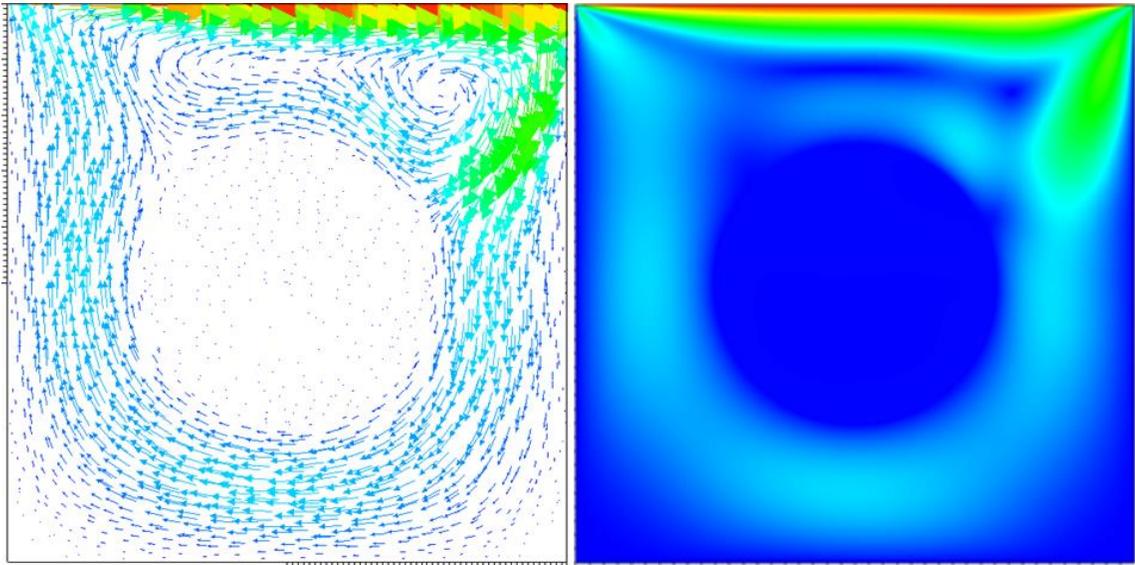


Figure 36 Velocity vector and distribution of the lid driven cavity flow with an obstacle in the flow domain.

II. Five obstacles with $Re = 200$

In this section, we increase the number of the obstacles from one to five. The radii of these five circles are all equal to 0.15. In addition, we slightly increase the Reynolds number from 160 to 200. Results from this setting are shown below.

The five obstacles can be seen clearly in the subplot regarding the velocity vector (the left subplot in Figure 37). However, they cannot be seen easily in the subplot related to the velocity distribution (i.e., the right subplot in Figure 37). This is because the flow velocity around these obstacles is close to zero. To get better resolution of this figure, we adjust the color scale so that red indicates velocity magnitude of 0.1, rather than 2 as in Figure 37. With this operation, these five obstacles can be shown obviously (see Figure 38).

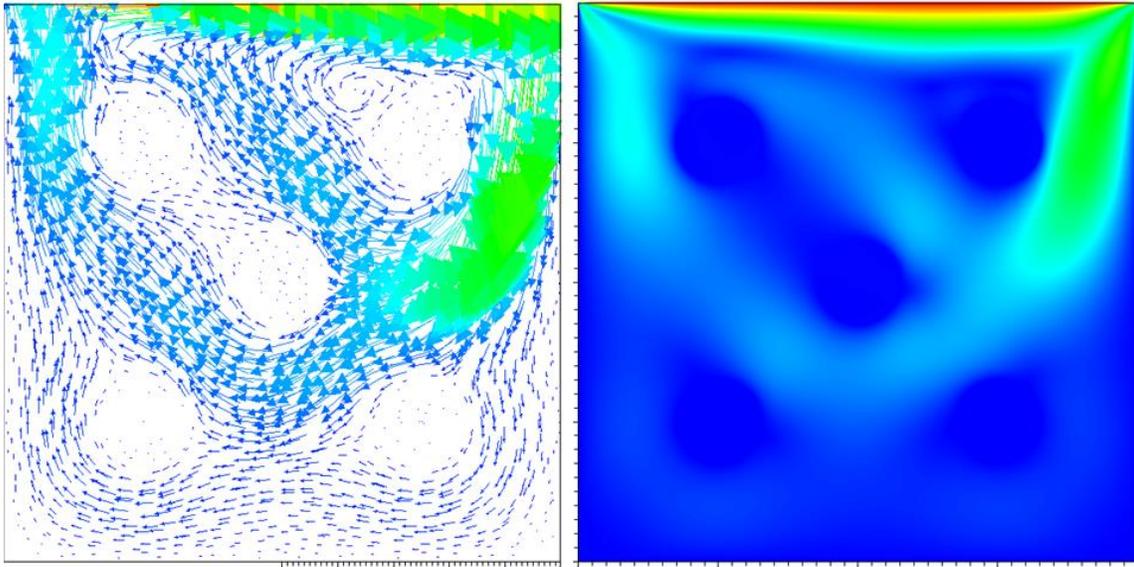


Figure 37 Velocity vector and distribution of the lid driven cavity flow with five obstacles in the flow domain.

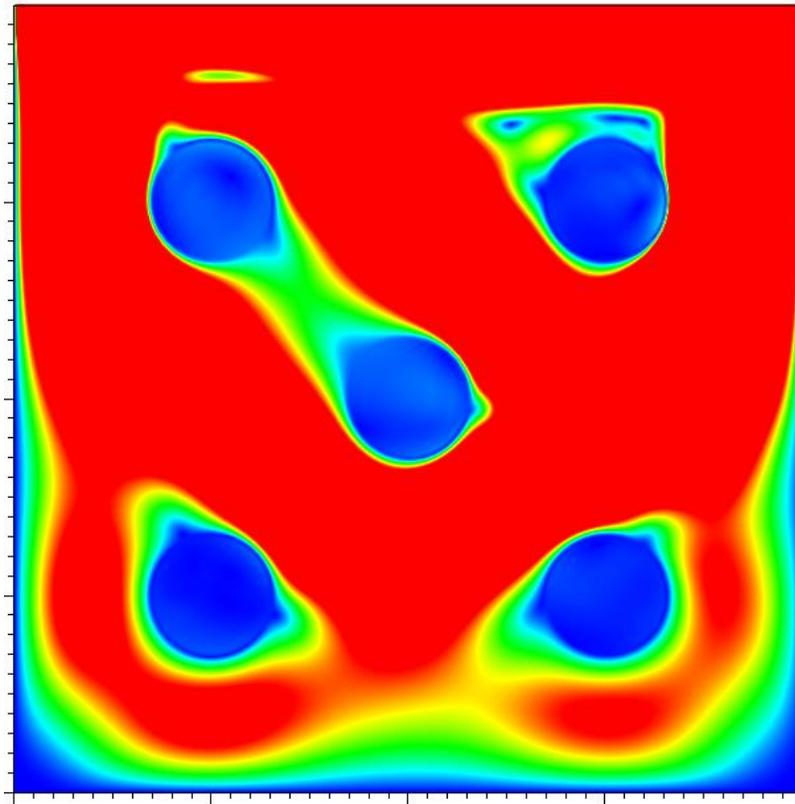


Figure 38 Adjusted velocity distribution of the lid driven cavity flow with five obstacles in the flow domain.

III. Fourteen obstacle with $Re = 400$

Now, we will make the test even more complicated. Fourteen obstacles are assigned here. Although the radii of these four circles are all equal to 0.1, due to their locations, the flow domain is no longer symmetric. Moreover, we increase the Reynolds number to 400 in this test. The corresponding velocity vector field and distribution are demonstrated in Figure 39.

Again, in the subplot regarding the velocity distribution, we adjust the color scale so that red indicates velocity magnitude of 0.1, rather than 2 as in Figure 39. Thus, these fourteen obstacles can be seen easily in this figure.

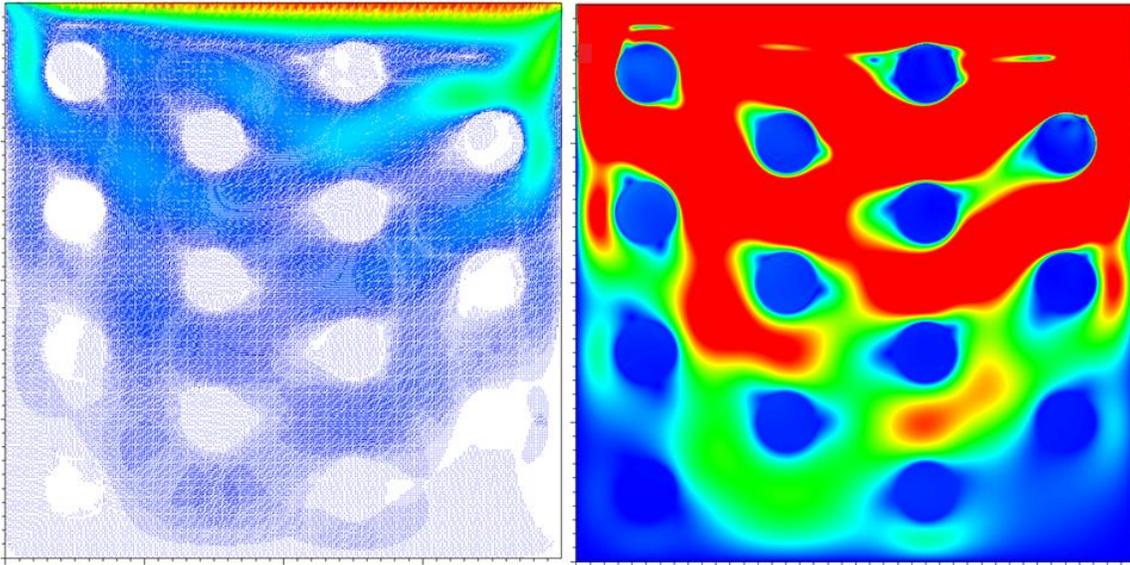


Figure 39 Velocity vector and distribution of the lid driven cavity flow with fourteen obstacles in the flow domain.

4.6.3 Test case 3: 3D channel flow with multiple obstacles

In this section, we are going to use our code to simulate a 3D channel flow with obstacles in it. Similar to what we have done in Test case 3, we simplify the shape of the obstacles as balls.

The fictitious domain used in this test case is a cube (see Figure 40). Its edge length is equal to two. The flowing fluid moves from the left face to the right face. For simplicity, we set both of them as:

$$\mathbf{u} = 2.0 \hat{i} + 0.0 \hat{j} + 0.0 \hat{k} .$$

All the other settings regarding this test case are identical to those in Test case 3.

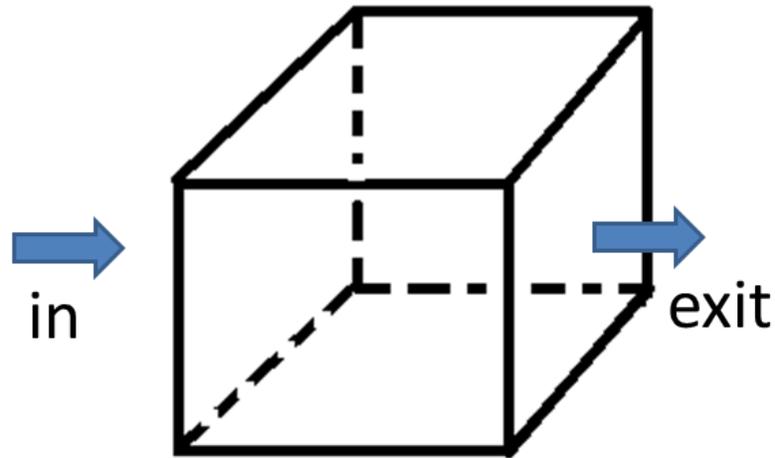


Figure 40 Fictitious domain for Test case 3.

I. One obstacle with $Re = 50$

Here, let us assign a ball in the fictitious domain. The radius of the ball is equal to 0.3 and its center locates at the center of Ω . The velocity vector and distribution on the center plane are sliced out in the following figure.

In this figure, we can see a circle in the domain, which should be the projection of the obstacle on this plane. However, we can also see nonzero velocity vectors in the subplot in the left subplot of Figure 41. The subplot regarding the velocity distribution reflects this situation as well. Even though the value in the obstacle is not what we really care about, we still can improve it by decreasing the outer loop stopping criterion τ . Figure 42 shows the result obtained when the outer loop stopping criterion is 0.01. Compared to Figure 41, whose outer loop stopping criterion, τ , is equal to 0.025, result in Figure 42 looks better.

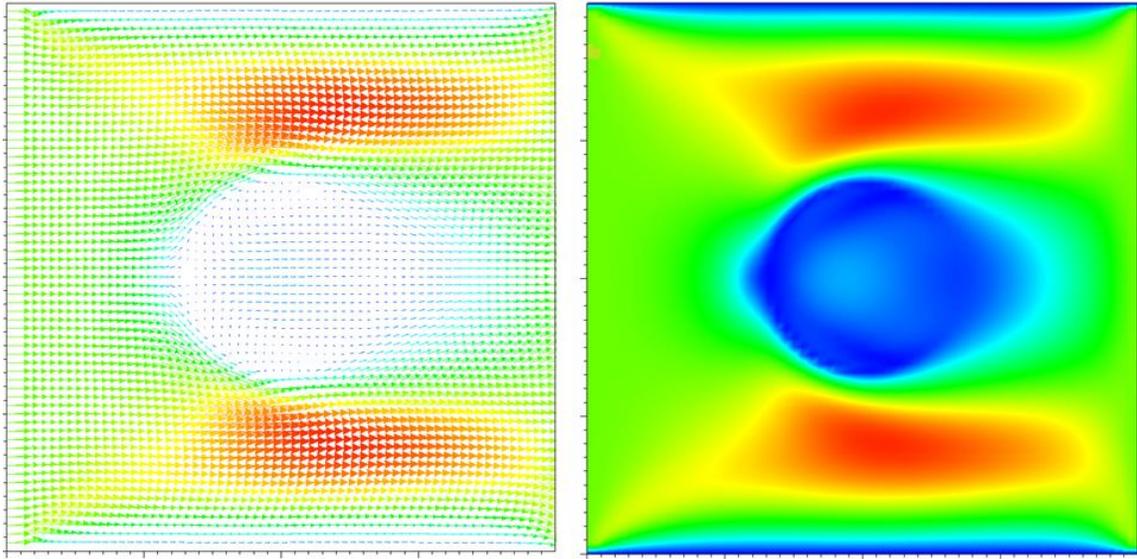


Figure 41 Velocity vector and distribution of the channel with an obstacle in the flow domain when the outer loop stopping criterion is 0.025.

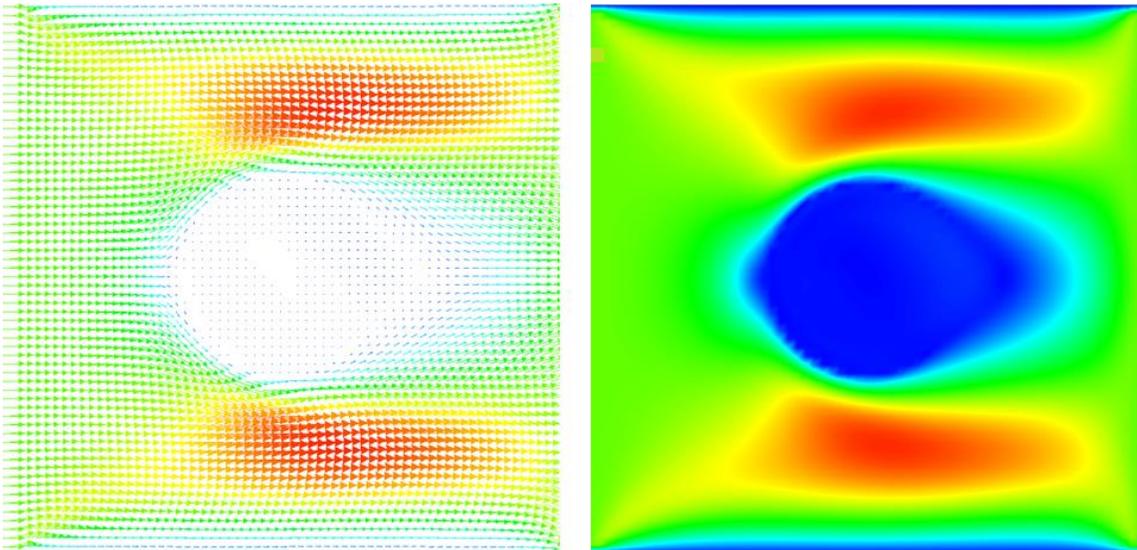


Figure 42 Velocity vector and distribution of the channel with an obstacle in the flow domain when the outer loop stopping criterion is 0.01.

II. Two obstacles with $Re = 60$

Next, we simulate two balls with different size in the flow domain, whose radii are 0.2 and 0.1 respectively. The center of the first ball locates at $(-0.5, -0.5, -0.5)$ and the other locates at $(0.5, 0.5, 0.5)$. In addition, we also slightly increase the Reynolds number to 60.

Since the locations of these two obstacles are not on the center plane of Ω , we output the slice composed of points A, B, C and D (refer to Figure 43). Figure 44 are the corresponding velocity vector and velocity distribution. From this figure, we can see projections of these two balls on the output slice, which have very small values within it. Our previous experiment suggests that we could further improve the results by decreasing τ .

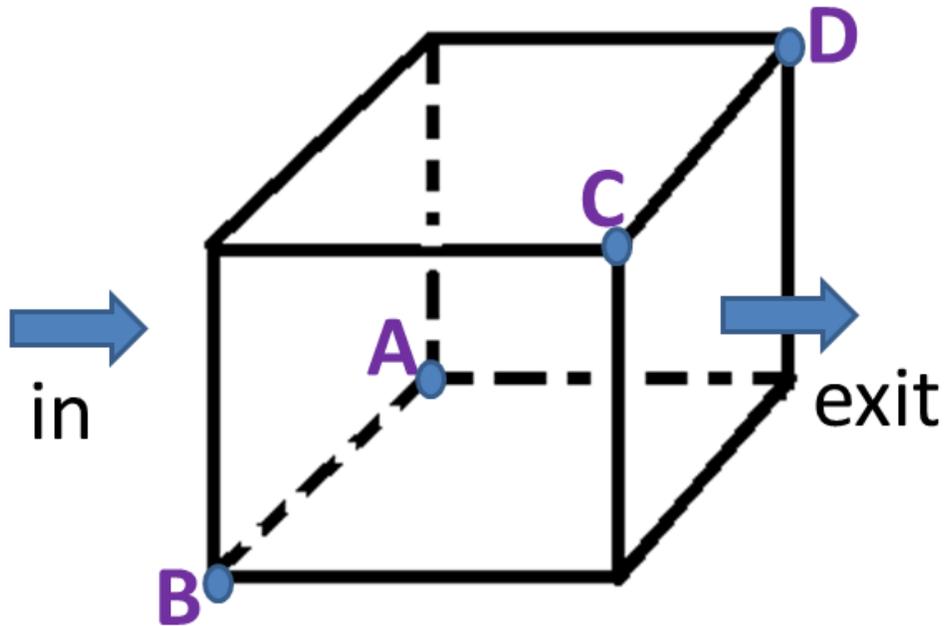


Figure 43 Slice output for demonstrating a flow passing around two balls.

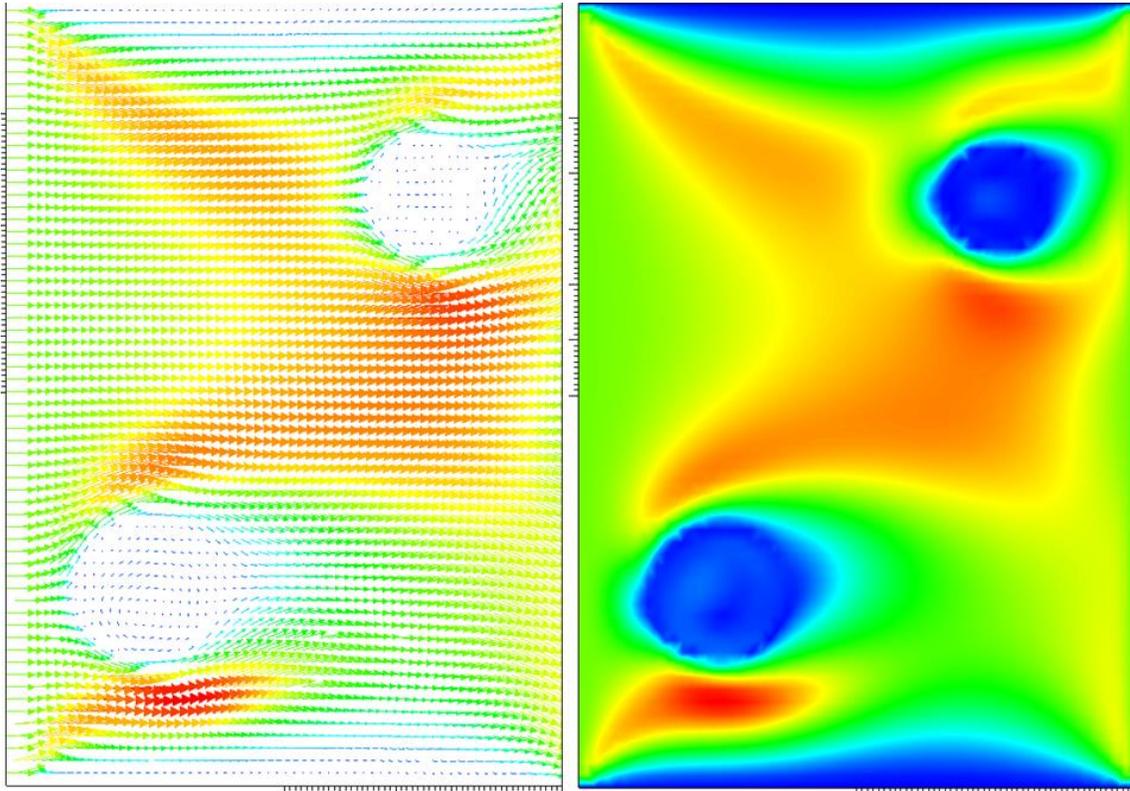


Figure 44 Velocity vector and velocity distribution of the channel with two obstacles in the flow domain.

III. Three obstacles with $Re = 70$

In this setting, three balls with different size are arranged in the domain. Their locations are $(-0.5, -0.5, -0.5)$, $(0.0, 0.0, 0.0)$ and $(0.5, 0.5, 0.5)$. Moreover, we increase the Reynolds number from 60 to 70. Likewise, we slice out the plane composed of points A, B, C and D (refer to Figure 43). Again, flow passing around these three balls can be seen clearly in Figure 45.

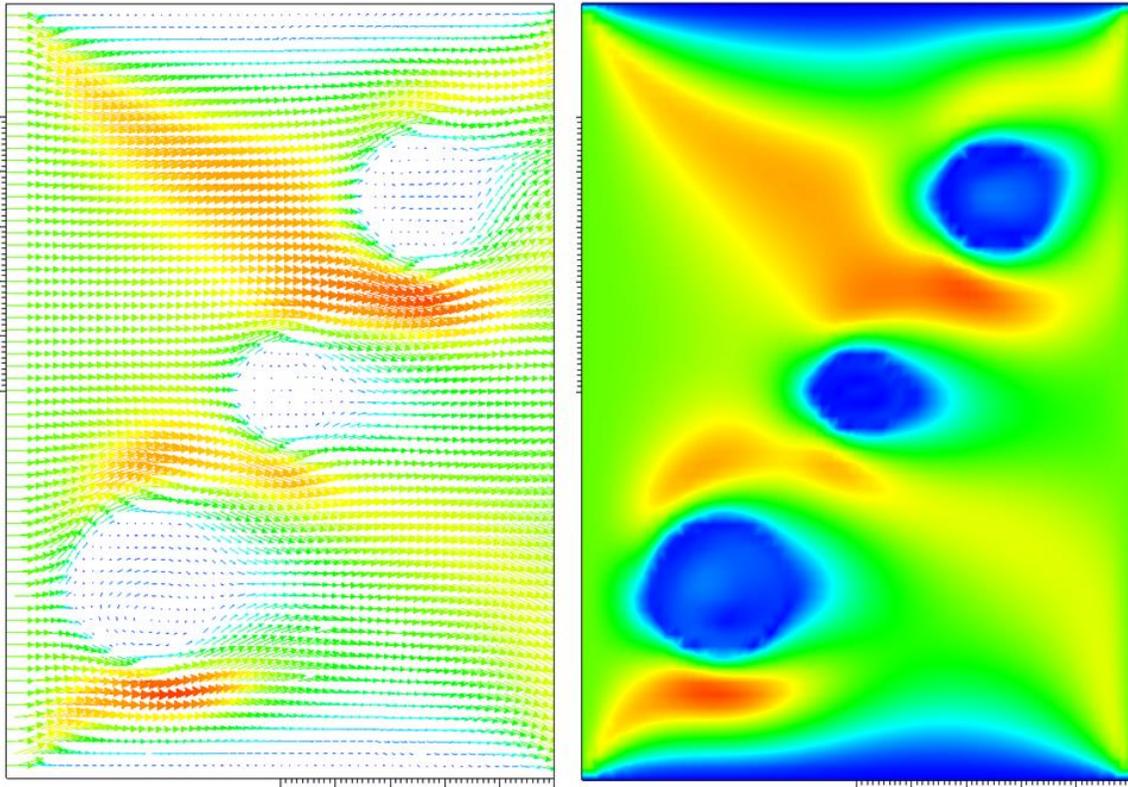


Figure 45 Velocity vector and velocity distribution of the channel with two obstacles in the flow domain.

Overall speaking, it looks like our implementation is capable of capturing the complex physical phenomena in this 3D case. However, one may notice that our results are obtained on a coarse mesh ($h = 1/32$) and low Reynolds number ($Re \leq 70$).

4.7 Summary

In this chapter, several test cases are used to evaluate our implementation for solving large scale steady Navier-Stokes problems.

A preliminary investigation is accomplished in our first case. In this test case, we have shown:

- I. Strategy 2, which replaces the immersed boundary by a boundary region, still works well in steady Navier-Stokes problem. In addition, with some adjustments, we can have results whose L_2 norm of the error is slightly better than 1st order accuracy and H^1 semi-norm of the error close to 1st order accuracy.
- II. Our code can be executed on multiple processors, which enhances the possibility of solving large scale problems.

Next, we used the code to simulate 2D and 3D cases with multiple obstacles in the flow domain. Corresponding results both look reasonable, which further demonstrates the capability of our implementation.

However, our experimental results are confined to low Re flows. There are two reasons to explain this:

- I. Due to the appearance of the advection term, the resulting coefficient matrix is not symmetric. If Re is large, which means the advection term is strong, then we may need to take some actions, such as applying the streamline upwind/Petrov Galerkin (SUPG) method or adding artificial viscosity to keep stability. Details about these stabilized methods can be in Bonet Chaple [60] and Franca et al. [61].
- II. In our implementation, we used Krylov subspace methods as our solvers. The convergence of these solvers highly depends on the preconditioner. Unfortunately, according to [55], until this moment, there is no efficient preconditioner to solve flow problems which are dominated by the advection term.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this research, we studied the fictitious domain method, in order to solve large scale flow problems without causing too much burden to researchers and computers.

Two strategies were proposed to deal with the immersed boundary condition, which is the critical issue regarding this approach. In Strategy 1, the constraint is extended everywhere in the fictitious domain but only equals the immersed boundary condition in the area outside the needed domain. As to Strategy 2, it replaces the immersed boundary by a boundary region. An additional-weighting function is added to validate this replacement.

Three different problems were designed and tested for evaluation. From the Poisson problem, which only has one variable:

- I. Strategy 1 is easy to apply. Its corresponding experimental results shows that it is capable of addressing this Poisson problem and could get approximate 1st order accuracy in its L_2 norm of the error.
- II. Strategy 2, in contrast to Strategy 1, is more complex. However, it can get solutions with better accuracy.

From the Stokes flow problem, whose solution has multiple variables:

- I. Reasonable results can be obtained by applying the proposed iterative algorithm.

II. Both of these two strategies are feasible. However, the obtained data shows that Strategy 1 can solve the problem, its error is not ideal. On the other hand, Strategy 2 not only could get correct solution profile, but also keeps acceptable accuracy, which is a more suitable choice to be implemented in the code for solving Navier-Stokes flow problems.

III. The three different functions used as the weighting function in Strategy 2 has similar data point distributions and accuracy in L_2 norm of the error and H^1 norm of the error.

From the Navier-Stokes problem, our ultimate goal:

- I. Our fictitious domain method is able to simulate steady Navier-Stokes problem with complex geometries, even though it is confined in low Reynolds numbers.
- II. Via parallelizing the code and applying effective Krylov subspace methods, problems with large scale can be solved efficiently.

5.2 Recommendations for future work

Although our experimental results proved that our fictitious domain method is able to solve some flow problems, it is still far from solving real engineering problems.

To complete this method, several possible research directions include:

- I. Applying techniques to stabilize the resulting linear system. Considering the situation that the flow may have a strong convection term, it is necessary to introduce some approaches to address the instability of the system.
- II. Extending our strategies for addressing the immersed boundary condition to transient Navier-Stokes problems. In addition to simulating flow problems with complex

geometries, the fictitious domain method also has the potential to address fluid structure interaction problems (FSI problems). In this kind of problems, the irregular boundary usually moves with time. Therefore, research regarding this extension is valuable in practice.

III. Including the influence from energy equation. Sometimes, flow problems are accompanied with heat convection. Combining the energy equation with the existing system can increase the value of this method.

REFERENCES

- [1] Lee, K. D., and Rubbert, P. E., 1981, "Transonic flow computations using grid systems with block structure," Seventh International Conference on Numerical Methods in Fluid Dynamics Stanford University, Stanford, California, pp. 266-271.
- [2] Shapira, Y., 2006, Solving PDEs in C++: numerical methods in a unified object-oriented approach, SIAM, Philadelphia, U.S.
- [3] Banek, J. A., Steger, J. L., and Dougherty, F. C., 1983, "A flexible grid embedding technique with application to the Euler equations," AIAA Paper 83-1944, pp. 373-382.
- [4] Banek, J. A., Buning, P. G., and Steger, J. L., 1985, "A 3D Chimera Grid Embedding technique," AIAA Paper, 85-1523, pp. 322-331.
- [5] Petersson, N. A., 1999, "Hole-cutting for three-dimensional overlapping grids," SIAM J. Sci. COMPUT., 21(2), pp. 646-665.
- [6] Hubbard, B. J., and Chen, H. C., 1994, "A Chimera scheme for incompressible viscous flows with applications to submarine hydrodynamics," 25th AIAA Fluid Dynamics Conference, Colorado Spring, CO, Paper no. 94-2210.
- [7] Chu, H.-C., 2012, "Numerical simulation of flow and heat transfer in internal multi-pass cooling channel within gas turbine blade," Master of Science, Texas A&M University, College Station, TX.
- [8] Kim, H. K., Kim, S. J., Yoon, K. H., Kang, H. S., and Song, K. N., 2001, "Fretting wear of laterally supported tube," WEAR, 250, pp. 535-543.
- [9] Glowinski, R., Pan, T. W., and Periaux, J., 1994, "A fictitious domain method for Dirichlet problem and applications," Computer Methods in Applied Mechanics and Engineering 111, pp. 283-303.
- [10] Quarteroni, A., and Valli, A., 2005, Domain decomposition methods for partial differential equations, Clarendon Press, Oxford.
- [11] Reddy, J. N., 1986, Applied functional analysis and variational methods in engineering, McGraw-Hill, Inc, New York City, U.S.
- [12] Zhou, G., and Saito, N., 2014, "Analysis of the fictitious domain method with penalty for elliptic problems," Japan J. Indust. Appl. Math, 31, pp. 57-85.

- [13] Saito, N., and Zhou, G., 2015, "Analysis of the fictitious domain method with an L^2 -penalty for elliptic problems," Numerical Functional Analysis and Optimization, 36, pp. 501-527.
- [14] Ramière, I., Belliard, M., and Angot, P., 2005, "On the simulation of nuclear power plant components using a fictitious domain approach," The 11th International Topical Meeting on Nuclear Thermal-Hydraulics (NURETH-11)Avignon, France, pp. 1-17.
- [15] Zhu, J., and Ma, Y., 2009, "Fictitious domain method with penalty for an incompressible fluid," Numerical Methods for Partial Differential Equations, 26(1), pp. 229-238.
- [16] Bryan, K., and Shubberu, Y., "Penalty functions and constrained optimization," Dept. of Mathematics, Rose-Hulman Institute of Technology.
- [17] Glowinski, R., Pan, T. W., and Periaux, J., 1994, "A fictitious domain method for external incompressible viscous flow modeled by Navier-Stokes equations," Computer Methods in Applied Mechanics and Engineering, 112, pp. 133-148.
- [18] Glowinski, R., Pan, T. W., and Periaux, J., 1995, "A Lagrange multiplier/fictitious domain method for the Dirichlet problem - generalization to some flow problems," Japan J. Indust. Appl. Math., 12, pp. 87-108.
- [19] Glowinski, R., Pan, T. W., and Periaux, J., 1998, "Distributed Lagrange multiplier methods for incompressible viscous flow around moving rigid bodies," Computer Methods in Applied Mechanics and Engineering, 151, pp. 181-194.
- [20] Glowinski, R., Pan, T. W., Hesla, T. D., Joseph, D. D., and Periaux, J., 1999, "A distributed Lagrange multiplier/fictitious domain method for flows around moving rigid bodies: application to particulate flow," Int. J. Numer. Meth. Fluids, 30, pp. 1043-1066.
- [21] Yu, Z., 2005, "A DLM/FD method for fluid/flexible-body interactions," Journal of Computational Physics, 207, pp. 1-27.
- [22] Burman, E., and Hansbo, P., 2010, "Fictitious domain finite methods using cut elements: I. a stabilized Lagrange multiplier method," Comput. Methods Appl. Mech. Engrg, 199(41-44), pp. 2680-2686.
- [23] Burman, E., and Hansbo, P., 2012, "Fictitious domain finite element methods using cut elements: II. a stabilized Nitsche method," Applied Numerical Mathematics, 62, pp. 328-341.

- [24] Haslinger, J., and Renard, Y., 2009, "A new fictitious domain approach by the extended finite element method," *SIAM J. Numer. Anal.*, 47(2), pp. 1474-1499.
- [25] Court, S., Fourniè, M., and Lozinski, A., 2014, "A fictitious domain approach for the stokes problem based on the extended finite element method," *Int. J. Numer. Meth. Fluids*, 74, pp. 73-99.
- [26] Court, S., Fourniè, M., and Lozinski, A., 2014, "A fictitious domain approach for fluid-structure interactions based on the extended finite element method," <http://arxiv.org/abs/1401.0559>.
- [27] Lai, M. C., and Peskin, C. S., 2000,, "An immersed boundary method with formal second-order accuracy and reduced numerical viscosity," *Journal of Computational Physics*, 160, pp. 705-719.
- [28] Peskin, C. S. A. N., 2002, "The immersed boundary method," *Acta Numerica*, pp. 479-517.
- [29] Jendoubi, A., Yakoubi, D., Fortin, A., and Tibirna, C., 2014, "An immersed boundary method for fluid flows around rigid objects," *Int. J. Numer. Meth. Fluids*, 75, pp. 63-80.
- [30] Loon, R. V., Anderson, P. D., van de Vosse, F. N., and Sherwin, S. J., 2007, "Comparison of various fluid-structure interaction for deformable bodies," *Computers and Structures*, 85(11-14), pp. 833-843.
- [31] Mark, A., Svenning, E., and Edelvik, F., 2013, "An immersed boundary method for simulation of flow with heat transfer," *International Journal of Heat and Mass Transfer*, 56, pp. 424-435.
- [32] Wang, X., and Liu, W. K., 2004, "Extended immersed boundary method using FEM and RKPM," *Comput. Method Appl. Mech. Engrg*, 193, pp. 1305-1321.
- [33] Grama, A., Gupta, A., Karypis, G., and Kumar, V., 2003, *Introduction to parallel computing*, Addison-Wisley, Boston.
- [34] Smith, I. M., Griffiths, D. V., and Margetts, L., 2013, *Programming the finite element method*, Wiley.
- [35] Barney, B., 2012, "Message Passing Interface (MPI)," <https://computing.llnl.gov/tutorials/mpi/>.

- [36] Blasco, J., Calzada, M. C., and Marin, M., 2009, "A fictitious domain, parallel numerical method for rigid particulate flows," *Journal of Computational Physics*, 228, pp. 7596-7613.
- [37] Wachs, A., 2011, "PeliGRIFF, a parallel DEM-DLM/FD direct numerical simulation tool for 3D particulate flows," *J Eng Math*, 71, pp. 131-155.
- [38] Chueh, C. C., Djilali, N., and Bangerth, W., , Vol.35, pp. B149-175., 2013, "An h-adaptive operator splitting method for two-phase flow in 3D heterogeneous porous media," *SIAM Journal on Scientific Computing*, 35(1), pp. B149-B175.
- [39] Freyer, M., Ale, A., Schulz, R., Zientkowska, M., Ntziachristos, V., and Englmerier, K.-H., 2010, "Fast automatic segmentation of anatomical structures in x-ray computed tomography images to improve fluorescence molecular tomography reconstruction," *Journal of Biomedical Optics*, 15, pp. 036006/036001-036008.
- [40] Kronbichler, M., Heister, T., and Bangerth, W., 2012, "High accuracy mantle convection simulation through modern numerical methods," *Geophysical Journal International*, 191, pp. 12-29.
- [41] Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., Maier, M., Turcksin, B., and Wells, D., 2016, "The deal.II library, version 8.4," *Journal of Numerical Mathematics*, 24(3), pp. 135-141.
- [42] Kovasznay, L. I. G., 1948, "Laminar flow behind a two-dimensional grid," *Proc. Camb. Phil. Soc.*, 44, pp. 58-62.
- [43] Biswas, G., Breuer, M., and Durst, F., 2004, "Backward-facing step flows for various expansion ratios at low and moderate Reynolds numbers" *Journal of Fluids Engineering*, 126, pp. 362-374.
- [44] Roache, P. J., 2002, "Code verification by the method of manufactured solutions," *Journal of Fluids Engineering*, 124, pp. 1-7.
- [45] Buffat, M., and Penven, L. L., 2011, "A spectral fictitious domain method with internal forcing for solving elliptic PDEs," *Journal of Computational Physics*, 230, pp. 2433-2450.
- [46] Sauer, T., 2011, *Numerical analysis*, Pearson, New Jersey.
- [47] Davis, T. A., 2006, *Direct methods for sparse linear systems*, SIAM, Philadelphia.
- [48] Panton, R. L., 2005, *Incompressible flow*, Johns Wiley & Sons, New York City.

- [49] Benzi, M., Golub, G. H., and Liesen, J., 2005, "Numerical solution of saddle point problems," *Acta Numerica*, pp. 1-137.
- [50] Glowinski, R., 1985, "Viscous flow simulation by finite element methods and related numerical techniques," *Progress and Supercomputing in Computational Fluid Dynamics*, pp. 173-210.
- [51] Glowinski, R., and Pironneau, O., 1992, "Finite element methods for Navier-Stokes equations," *Annu. Rev. Fluid Mech.*(24), pp. 167-204.
- [52] Glowinski, R., and Le Tallec, P., 1989, *Augmented lagrangian and operator-splitting methods in nonlinear mechanics*, Society for Industrial and Applied Mathematics, Philadelphia.
- [53] Saad, Y., 2003, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia.
- [54] Girault, V., and Raviart, P. A., 1986, *Finite element methods for navier-stokes equations*, Springer, Berlin, Germany.
- [55] Heister, T., 2011, "A massively parallel finite element framework with application to incompressible flows," PhD, University of Göttingen, Göttingen.
- [56] Murphy, M. F., Golub, G. H., and Wathen, A. J., 2000, "A note on preconditioning for indefinite linear systems," *SIAM J. Sci. COMPUT.*, 21, pp. 1969-1972.
- [57] Silvester, D., and Wathen, A. J., 1994, "Fast iterative solution of stabilised Stokes systems part II: using general block preconditioners," *SIAM J. Numer. Anal.*, 31(5), pp. 1352-1367.
- [58] Kronbichler, M., Heister, T., and Bangerth, W., 2012, "High accuracy mantle convection simulation through modern numerical methods," *Geophys. J. Int.* , 191, pp. 12-29.
- [59] Ghia, U., Ghia, K. N., and Shin, C. T., 1982, "High-Re solutions for incompressible flow using the navier-stokes equations and a multigrid method," *Journal of Computational Physics* 48, pp. 387-411.
- [60] Bonet Chaple, R. P., 2006, "Numerical stablization of convection-diffusion-reaction problmes," Technical report.
- [61] Franca, L. P., Hauke, G., and Masud, A., 2004, *Finite element methods: 1970's and beyond*, International Center for Numerical Methods in Engineering (CIMNE), Barcelona, Spain.