# Development of preconditioning strategies for simulating very large floating structures

Master Thesis

*Submitted in partial fulfilment of the requirements of*
*BITS F422T Thesis*

*By*

Shreyas Prashanth
ID No. 2020B4A42096G

*Under the supervision of:*

Dr. Alexander Heinlein
&
Dr. Oriol Colomés


BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS


DELFT UNIVERSITY OF TECHNOLOGY
May 2025

# Declaration of Authorship

I, Shreyas PRASHANTH, declare that this Master Thesis titled, 'Development of preconditioning strategies for simulating very large floating structures' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: May, 2025

# *Acknowledgements*

I would like to thank Prof. Alexander Heinlein for giving me this opportunity to pursue this thesis under his supervision. His extensive support, guidance, and feedback have been invaluable. Thanks to Prof. Alexander Heinlein, I have developed a deep interest in domain decomposition methods and high-performance computing.

I am also grateful to Prof. Oriol Colomes for co-supervising my thesis. His subject matter expertise on offshore structures and Gridap has been very helpful in this project. I would like to thank him for introducing me to Gridap.

I would like to thank Filipe Cumaru for taking time out to help me implement Trilinos locally. I thank Shagun Agarwal for helping me with setting up test cases of the floating structures problem.

We acknowledge the use of computational resources of the DelftBlue supercomputer, provided by Delft High Performance Computing Centre (https://www.tudelft.nl/dhpc).

I thank Prof. Vaibhav Joshi for being my on-campus supervisor for this project. His help with respect to the organizational tasks has been invaluable. I thank the secretary, Minaksie Ramsoekh, for the onboarding process and for helping with organizational-related issues. I would like to thank BITS Pilani Goa campus for allowing students to pursue undergraduate research at any research lab around the world.

Finally, I would like to thank my family and friends for constantly supporting me.

# *Digital Assistance Declaration*

I would like to acknowledge the use of Grammarly and Writefull for formatting and grammar correction as writing aids while writing this thesis report.

# Certificate

This is to certify that the thesis entitled, "*Development of preconditioning strategies for simulating very large floating structures*" and submitted by Shreyas P. ID No. 2020B4A42096G in partial fulfillment of the requirements of BITS F422T Thesis embodies the work done by him under my supervision.

*Supervisor*
Dr. Alexander Heinlein
Assistant Professor,
Delft University of Technology
Date: May, 2025

# Abbreviations

| | |
|---|---|
| **FSI** | **F**luid **S**tructure **I**nteraction |
| **DD** | **D**omain **D**ecomposition |
| **1D** | **1 D**imensional |
| **2D** | **2 D**imensional |
| **VLFS** | **V**ery **L**arge **F**loating **S**tructures |
| **DN** | **D**irichlet-**N**eumann |
| **RN** | **R**obin-**N**eumann |
| **DR** | **D**irichlet-**R**obin |
| **RR** | **R**obin-**R**obin |
| **BGS** | **B**lock **G**auss **S**eidel |
| **PDE** | **P**artial **D**ifferential **E**quation |
| **ODE** | **O**rdinary **D**ifferential **E**quation |
| **CSC** | **C**ompressed **S**parse **C**olumn |
| **CRS** | **C**ompressed **R**ow **S**parse |
| **GMRES** | **G**eneralized **M**inimal **R**esidual **M**ethod |
| **FGMRES** | **F**lexible **G**eneralized **M**inimal **R**esidual **M**ethod |
| **LU** | **L**ower **U**pper |
| **DOF** | **D**egrees **O**f **F**reedom |

# Abstract

Floating solar arrays are increasingly being pursued around the world as a form of renewable energy source. This has increased interest in understanding the behavior of floating structures in offshore environments. Fluid-Structure Interaction models are developed and used to simulate this phenomena. In a monolithic linear system, the fluid and solid domains are solved together at the same time. These systems are very large and ill-conditioned, making the use of the monolithic set of equations difficult to solve using iterative solvers.

To solve this issue, this project involves developing preconditioning strategies for these monolithic systems. Using non-overlapping domain decomposition methods, we introduce block preconditioners for this linear monolithic system. These block preconditioners are developed to be immune to the instabilities caused by the added mass effect. This makes the block preconditioners developed robust to the change in the geometry of the solid domain and the density ratio between the solid and fluid, achieving excellent convergence properties. To make the system suitable to simulate on parallel computing clusters, overlapping domain decomposition methods are applied while solving the inner blocks in the block preconditioner.

This preconditioning system allows us to simulate very large systems on existing parallel computing hardware. This system is implemented using Gridap.jl, an open-source finite element library, and Trilinos, an open-source library for solving large linear systems. Gridap is written using the Julia programming language with Julia's JIT compiler to provide an easy and understandable interface, yet being very efficient and fast. Trilinos is written in the C++ programming language, providing high-performance functionalities to solve large linear systems on large distributed parallel systems. Through this project, a seamless interface between Gridap and Trilinos is built.

This project shows that the block preconditioner based on the Robin-Neumann coupling between the fluid and solid domain is most suited for these problems. The developed implementation using Gridap and Trilinos is shown to be efficient using the strong and weak scaling tests while simulating large problems in parallel compute clusters. The project is concluded by simulating a floating structure problem using the developed system.

# Contents

# Chapter 1

# Introduction

Offshore floating solar platforms have been gaining popularity as a renewable energy source worldwide. An accurate analysis of the hydro-elastic response of these systems has become necessary to design them. Monolithic fluid-structure interaction solvers [15] have been developed to analyze these hydro-elastic responses computationally of these very large floating structures (VLFS). These monolithic linear systems, where we solve the fluid and solid equations together, are very large and ill-conditioned, making them difficult to solve using iterative solvers.

In this project, we develop preconditioners for these monolithic linear systems, making it easier for us to solve them using iterative solvers on existing parallel computing platforms. These preconditioners are developed based on the concepts of overlapping and non-overlapping domain decomposition methods. Through this project, we also understand the implementation of these systems using Gridap, an open-source finite element library, and Trilinos, an open-source library for solving large linear systems.

We start by giving a brief introduction to domain decomposition methods in chapter 2. We start by introducing non-overlapping domain decomposition methods for elliptic PDEs and their implementation. We refer to [33], [9], [13] and [24]. Using this understanding, we look at block preconditioners for the Stokes fluid flow problem. We refer to [37] and [10]. We then introduce overlapping domain decomposition methods, primarily limiting ourselves to one-level additive Schwarz and two-level additive Schwarz with a Lagrangian-based coarse space. We refer to [43], [11] and [35].

In chapter 3, we develop block preconditioners for thin-structured FSI problems. We look at the various coupling algorithms (Dirichlet-Neumann and Robin-Robin) developed and their convergence specifically for thin-structured FSI problems. Using these coupling algorithms, we develop block preconditioners that can be applied to the monolithic system. We refer to [17], [5], [6], [31], [12], [7], [30], [23] and [19].

We then look at defining a floating structure problem to which we can apply the block preconditioner system developed. We refer to [1] and [15]. The developed block preconditioners are applied to this problem.

We implement the developed system using Gridap [46] and Trilinos [41]. The package Gridap Solvers [32] is used to implement the block preconditioners and the outer matrix solver. Distributed finite element matrix assembly and processing are implemented via GridapDistributed [4]. Distributed matrix data structures in GridapDistributed are implemented with the PartitionedArrays.jl package. The overlapping domain-decomposition-based preconditioner is implemented using FROSch [27] present in the ShyLU [36] package of Trilinos, Belos, the iterative solver package in Trilinos, and Amesos2, the direct solvers package in Trilinos [8]. While developing an interface between Trilinos and Gridap, CxxWrap.jl [28] is used to run the C++ executable from the Julia environment.

In chapter 4, we discuss the implementation methodology and results. We show the convergence analysis of all the methods discussed previously. We start with showing results for the non-overlapping and overlapping DD methods for an elliptic PDE problem. We then look at the results of the block preconditioner implementation on a toy FSI problem implemented using the preconditioned Richardson solver. Following this, we look at the results of the parallel block preconditioner implementation on the toy FSI problem implemented using the FGMRES matrix solver. Through this example, we show that we achieve optimal strong and weak scaling while simulating a large FSI toy problem. We then implement the proposed system for the floating structure problem for an infinite membrane with a damping zone and an infinite membrane with periodic boundary conditions.

# Chapter 2

# Introduction to Preconditioning and Domain-Decomposition Methods

## 2.1 The linear system and the preconditioners

Consider a linear system $Ax = b$. If the system is very big and complex, then it becomes difficult to solve it. Therefore, we have introduced a preconditioner $P^{-1}$ that makes it easier to solve the system. There are two types of preconditioners. A left preconditioner would be applied to the linear system like this $P^{-1}Ax = P^{-1}b$. A right preconditioner would be applied to a linear system like this $AP^{-1}y = b$ with $y = P^{-1}x$.
A preconditioner should have the following traits -

- Cheap to compute (easy to apply $P^{-1}$).
- Make it easier to solve the problem. We will look into detail on what this means in further sections.

The two extremes of a preconditioner can be the following.

- A preconditioner can be the Identity matrix $I$. In this case, it is very easy (trivial) to compute the preconditioner $P^{-1}$, but it does make it easier to solve the problem as the preconditioned linear system would be identical to the original linear system.
- A preconditioner can be $A^{-1}$. In this case, the preconditioned linear system would directly yield us the answer. However, in order to compute $A^{-1}$, it would amount to solving the original linear system. Hence we have not made the problem easier to solve.

While solving large linear systems, we use iterative solvers where we iterate towards a solution from an initial guess. To understand the convergence properties of iterative solvers, we introduce the condition number of a matrix. The condition number ($\kappa$) of a matrix A as introduced in [45] is given by the following, where $\lambda$ is an eigenvalue of matrix A.

$$\kappa(A) = \left| \frac{\lambda_{max}}{\lambda_{min}} \right|$$

If $\kappa(A) \approx 1$, then we can say that the linear system is well conditioned and it's easy to find the solution through iterative solvers. But when $\kappa(A)$ is large, it gets difficult to solve using iterative methods due to slow or no convergence to a solution.

### 2.1.1 Iterative Richardson method

The Richardson method [38] is an iterative method to solve a linear system $Ax = b$. Preconditioners and relaxation parameters are used to accelerate convergence. In these methods, the solution is iteratively updated. The update step is derived as follows.

$$x = x + b - Ax \tag{2.1}$$

Rewriting (2.1) to update the solution iteratively gives the following. We use the superscript $k$ to indicate the iteration steps.

$$x^{k+1} = x^k + b - Ax$$

The iteration step when a preconditioner and relaxation is used is given by the following.

$$x^{k+1} = x^k + \alpha_k P^{-1}(Ax^k - b)$$

The iteration matrix for this modified system is $I - \alpha_k P^{-1} A$. Convergence occurs only if the spectral radius of the iteration matrix is less than one. This can be controlled by using an optimal preconditioner and relaxation parameter.

### 2.1.2 GMRES method

This is an iterative Krylov method to find the solution of a linear system $Ax = b$ introduced in [40]. In this method, the solution is constructed from the Krylov subspace.

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\}$$

For an SPD matrix, the convergence is directly related to the condition number of the matrix. We can also look at the eigenvalue distribution of the matrix to understand the convergence of the linear system. The convergence can be influenced by using a preconditioner.

We can use a preconditioner along with the GMRES method. The GMRES method assumes that the preconditioner remains constant; hence, it calculates the preconditioner only once and uses this preconditioned Krylov space for all the other iterations. But when we use iterative solvers in the inner solves of the preconditioner, then the preconditioner actually changes every iteration as a tolerance for the error is set and is less than the machine precision. Therefore, a flexible GMRES method [39] is introduced that calculates the preconditioner every iteration.

## 2.2 Block preconditioners

Let us consider that we have the following linear system where the coefficient matrix is partitioned into sub-matrices or blocks.

$$AX = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

We can derive preconditioners for the above block linear system called block preconditioners.

### 2.2.1 Preconditioners based on block LU factorization

In this section, we look at how we construct block preconditioners by matrix factorization. The LU factorization of matrix A is as follows:

$$A = LU = \begin{bmatrix} A_{11} & 0 \\ A_{12}A_{11}^{-1} & A_{22} - A_{12}A_{11}^{-1}A_{21} \end{bmatrix} \begin{bmatrix} I & A_{21} \\ 0 & I \end{bmatrix}$$

$A_{22} - A_{12}A_{11}^{-1}A_{21}$ is denoted as $s$ or the Schur complement. L can be used as a preconditioner if we can find a good and cheap estimate for $A^{-1}$ and $s^{-1}$.

### 2.2.2 Block Gauss-Seidel preconditioner

We can also construct a preconditioner based on the block Gauss-Seidel (GS) method. This preconditioner is based on the iterative Gauss-Seidel method. These are called block GS preconditioners. For matrix A, we have:-

$$P_{GS} = \begin{bmatrix} A_{11} & 0 \\ A_{12} & A_{22} \end{bmatrix}$$

$$P_{GS}^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{11}^{-1}A_{12}A_{22}^{-1} & A_{22}^{-1} \end{bmatrix}$$

Through this preconditioner, we modify the problem where we can find the inverses of each diagonal block $(A_{11}, A_{22})$ independently and use them to construct the preconditioner.

## 2.3 Non-overlapping domain-decomposition methods

Non-overlapping domain-decomposition methods can be constructed by using the idea of the block Gauss-Seidel preconditioner previously discussed. We will introduce a finite element problem through which these methods are discussed. These methods are adopted from [34], [9] and [13].

### 2.3.1 Introducing a coupled elliptic PDE problem

We consider the following coupled problem. There are two different Poisson problems introduced on the left domain $(\Omega_l)$, and the right domain $(\Omega_r)$ as the left problem and the right problem respectively.

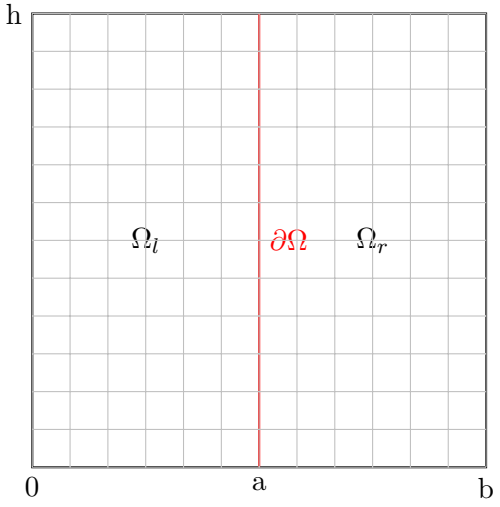



FIGURE 2.1: Computational domain for the coupled elliptic PDE problem

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $u_l$ | Variable on the left domain | $u_r$ | Variable on the right domain |
| $a_1$ | Diffusion coefficient for the left domain | $a_2$ | Diffusion coefficient for the right domain |
| $\Omega_l$ | Left domain | $\Omega_r$ | Right domain |
| $\partial_{lr}$ | Left-Right domain interface | $n$ | Normal vector to the interface |
| A | Discrete finite element matrix on the left domain. | B | Discrete finite element matrix on the right domain. |

TABLE 2.1: Definitions of terms related to the coupled elliptic PDE problem.

The equations governing this problem are as follows.

$$\begin{aligned} a_1 \Delta u_l &= f_1 \text{ on } \Omega_l \\ a_2 \Delta u_r &= f_2 \text{ on } \Omega_r \\ u_{l\Gamma} &= u_{r\Gamma} \text{ on } \partial_{lr} \\ a_1 \nabla u_{l\Gamma} \cdot n_l + a_2 \nabla u_{r\Gamma} \cdot n_r &= 0 \text{ on } \partial_{lr} \end{aligned}$$

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $u^h$ | Trial function | $v^h$ | Test function |
| $\Omega_l$ | Whole domain | $\Gamma_d$ | Dirichlet boundary |
| $U^h$ | Trial space | $V^h$ | Test space |

TABLE 2.2: Finite element terms related to the coupled elliptic PDE problem

The trial space for the following problem is defined as follows.

$$U^h = \{u^h \in H^1(\Omega) : u^h = u_0 \text{ on } \Gamma_D\}$$

The test space for the following problem is defined as follows.

$$V^h = \{v^h \in H^1(\Omega) : v^h = 0 \text{ on } \Gamma_D\}$$

The discrete form of this coupled problem after finite element discretization yields the following.

$$\begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ 0 & I & 0 & -I \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} & B_{\Gamma I} & B_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_{lI}^{n+1} \\ u_{l\Gamma}^{n+1} \\ u_{rI}^{n+1} \\ u_{r\Gamma}^{n+1} \end{bmatrix} = \begin{bmatrix} r_l \\ 0 \\ r_r \\ r_\Gamma \end{bmatrix} \tag{2.2}$$

### 2.3.2 Dirichlet-Neumann method

#### 2.3.2.1 Algorithm

Under the Dirichlet-Neumann Algorithm, the left Poisson problem is solved using the Dirichlet data from the right Poisson problem of the previous iteration, and the right Poisson problem is solved using the Neumann data from the left Poisson problem that was just solved. Here $\omega$ is the relaxation parameter introduced.

$$\begin{aligned} a_1 \Delta u_l^{k+1} &= f_1 \quad \text{on } \Omega_l \\ u_{l\Gamma}^{k+1} &= u_{r\Gamma}^{k} \quad \text{on } \partial_{lr} \\ a_2 \Delta u_r^{k+1} &= f_2 \quad \text{on } \Omega_r \\ a_1 \nabla u_{l\Gamma}^{k+1} \cdot n_l + a_2 \nabla \hat{u}_{r\Gamma}^{k+1} \cdot n_r &= 0 \quad \text{on } \partial_{lr} \end{aligned}$$

$\hat{u}_{r\Gamma}$ is relaxed each iteration using the relaxation parameter $\omega$.

$$u_{r\Gamma}^{k+1} = \omega \hat{u}_{r\Gamma}^{k+1} + (1-\omega) u_{r\Gamma}^{k}$$

The algorithm expressed in discrete form yields the following.

$$\begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} & B_{\Gamma I} & B_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_{lI}^{k+1} \\ u_{l\Gamma}^{k+1} \\ u_{rI}^{k+1} \\ \hat{u}_{r\Gamma}^{k+1} \end{bmatrix} = \begin{bmatrix} r_l \\ u_{r\Gamma}^{k} \\ r_r \\ r_\Gamma \end{bmatrix} \tag{2.3}$$

$$u_{r\Gamma}^{k+1} = \omega \hat{u}_{r\Gamma}^{k+1} + (1-\omega) u_{r\Gamma}^{k}$$

Applying the preconditioner $P_{DN}$ shown below on the monolithic system 2.2, we get the iterative discrete system 2.3.

$$P_{DN} = \begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} & B_{\Gamma I} & B_{\Gamma\Gamma} \end{bmatrix}$$

#### 2.3.2.2 Convergence criteria

We can solve the problem analytically using the Dirichlet-Neumann algorithm to find the relative error. This result is taken from [33].

$$e = 1 - \omega\left(\frac{a_1 tanh(\frac{n\pi(b-a)}{H})}{a_2 tanh(\frac{n\pi a}{H})} + 1\right) \text{ where } n \in \mathbb{N} \tag{2.4}$$

For the algorithm to converge to a solution, we need the relative error ($e$) to be less than one for every $n$. We can manipulate the relaxation parameter $\omega$ to achieve this.
From 2.4 we can observe that when $a = b$, it is possible to have $e = 0$ for all $n$. This means we can reach the solution in one iteration if we choose an optimal $\omega$.

$$\omega_{opt} = \frac{1}{\frac{a_1}{a_2} + 1} \text{ when a = b}$$

For all other cases, we just minimize $e$ for $n = 1$. Then we get $\omega_{opt}$ as follows.

$$\omega_{opt} = \frac{1}{\left(\frac{a_1 tanh(\frac{\pi(b-a)}{b})}{a_2 tanh(\frac{\pi a}{b})} + 1\right)}$$

### 2.3.3 Robin-Robin method

The original problem had Dirichlet and Neumann transmission conditions on the interface. We introduce new transmission conditions that are the linear combination of the Dirichlet and Neumann transmission conditions. $\alpha_f$ and $\alpha_s$ are the parameters introduced here.

$$\begin{aligned} a_1 \Delta u_l &= f_1 \quad \text{on } \Omega_l \\ \alpha_f u_{l\Gamma} + a_1 \nabla u_{l\Gamma} \cdot n_l &= \alpha_f u_{r\Gamma} - a_2 \nabla u_{r\Gamma} \cdot n_r \quad \text{on } \partial_{lr} \\ a_2 \Delta u_r &= f_2 \quad \text{on } \Omega_r \\ \alpha_s u_{l\Gamma} + a_1 \nabla u_{l\Gamma} \cdot n_l &= \alpha_s u_{r\Gamma} - a_2 \nabla u_{r\Gamma} \cdot n_r \quad \text{on } \partial_{lr} \end{aligned}$$

The discrete form of this problem is as follows.

$$\begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ A_{\Gamma I} & A_{\Gamma\Gamma} + \alpha_f I & B_{\Gamma I} & B_{\Gamma\Gamma} - \alpha_f I \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} - \alpha_s I & B_{\Gamma I} & B_{\Gamma\Gamma} + \alpha_s I \end{bmatrix} \begin{bmatrix} u_{lI} \\ u_{l\Gamma} \\ u_{rI} \\ u_{r\Gamma} \end{bmatrix} = \begin{bmatrix} r_l \\ r_\Gamma \\ r_r \\ r_\Gamma \end{bmatrix}$$

A mapping $Q$ is introduced that maps the old problem to the new problem.

$$Q = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & \alpha_f I & 0 & I \\ 0 & 0 & I & 0 \\ 0 & -\alpha_s I & 0 & I \end{bmatrix}$$

#### 2.3.3.1 Algorithm

We solve the left problem using the Robin transmission data of the right problem of the previous iteration, and then we solve the right problem using the Robin transmission data of the left problem we just solved. The algorithm is as follows:

$$\begin{aligned} a_1 \Delta u_l^{k+1} &= f_1 \quad \text{on } \Omega_l \\ \alpha_f u_{l\Gamma}^{k+1} + a_1 \nabla u_{l\Gamma}^{k+1} \cdot n_l &= \alpha_f u_{r\Gamma}^{k} - a_2 \nabla u_{r\Gamma}^{k} \cdot n_r \quad \text{on } \partial_{lr} \\ a_2 \Delta u_r^{k+1} &= f_2 \quad \text{on } \Omega_r \\ \alpha_s u_{l\Gamma}^{k+1} + a_1 \nabla u_{l\Gamma}^{k+1} \cdot n_l &= \alpha_s u_{r\Gamma}^{k+1} - a_2 \nabla \hat{u}_{r\Gamma}^{k+1} \cdot n_r \quad \text{on } \partial_{lr} \end{aligned}$$

The relaxation step for $\hat{u}_{r\Gamma}$ is introduced as follows:

$$u_{r\Gamma}^{k+1} = \omega \hat{u}_{r\Gamma}^{k+1} + (1-\omega) u_{r\Gamma}^{k}$$

The algorithm expressed in discrete form yields the following.

$$\begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ A_{\Gamma I} & A_{\Gamma\Gamma} + \alpha_f I & 0 & 0 \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} - \alpha_s I & B_{\Gamma I} & B_{\Gamma\Gamma} + \alpha_s I \end{bmatrix} \begin{bmatrix} u_{lI}^{k+1} \\ u_{l\Gamma}^{k+1} \\ u_{rI}^{k+1} \\ u_{r\Gamma}^{k+1} \end{bmatrix} = \begin{bmatrix} r_l \\ r_\Gamma - B_{\Gamma I} u_{rl}^{k} - (B_{\Gamma\Gamma - \alpha_f I}) u_{r\Gamma}^{k} \\ r_r \\ r_\Gamma \end{bmatrix}$$

We can construct a Robin-Robin preconditioner $P_{RR}$ that produces the above iterative linear system for the original linear system 2.2.

$$P_{RR} = Q^{-1} \begin{bmatrix} A_{II} & A_{I\Gamma} & 0 & 0 \\ A_{\Gamma I} & \alpha_f I + A_{\Gamma\Gamma} & 0 & 0 \\ 0 & 0 & B_{II} & B_{I\Gamma} \\ A_{\Gamma I} & -\alpha_s I + A_{\Gamma\Gamma} & B_{\Gamma I} & B_{\Gamma\Gamma} + \alpha_s I \end{bmatrix}$$

We can construct a class of Robin-Robin preconditioners by varying these Robin parameters. The following table is a summary of all the algorithms we can construct from the generalized Robin-Robin algorithm.

| Name | $\alpha_f$ | $\alpha_s$ |
|---|---|---|
| Dirichlet-Neumann | Inf | 0 |
| Dirichlet-Robin | Inf | Positive bounded value |
| Robin-Neumann | Positive bounded value | 0 |
| Robin-Robin | Positive bounded value | Positive bounded value |

TABLE 2.3: Summary of all the algorithms relating to the Robin-Robin method

### 2.3.4 Block preconditioner for the Stokes problem

#### 2.3.4.1 Stokes problem definition

The governing equations associated with the non-transient incompressible Stokes fluid flow problem are shown below. We consider the computational domain $\Omega$ shown in figure 2.2 while defining the problem.

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $u$ | Fluid Velocity | $p$ | Pressure |
| $f$ | Forcing function | $g$ | Dirichlet Boundary Data |

TABLE 2.4: Definition for the terms in the stokes problem

$$\Delta u - \nabla p = f \quad \text{on } \Omega$$

$$\nabla \cdot u = 0 \quad \text{on } \Omega$$

$$u = g \quad \text{on } \partial\Omega$$

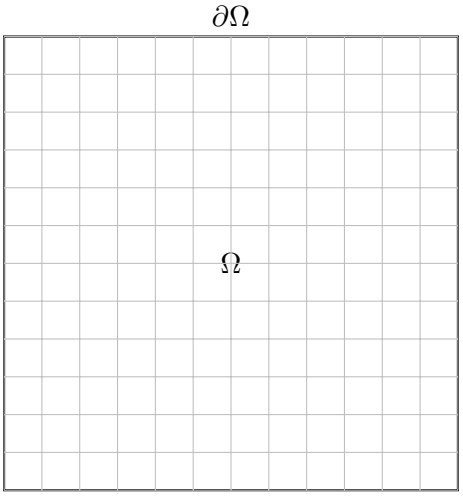



FIGURE 2.2: Computational domain for the stokes problem

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $u^h$ | Velocity trial function | $v^h$ | Velocity test function |
| $p^h$ | Pressure trial function | $q^h$ | Pressure test function |
| $\Omega$ | Whole domain | $\Gamma_d$ | Dirichlet boundary |
| $U^h$ | Velocity trial space | $V^h$ | Velocity test space |
| $P^h$ | Pressure trial space | $Q^h$ | Pressure test space |

TABLE 2.5: Finite element terms related to the stokes problem

The trial and test spaces are defined as the following.

$$U^h = \{u^h \in H^1(\Omega) : u^h = u_0 \text{ on } \Gamma_D\}$$

$$V^h = \{v^h \in H^1(\Omega) : v^h = 0 \text{ on } \Gamma_D\}$$

$$P^h = \{p^h \in L^2(\Omega)\}$$

$$Q^h = \{q^h \in L^2(\Omega) : \int_\Omega q^h \, d\Omega = 0\}$$

$P2$ $P1$ elements are used here as they satisfy the inf-sup condition [3] to provide a stable finite element solution. We can write the discrete form for this problem as follows.

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{2.5}$$

#### 2.3.4.2 Algorithm

The goal is to develop a block preconditioner for the given Stokes problem. It is also interesting to observe that this is in the form of a saddle point problem as seen in 2.5. Using the block preconditioner that is based on a block LU decomposition as mentioned in [16], we get the following.

$$P = \begin{bmatrix} A & B^T \\ 0 & S \end{bmatrix}$$

$$S = -BA^{-1}B^T$$

The challenge here is to cheaply estimate $A^{-1}$ and $S^{-1}$. For the selected problem, as we take the viscosity to be 1, we can easily approximate the Schur complement as the pressure mass matrix $M_p$ as mentioned in [37]. A is an easily invertible matrix in a tri-diagonal and positive definite form. This changes the preconditioner to the following.

$$P = \begin{bmatrix} \hat{A} & B^T \\ 0 & M_p \end{bmatrix}$$

## 2.4 Overlapping Domain-Decomposition methods

### 2.4.1 Introducing an elliptic PDE problem

We consider the following one-dimensional coupled problem.

$$\frac{\partial^2 u}{\partial x^2} = 1 \text{ where } x \in [0,1] \tag{2.6}$$

$$u(0) = u(1) = 0$$

### 2.4.2 Alternating Schwarz methods

To understand these methods, let us first create overlapping subdomains for the problem.

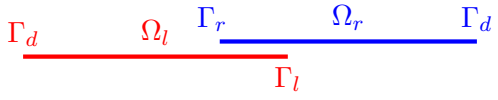



FIGURE 2.3: Overlapping Subdomains on domain $\Omega$

We refer to [22] to understand the history of the Schwarz methods. The alternating Schwarz iterative algorithm to solve 2.6 would yield the following.

$$\begin{aligned}\frac{\partial^2 u_l^{k+1}}{\partial x^2} &= 1 \text{ where } x \in \Omega_l \\ u_l^{k+1} &= u_r^k \text{ on } \Gamma_l \\ \frac{\partial^2 u_r^{k+1}}{\partial x^2} &= 1 \text{ where } x \in \Omega_r \\ u_r^{k+1} &= u_l^{k+1} \text{ on } \Gamma_r\end{aligned}$$

This algorithm was first introduced in the 1800s to solve problems in complex geometries. From the algorithm, we can see that we solve the left Poisson problem first and use the right Poisson problem solution of the previous iteration step as the boundary condition for the left problem. Then we solve the right Poisson problem using the solution of the left Poisson problem as the boundary condition. These steps are repeated until the solution converges along the whole domain.
It is important to note that this algorithm cannot be implemented in parallel. Hence, we introduce the one-level additive Schwarz method.

### 2.4.3 One-level additive Schwarz method

We try to parallelize the alternating Schwarz method. We introduce the Jacobi-Schwarz algorithm, and it is as follows.

$$\begin{aligned}\frac{\partial^2 u_l^{k+1}}{\partial x^2} &= 1 \text{ where } x \in \Omega_l \\ u_l^{k+1} &= u_r^k \text{ on } \Gamma_l \\ \frac{\partial^2 u_r^{k+1}}{\partial x^2} &= 1 \text{ where } x \in \Omega_r \\ u_r^{k+1} &= u_l^k \text{ on } \Gamma_r\end{aligned}$$

Here it can be noted that on each subdomain, we can solve independently during each iteration. We now look to define the discretizations and derive the discrete form for this problem.

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $u$ | Trial function | $v$ | Test function |
| $u_i^h$ | Trial function for the $i^{th}$ subdomain | $v_i^h$ | Test function for the $i^{th}$ subdomain |
| $\Omega$ | Whole domain | $\Omega_i$ | Domain of the $i^{th}$ subdomain |
| $U^h$ | Trial space | $V^h$ | Test space |
| $U_i^h$ | Trial space for the $i^{th}$ subdomain | $V_i^h$ | Test space for the $i^{th}$ subdomain |
| $A$ | Stiffness matrix for the problem | $A_i$ | Stiffness matrix for the $i^{th}$ subdomain |

TABLE 2.6: Finite element terms related to the elliptic PDE problem

The discrete problem for 2.6 would be the following.

$$Au = f$$

Introducing a restriction operator that does the following mapping for each subdomain. Here $V$ represents the global finite element space and $V_i$ represents the local finite element space of a subdomain.

$$R_i^T : V_i \to V$$

Then the finite element space for the whole problem can be stitched up as follows using the local finite element spaces and its associated restriction operators.

$$V = \sum_i R_i^T V i$$

Relating the local bilinear form to the bilinear form of the problem, we get the following.

$$a_i(u, v) = a(R_i^T u_i, R_i^T v_i) \tag{2.7}$$

Using 2.7, we can get the following relationship.

$$A_i = R_i A R_i^T$$

#### 2.4.3.1 Algorithm

In the additive Schwarz algorithm [43], the problem is solved in all subdomains at the same time using solutions of other subdomains as boundary conditions from the previous iteration. It is also to be noted that during each iteration, in the overlap region, the solution is a combination of contributions from neighboring subdomains, which are computed independently and then aggregated additively.

The contribution from each local subdomain towards the global solution would be as follows.

$$u|_{i^{th}contribution} = (R_i^T A_i^{-1} R_i) f$$

In this algorithm, all the contributions are added to form the global solution.

$$u = \sum_i (R_i^T A_i^{-1} R_i) f \tag{2.8}$$

If we introduce some sort of restriction in this addition, then we get the restrictive additive Schwarz method. Here the restriction is based on having weighted contributions from all subdomains based on the partition of unity. $D_i$ is a diagonal matrix filled with positive numbers that act as these weights.

$$u = \sum_i (R_i^T D_i A_i^{-1} R_i) f$$

Now we try to develop a preconditioner that converts the original problem to 2.8. We define this as the one-level additive Schwarz preconditioner.

$$P_{1AS} = \sum_i (R_i^T A_i^{-1} R_i)$$

#### 2.4.3.2 Convergence criteria

The condition number estimate of the one-level additive Schwarz preconditioned system applied on exact solvers is shown below. This is taken from [43].

$$\kappa(P_{1AS}^{-1} A) \leq C(1 + \frac{1}{H\delta})$$

Here $H$ represents the size of the subdomains and $\delta$ represents the size of the overlap. As we increase the number of subdomains for a particular problem, the subdomain size $H$ decreases accordingly. So, we can say that as the number of subdomains increases, the condition number of the system increases without any upper bound. As we know that having a large condition number is equal to difficulty in solving the problem, this negates the use of the preconditioner. Therefore, the one-level additive Schwarz preconditioner is not scalable.

### 2.4.4 Two level additive Schwarz method

This method solves the key problem associated with the one-level additive Schwarz method relating to scalability. We will consider the same discrete problem considered for the one-level additive Schwarz method.

#### 2.4.4.1 Algorithm

In this method, a coarse problem is introduced as mentioned in [43], in addition to the problems associated with each local subdomain. For simplicity, we assume that this coarse problem is introduced on a triangulation $\tau_c$ that is nested within $\tau$. The restriction operator $R_0$ maps the finite element space $V$ of the whole domain to the constructed coarse domain $V_0$. The restriction operator $R_0$ is the interpolation of the coarse basis function onto the fine mesh $\tau$.
The global solution we get through this algorithm is the following.

$$u = (R_0^T A_0^{-1} R_0 + \sum_i (R_i^T A_i^{-1} R_i)) f \tag{2.9}$$

Now we try to develop a preconditioner that converts the original problem to 2.9. We define this as the two-level additive Schwarz preconditioner.

$$P_{2AS} = R_0^T A_0^{-1} R_0 + \sum_i (R_i^T A_i^{-1} R_i)$$

#### 2.4.4.2 Convergence criteria

The condition number estimate we get for using the two-level additive Schwarz method is as follows, obtained from [43].

$$\kappa(P_{2AS}^{-1} A) \leq C(1 + \frac{H}{\delta})$$

We can observe that as the number of subdomains increases, there is an upper bound for the condition number. We can therefore say that this method will remain well-conditioned irrespective of the number of subdomains we choose for a problem. Therefore, this method is scalable.

# Chapter 3

# Block Preconditioners for Fluid-Structure Interaction Problems

## 3.1 Introduction of the toy problem

In this section, we introduce a toy problem adapted from [21] through which we develop the block preconditioners required. In this thesis, we will be limiting our study to the thin-solid assumption. This means that the solid domain and the fluid-solid interface domain remain the same. The thin solid assumption is made for the following reasons.

- The solid has a very small thickness to length ratio.
- The response in primarily governed by the bending dominated behavior where transverse displacements are very minimal.
- This reduces the number of unknown degrees of freedom required in the simulation.

The computational domain that is used for this particular toy problem in this thesis is shown below.

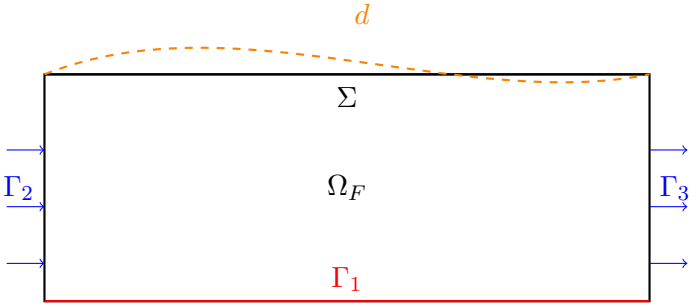



FIGURE 3.1: Computational domain for the toy FSI problem

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $\Omega_F$ | Fluid domain | $\Sigma$ | Solid/interface domain |
| $u$ | Fluid velocity | $p$ | Fluid pressure |
| $n$ | Normal vector towards the interface | $d$ | Solid displacement |
| $\rho_f$ | Fluid density | $\rho_s$ | Solid density |
| $\epsilon$ | Solid thickness | $E$ | Young's modulus |
| $R$ | Radius | Poisson's ratio | $\nu$ |
| L | $\frac{E\epsilon}{R^2(1-\nu^2)}$ | | |
| $u_i$ | Fluid velocity trial function defined on $\Omega_f \setminus \Sigma$ | $v_i$ | Fluid velocity test function defined on $\Omega_f \setminus \Sigma$ |
| $u_\Gamma$ | Fluid velocity trial function defined on $\Sigma$ | $v_\Gamma$ | Fluid velocity test function defined on $\Sigma$ |
| $p$ | Fluid pressure trial function | $q$ | Fluid pressure test function |
| $d$ | Solid displacement trial function | $s$ | Solid displacement test function |
| $U_i$ | Fluid velocity trial space defined on $\Omega_f \setminus \Sigma$ | $V_i$ | Fluid velocity test space defined on $\Omega_f \setminus \Sigma$ |
| $U_\Gamma$ | Fluid velocity trial space defined on $\Sigma$ | $V_\Gamma$ | Fluid velocity test space defined on $\Sigma$ |
| $P$ | Fluid pressure trial space | $Q$ | Fluid pressure test space |
| $D$ | Solid displacement trial space | $S$ | Solid displacement test space |
| $C_{II}$ | Matrix constructed from space $U_i$ and $V_i$ | $C_{I\Gamma}$ | Matrix constructed from space $U_i$ and $V_\Gamma$ |
| $C_{\Gamma I}$ | Matrix constructed from space $U_\Gamma$ and $V_i$ | $C_{\Gamma\Gamma}$ | Matrix constructed from space $U_\Gamma$ and $V_\Gamma$ |
| $N_{\Gamma\Gamma}$ | Matrix constructed from space $D$ and $S$ | $D_I$ | Matrix constructed from space $U_i$ and $Q$ |
| $D_\Gamma$ | Matrix constructed from space $U_\Gamma$ and $Q$ | $r_{f\Gamma}$ | Residual constructed from space $V_\Gamma$ |
| $r_p$ | Residual constructed from space $Q$ | $r_f$ | Residual constructed from space $V_i$ |
| $r_s$ | Residual constructed from space $S$ | Jx = r | Discrete system for the toy problem |

TABLE 3.1: Definition of terms relating to the toy FSI problem

A generalized string model is used to model the solid domain taken from [12]. The following are the equations that are used as the governing equations for this problem. We will refer to this as

the toy problem going forward.

- Fluid domain

$$\begin{aligned} \rho_f \frac{\partial u}{\partial t} + \nabla p &= 0 \text{ on } \Omega_f \\ \nabla \cdot u &= 0 \text{ on } \Omega_f \\ u \cdot n_\Gamma &= 0 \text{ on } \Gamma_1 \\ p &= 0 \text{ on } \Gamma_2 \cup \Gamma_3 \end{aligned}$$

- Solid domain and the interface

$$\rho_s \epsilon \ddot{d} + Ld = p \cdot n_{fs} \text{ on } \Sigma \tag{3.1}$$

$$u = \dot{d} \text{ on } \Sigma \tag{3.2}$$

The fluid velocity and solid displacement finite element spaces are in $H^1$. The fluid pressure finite element space is in $L^2$. Second-order Lagrangian finite elements are used as interpolation functions for fluid velocity and solid displacement. First-order Lagrangian finite elements are used for fluid pressure. The weak form of the system is given as follows.

$$\int (\rho_f \frac{\partial u}{\partial t} \cdot v)\Omega_f - \int (p(\nabla \cdot v))\Omega_f = -\int (p_{\Gamma 2} v \cdot n) d\Gamma_2 - \int (p_{\Gamma 1} v \cdot n) d\Gamma_1$$

$$\int ((\nabla \cdot u) q) d\Omega_f = 0$$

$$\int (\rho_s \epsilon \frac{\partial^2 d}{\partial t^2} s + Lds) d\Sigma = \int ps d\Sigma$$

For the purpose of understanding the numerical behavior related to the coupling algorithms, we modify these equations by applying the divergence operator. We will refer to this problem as the modified toy problem going forward.

$$\begin{aligned} -\Delta p &= 0 \text{ on } \Omega_f \\ \nabla p \cdot n_{fs} &= 0 \text{ on } \Gamma_1 \\ \rho_s \epsilon \ddot{d} + Ld &= p \text{ on } \Sigma \\ \nabla p \cdot n_{fs} &= -\rho_f \ddot{d} \text{ on } \Sigma \end{aligned} \tag{3.3}$$

To understand how we get the Neumann boundary condition on the Fluid-Solid interface (3.3), we look at the weak form of the fluid equation in the modified toy problem.

$$\int (q\Delta p)d\Omega_f = \int ((\nabla p \cdot n_{fs})tr_\Sigma(q))d\Sigma - \int (\nabla q \cdot \nabla p)d\Omega_f$$

$(\nabla p \cdot n_{fs})$ is defined as $-\rho_f \frac{\partial u}{\partial t}$ according to the toy problem. As Dirichlet boundary conditions on $u$ have been imposed according to the toy problem, $-\rho_f \frac{\partial u}{\partial t}$ would be the Neumann boundary condition we would impose on the interface in the modified toy problem.
Now we will introduce a new term called the added mass operator or the Stekloff Poincaré operator adopted from [12]. For a finite element problem, this operator gives the trace of the solution along for the Neumann boundary (Neumann to Dirichlet map). The formal definition of this operator $M_A$ is given below.
Let the operator $M_A : H^{1/2}(\Sigma) -> H^{1/2}(\Sigma)$ be defined for each $g \epsilon H^{-1/2}(\Sigma)$, where we set $M_A(g) = q|_\Sigma$ where $q \epsilon H^1(\Omega)$ for the following problem.

$$\begin{aligned} -\Delta q &= 0 \text{ on } \Omega \\ \frac{\partial q}{\partial n} &= g \text{ on } \Sigma \\ \frac{\partial q}{\partial n} &= 0 \text{ on } \Gamma_1 \\ q &= 0 \text{ on } \Gamma_2 \end{aligned}$$

## 3.2 Coupling algorithms

We derive all the expressions using a simple backward difference scheme for the time discretization.

### 3.2.1 Explicit Dirichlet-Neumann method

#### 3.2.1.1 Algorithm

In this coupling algorithm, the fluid and solid equations are solved staggered in time. The fluid problem is solved using the interface Dirichlet data of the solid problem from the previous time step, and similarly, the solid problem is solved using the interface Neumann data from the fluid problem. Implementing this algorithm for the toy problem gives the following.

- Fluid Step

$$\rho_f \frac{u^{n+1} - u^n}{\delta t} + \nabla p^{n+1} = 0 \text{ on } \Omega_f$$

$$\nabla \cdot u^{n+1} = 0 \text{ on } \Omega_f$$

$$u^{n+1} = \frac{d^n - d^{n-1}}{\delta t} \text{ on } \Sigma$$

- Solid Step

$$\rho_s \epsilon [\frac{d^{n+1} - 2d^n + d^{n-1}}{\delta t^2}] + Ld^n = p \cdot n_{fs}|_\Sigma \text{ on } \Sigma$$

#### 3.2.1.2 Convergence criteria

To help us derive and understand the convergence criteria, we implement the algorithm for the modified toy problem, yielding the following.

$$-\Delta p^{n+1} = 0 \text{ on } \Omega_f$$

$$\frac{\partial p^{n+1}}{\partial n_{fs}} = -\rho_f [\frac{d^n - 2d^{n-1} + d^{n-2}}{\delta t^2}] \text{ on } \Sigma$$

We can express the above set of equations in one interface equation using the added mass operator $M_A$.

$$\rho_s \epsilon [\frac{d^{n+1} - 2d^n + d^{n-1}}{\delta t^2}] + \rho_f M_A [\frac{d^n - 2d^{n-1} + d^{n-2}}{\delta t^2}] + Ld^n = 0 \text{ on } \Sigma$$

For the added mass operator $M_A$, we consider the following eigenvalue problem with $z_i$ being the eigenvectors and $\mu_i$ being the associated eigenvalues.

$$M_A z_i = \mu_i z_i$$

If $M_A$ has m independent eigenvalues and eigenvectors, then we can represent d as the following.

$$d = \sum_{i=1}^{m} d_i z_i$$

We want to derive $M_A d$ in terms of $d_i$, $z_i$ and $\mu_i$.

$$M_A d = M_A(\sum_i d_i z_i) = \sum_i d_i (M_A z_i)$$

Using 3.2.2.2 to simplify 3.2.1.2, we get the following.

$$M_A d = \sum_i d_i \mu_i z_i$$

Simplifying 3.2.1.2 using 3.2.1.2 we get the following difference equation.

$$\frac{\rho_f \epsilon}{\delta t^2} d_i^{n+1} + [\frac{\rho_f \mu_i - 2\rho_s \epsilon}{\delta t^2} + L] d_i^n + [\frac{\rho_f \mu_i - 2\rho_s \epsilon}{\delta t^2}] d_i^{n-1} + \frac{\rho_f \mu_i}{\delta t^2} d_i^{n-2} = 0 \tag{3.4}$$

We want to understand the stability of the difference equation. We propose an exponential solution $\lambda^n$ to be the trial solution for 3.4. After substituting $\lambda^n$ and simplifying and homogenizing the equation, we get the characteristic equation.

$$P(\lambda) = \frac{\rho_f \epsilon}{\delta t^2} \lambda^3 + [\frac{\rho_f \mu_i - 2\rho_s \epsilon}{\delta t^2} + L] \lambda^2 + [\frac{\rho_f \mu_i - 2\rho_s \epsilon}{\delta t^2}] \lambda + \frac{\rho_f \mu_i}{\delta t^2} = 0 \tag{3.5}$$

The made-up solution to the characteristic equation would be something like $\lambda, \lambda^2, \lambda^3 ... \lambda^n$. For the difference equation to be stable, we would need to have an upper bound for $\lambda^n$. This is only possible when $|\lambda| < 1$. We derive the stability condition for this algorithm from here.
Plotting the cubic polynomial 3.5, we get the following.

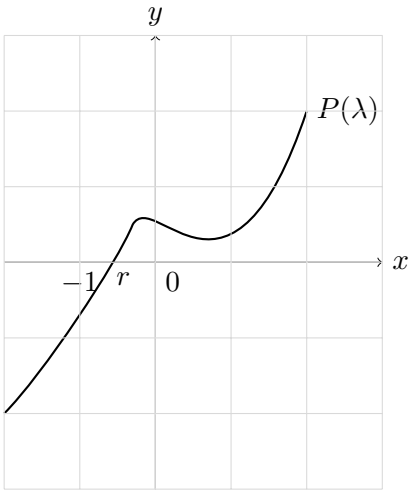



FIGURE 3.2: Characteristic equation plot

For the algorithm to be stable, we want the root $|r|$ of the characteristic equation to be less than one. We also have:

$$P(-1) = L + \frac{4}{\delta t^2}(\rho_f \mu_i - \rho_s \epsilon) \tag{3.6}$$

From the plot, we can observe that we want the root to be in between -1 and 0. So for that to happen, we want $P(-1)$ to be negative. From 3.6, we can prove the condition for instability by showing $P(-1) > 0$. We get the following condition.

$$\frac{\rho_f \mu_{max}}{\rho_s \epsilon} > 1 \text{ Condition for instability} \tag{3.7}$$

From 3.7, we can make the following inferences.

- Instability depends on the density ratio.

- Instability does not depend on the time step ($\delta t$).
- Instability occurs when the structure is thin and slender (large $\mu_{max}$). Refer [12].

### 3.2.2 Implicit Dirichlet-Neumann method

This is an iterative algorithm, unlike the explicit Dirichlet-Neumann method.

#### 3.2.2.1 Algorithm

In this algorithm, for each iteration, a fluid problem is solved using the Dirichlet data from the solid problem of the previous iteration, and then the solid problem is solved using the Neumann fluid data from the same iteration. A relaxation parameter $\omega$ is introduced. This iteration is done for each time step until the solution converges. Due to the implicit nature of this algorithm, we always get an accurate solution if the solution converges. The downside of using this method is the added computational cost involved as the solver has to run for multiple iterations.
Solving the toy problem for the $(n+1)^{st}$ timestep at the $(k+1)^{st}$ iteration, we get the following.

- Fluid Step

$$\rho_f \frac{u_{k+1} - u^n}{\delta t} + \nabla p^{n+1} = 0 \text{ on } \Omega_f$$

$$\nabla \cdot u_{k+1} = 0 \text{ on } \Omega_f$$

$$u_{k+1} = \frac{d_k - d^n}{\delta t} \text{ on } \Sigma$$

- Solid Step

$$\rho_s \epsilon [\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = p \text{ on } \Sigma$$

- Relaxation Step

$$d_{k+1} = \omega \hat{d}_{k+1} + (1-\omega) d_k$$

#### 3.2.2.2 Convergence criteria

We implement this algorithm on the modified toy problem to help us develop convergence criteria for this method.

$$-\Delta p_{k+1} = 0 \text{ on } \Omega_f$$

$$\frac{\partial p_{k+1}}{\partial n_{fs}} = -\rho_f [\frac{d_k - 2d^n + d^{n-1}}{\delta t^2}] \text{ on } \Sigma$$

Reducing the above set of equations to the interface equation using the added mass operator $M_A$ gives us the following.

$$\rho_s\epsilon[\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + \rho_f M_A[\frac{d_k - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = 0 \text{ on } \Sigma \tag{3.8}$$

For the added mass operator $M_A$, we consider the following eigenvalue problem with $z_i$ being the eigenvectors and $\mu_i$ being the associated eigenvalues.

$$M_A z_i = \mu_i z_i$$

If $M_A$ has m independent eigenvalues and eigenvectors, then we can represent d as the following.

$$d = \sum_{i=1}^{m} d_i z_i$$

Similar to what we did in the implicit Dirichlet-Neumann method, we express 3.8 in terms of $d_i$, $z_i$ and $\mu_i$.

$$\frac{1}{\omega}[\frac{\rho_s\epsilon}{\delta t^2} + L]d_{i(k+1)} + [\frac{\rho_f\mu_i}{\delta t^2} - [\frac{1-\omega}{\omega}][\frac{\rho_s\epsilon}{\delta t^2} + L]]d_{i(k)} + [\frac{\rho_s\epsilon + \rho_f\mu_i}{\delta t^2}][d_i^{n-1} - 2d_i^n] = 0 \tag{3.9}$$

Writing the characteristic homogenized equation of 3.9 gives the following.

$$\frac{1}{\omega}[\frac{\rho_s\epsilon}{\delta t^2} + L]\lambda + [\frac{\rho_f\mu_i}{\delta t^2} - [\frac{1-\omega}{\omega}][\frac{\rho_s\epsilon}{\delta t^2} + L]] = 0$$

For the above characteristic equation to be stable, we need $|\lambda| < 1$ or the absolute value of the root of the characteristic equation to be less than one. Using this information, the stability criteria derived for this method are as follows.

$$|\frac{(1-\omega)(\rho_s\epsilon + L\delta t^2) - \omega\rho_f\mu_i}{\rho_s\epsilon + L\delta t^2}| < 1 \tag{3.10}$$

As $\omega$ is a parameter we can control, we can define an optimum range of $\omega$ suitable for this method.

$$0 < \omega < \frac{2(\rho_s\epsilon + L\delta t^2)}{\rho_s\epsilon + \rho_f\mu_{max} + L\delta t^2} \tag{3.11}$$

From 3.10 and 3.11, we can deduce the following inferences about the usage of the implicit Dirichlet-Neumann method.

- For a thin/slender body, $\mu_{max}$ is large, therefore $\omega$ range is small. Refer [12].
- If $\frac{\rho_f}{\rho_s}$ is large, then $\omega$ would be small. This means that we would iterate slower towards the solution leading to more number of iterations required.
- Does not converge all the time if relaxation is not used.

- The convergence rate is dependent on the timestep size ($\delta t$).

#### 3.2.2.3 Dirichlet-Neumann block preconditioner

We will now represent the Dirichlet-Neumann method applied to the toy problem in discrete form.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ 0 & 0 & I & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i^{k+1} \\ p^{k+1} \\ u_\Gamma^{k+1} \\ d^{k+1} \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ \partial_t(d)^k \\ r_{f\Gamma} + r_s \end{bmatrix} \tag{3.12}$$

The discrete version of the toy problem is given as follows.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ 0 & 0 & I & -\partial_t(I) \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i \\ p \\ u_\Gamma \\ d \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ 0 \\ r_{f\Gamma} + r_s \end{bmatrix} \tag{3.13}$$

Now we try to develop a preconditioner that converts the original discrete toy problem 3.13 to 3.12.

$$P_{DN} = \begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ 0 & 0 & I & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix} \tag{3.14}$$

### 3.2.3 Implicit Robin-Robin method

We have observed that the convergence for the Dirichlet-Neumann method for scenarios where there is a high added mass effect is poor. Therefore, to mitigate this, we introduce the family of Robin-Robin methods where we change the transmission conditions to Robin conditions. This has been explained in detail for an elliptic PDE problem in 2. Introducing the toy problem modified with the Robin transmission conditions. Let us refer to this problem as the Robin-Robin toy problem. $\alpha_f$ and $\alpha_s$ are introduced as the Robin-Robin parameters here.

- Fluid domain

$$\begin{aligned} \rho_f \frac{\partial u}{\partial t} + \nabla p &= 0 \text{ on } \Omega_f \\ \nabla \cdot u &= 0 \text{ on } \Omega_f \\ u \cdot n_\Gamma &= 0 \text{ on } \Gamma_1 \\ p &= 0 \text{ on } \Gamma_2 \cup \Gamma_3 \end{aligned}$$

- Solid domain and the interface

$$\rho_s \epsilon \ddot{d} + Ld + \alpha_f u = p + \alpha_f \dot{d} \text{ on } \Sigma$$

$$\rho_s \epsilon \ddot{d} + Ld + \alpha_s \dot{d} = p + \alpha_s u \text{ on } \Sigma$$

The discrete form of the problem would be the following.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & \alpha_f I + C_{\Gamma\Gamma} & -\alpha_f * \partial_t(I) + N_{\Gamma\Gamma} \\ C_{\Gamma I} & G_\Gamma & -\alpha_s I + C_{\Gamma\Gamma} & \alpha_s * \partial_t(I) + N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i \\ p \\ u_\Gamma \\ d \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ r_\Gamma + r_s \\ r_{f\Gamma} + r_s \end{bmatrix} \tag{3.15}$$

A mapping Q is introduced mapping the toy problem to the Robin-Robin toy problem.

$$Q = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \alpha_f I & I \\ 0 & 0 & -\alpha_s I & I \end{bmatrix}$$

#### 3.2.3.1 Algorithm

In this algorithm, for each iteration, a fluid problem is solved using the Robin data from the solid problem of the previous iteration, and then the solid problem is solved using the Robin fluid data from the same iteration. A relaxation parameter $\omega$ is introduced. This iteration is done for each time step until the solution converges. Solving the Robin-Robin toy problem for the $(n+1)^{st}$ timestep at the $(k+1)^{st}$ iteration, we get the following.

- Fluid Step

$$\rho_f \frac{u_{k+1} - u^n}{\delta t} + \nabla p^{n+1} = 0 \text{ on } \Omega_f$$

$$\nabla \cdot u_{k+1} = 0 \text{ on } \Omega_f$$

$$\alpha_f u_{k+1} + \rho_s \epsilon [\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = \alpha_f \frac{d_k - d^n}{\delta t} + p^k \cdot n_{fs} \text{ on } \Sigma$$

- Solid Step

$$\alpha_s \frac{d_k - d^n}{\delta t} + \rho_s \epsilon [\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = \alpha_s u_{k+1} + p^k \cdot n_{fs} \text{ on } \Sigma$$

- Relaxation Step

$$d_{k+1} = \omega \hat{d}_{k+1} + (1-\omega) d_k$$

#### 3.2.3.2 Robin-Robin block preconditioner

We will now represent the Robin-Robin method applied to the toy problem in discrete form.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & \alpha_f I + C_{\Gamma\Gamma} & 0 \\ C_{\Gamma I} & G_\Gamma & -\alpha_s I + C_{\Gamma\Gamma} & \alpha_s * \partial_t(I) + N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i^{k+1} \\ p^{k+1} \\ u_\Gamma^{k+1} \\ d^{k+1} \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ r_{f\Gamma} + r_s + (\alpha_f * \partial_t(I) + N_{\Gamma\Gamma}) d^k \\ r_{f\Gamma} + r_s \end{bmatrix} \tag{3.16}$$

Now we try to develop a preconditioner that converts the original toy problem 3.13 to 3.16.

$$P_{RR} = Q^{-1} * \begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & \alpha_f I + C_{\Gamma\Gamma} & 0 \\ C_{\Gamma I} & G_\Gamma & -\alpha_s I + C_{\Gamma\Gamma} & \alpha_s * \partial_t(I) + N_{\Gamma\Gamma} \end{bmatrix} \tag{3.17}$$

### 3.2.4 Implicit Robin-Neumann method

This method comes under the umbrella of the Robin-Robin methods discussed in the previous section. Here we set $\alpha_s = 0$. We therefore will not discuss the problem formulation in detail for this method. The Robin-Neumann problem is given as the following.

- Fluid domain

$$\begin{aligned} \rho_f \frac{\partial u}{\partial t} + \nabla p &= 0 \text{ on } \Omega_f \\ \nabla \cdot u &= 0 \text{ on } \Omega_f \\ u \cdot n_\Gamma &= 0 \text{ on } \Gamma_1 \\ p &= 0 \text{ on } \Gamma_2 \cup \Gamma_3 \end{aligned}$$

- Solid domain and the interface

$$\rho_s \epsilon \ddot{d} + Ld + \alpha_f u = p + \alpha_f \dot{d} \text{ on } \Sigma$$

$$\rho_s \epsilon \ddot{d} + Ld = p \text{ on } \Sigma$$

#### 3.2.4.1 Algorithm

Applying the iterative algorithm on the Robin-Neumann toy problem gives the following.

- Fluid Step

$$\rho_f \frac{u_{k+1} - u^n}{\delta t} + \nabla p^{n+1} = 0 \text{ on } \Omega_f$$

$$\nabla \cdot u_{k+1} = 0 \text{ on } \Omega_f$$

$$\alpha_f u_{k+1} + \rho_s \epsilon [\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = \alpha_f \frac{d_k - d^n}{\delta t} + p_k \text{ on } \Sigma$$

- Solid Step

$$\rho_s \epsilon [\frac{\hat{d}_{k+1} - 2d^n + d^{n-1}}{\delta t^2}] + L\hat{d}_{k+1} = p_k \text{ on } \Sigma$$

- Relaxation Step

$$d_{k+1} = \omega \hat{d}_{k+1} + (1 - \omega) d_k$$

#### 3.2.4.2 Convergence criteria

To get the optimum convergence, we can vary the Robin parameter $\alpha_f$. Referring to [5], it is shown that if we do backward differencing (BDF) for discretizing the temporal equation, we get the following. Using the relation 3.2 in 3.1 gives us the following.

$$\rho_s \epsilon \frac{\partial u}{\partial t} + Ld = p \tag{3.18}$$

Temporally discretizing 3.18 and upon further simplification, gives the following.

$$(\frac{\rho_s \epsilon}{\Delta t} + L\Delta t) u^{n+1} + p^{n+1} = (\frac{\rho_s \epsilon}{\Delta t^2} - L\Delta t) d^n - \frac{\rho_s \epsilon}{\Delta t^2} d^{n-1} \tag{3.19}$$

From 3.19, we can estimate the optimal Robin parameter as follows.

$$\alpha_f = \frac{\rho_s \epsilon}{\Delta t} + L\Delta t$$

We can use the $\alpha_f$ value derived and apply the Robin-Neumann algorithm on the modified toy problem. Reducing this problem to an interface equation using the added mass operator $M_A$ gives us the following result.

$$(\frac{L\Delta t^2 + \rho_s \epsilon}{\rho_f} M_A^{-1} + I) p^{k+1} = Ld^n \tag{3.20}$$

Looking at 3.20, we can observe that this is an explicit update equation for the pressure term. We can say that the Robin-Neumann algorithm does not develop any instabilities due to the algorithm, therefore making this algorithm immune to the added mass effects. It is also important to note that this is only true for the case of simulating thin solid FSI problems. More results on the optimum Robin parameter for other time discretizations and convergence results are discussed in 4.
For other more complex time discretizations, we introduce a new parameter $\gamma$ to estimate the optimal Robin parameter. $\gamma$ is selected experimentally through parametric analysis.

$$\alpha_f = \gamma(\frac{\rho_s \epsilon}{\Delta t} + L\Delta t) \tag{3.21}$$

#### 3.2.4.3 Robin-Neumann block preconditioner

We will now represent the Robin-Neumann method applied to the toy problem in discrete form.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & \alpha_f I + C_{\Gamma\Gamma} & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i^{k+1} \\ p^{k+1} \\ u_\Gamma^{k+1} \\ d^{k+1} \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ r_{f\Gamma} + r_s + (\alpha_f * \partial_t(I) + N_{\Gamma\Gamma})d^k \\ r_{f\Gamma} + r_s \end{bmatrix} \tag{3.22}$$

Now we try to develop a preconditioner that converts the original toy problem 3.13 to 3.22.

$$P_{RN} = Q^{-1} * \begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & \alpha_f I + C_{\Gamma\Gamma} & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix}$$

## 3.3 Parallel preconditioners for FSI problems

In this section, we look at the parallel implementation of the block preconditioners. One of the first challenges that we observe is that we cannot use exact solvers for solving the inner blocks in the block preconditioner, as this is computationally very inefficient when solving large problems in parallel. We use iterative solvers based on Krylov solvers to solve these inner blocks. The result of using iterative solvers in the preconditioner means that the preconditioner changes every iteration, as the solvers do not produce exact solutions with machine precision. Therefore, we cannot use GMRES as the outer solver for solving the preconditioned monolithic system, as the algorithm is only designed for constant preconditioning. We therefore use the FGMRES algorithm as the outer solver, as this is designed for variable preconditioning. Refer [39]. The

FGMRES algorithm is designed to be used with right preconditioners; we therefore have to reformulate the block preconditioners and the FSI problem.

### 3.3.1 Parallel block FSI preconditioners

In this section, we will reformulate the discrete FSI problem and the associated block preconditioners mentioned in 3.2. The discrete problem mentioned in 3.13 would be the following.

$$\begin{bmatrix} N_{\Gamma\Gamma} & C_{\Gamma I} & C_{\Gamma\Gamma} & G_\Gamma \\ 0 & C_{II} & C_{I\Gamma} & G_I \\ -\partial_t(I) & 0 & I & 0 \\ 0 & D_I & D_\Gamma & 0 \end{bmatrix} \begin{bmatrix} d \\ u_i \\ u_\Gamma \\ p \end{bmatrix} = \begin{bmatrix} r_{f\Gamma} + r_s \\ r_f \\ 0 \\ r_p \end{bmatrix}$$

The associated Robin-Robin problem as mentioned in 3.15 would be the following .

$$\begin{bmatrix} \alpha_s * \partial_t(I) + N_{\Gamma\Gamma} & C_{\Gamma I} & -\alpha_s I + C_{\Gamma\Gamma} & G_\Gamma \\ 0 & C_{II} & C_{I\Gamma} & G_I \\ -\alpha_f * \partial_t(I) + N_{\Gamma\Gamma} & C_{\Gamma I} & \alpha_f I + C_{\Gamma\Gamma} & G_\Gamma \\ 0 & D_I & D_\Gamma & 0 \end{bmatrix} \begin{bmatrix} d \\ u_i \\ u_\Gamma \\ p \end{bmatrix} = \begin{bmatrix} r_{f\Gamma} + r_s \\ r_f \\ r_{f\Gamma} + r_s \\ r_p \end{bmatrix}$$

The new mapping matrix $Q$ would be the following.

$$Q = \begin{bmatrix} I & 0 & -\alpha_s I & 0 \\ 0 & I & 0 & 0 \\ I & 0 & \alpha_f I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

The Dirichlet-Neumann Preconditioner 3.14 written as a right-preconditioner would be the following.

$$P_{DN_{parallel}} = \begin{bmatrix} N_{\Gamma\Gamma} & C_{\Gamma I} & C_{\Gamma\Gamma} & G_\Gamma \\ 0 & C_{II} & C_{I\Gamma} & G_I \\ 0 & 0 & I & 0 \\ 0 & D_I & D_\Gamma & 0 \end{bmatrix}$$

The Robin-Robin Preconditioner 3.17 written as a right preconditioner would be the following.

$$P_{RR_{parallel}} = \begin{bmatrix} \alpha_s * \partial_t(I) + N_{\Gamma\Gamma} & C_{\Gamma I} & -\alpha_s I + C_{\Gamma\Gamma} & G_\Gamma \\ 0 & C_{II} & C_{I\Gamma} & G_I \\ 0 & C_{\Gamma I} & \alpha_f I + C_{\Gamma\Gamma} & G_\Gamma \\ 0 & D_I & D_\Gamma & 0 \end{bmatrix} * Q^{-1}$$

The general structure of the FSI block preconditioner is as follows.

$$P = \begin{bmatrix} \text{Solid Block} & \text{Coupling Terms} \\ 0 & \text{Fluid Block} \end{bmatrix}$$

While applying the block preconditioner, we would need to compute the inverses of the Solid Block and Fluid Block. For these inner solves, we employ the preconditioned GMRES matrix solver with the one-level additive Schwarz preconditioner. We can apply the one-level additive Schwarz preconditioner directly on the monolithic fluid block. Refer [29]. However, it is noted that as we increase the number of subdomains, the one-level additive Schwarz preconditioner does not scale well. To solve this issue, a coarse problem is added while constructing the preconditioner, as shown in [29] and [26]. It was seen that, specific to our use-case and the problem size, the added computational cost of adding the coarse space outweighed the benefit of having fewer iterations. Therefore, for this thesis, we stick to using the one-level additive Schwarz preconditioner while solving for the fluid and solid block inside the block preconditioner.

## 3.4 The floating structure problem

In this section, we will introduce the floating structure problem. For this thesis, we are looking at simulating floating pre-tensioned elastic membranes [1].

### 3.4.1 Problem definition

We consider the following computational domain similar to the one we used for the toy problem. We model the fluid flow using the Euler equation and neglect the convective term (non-linear terms). While modeling the elastic membrane, we neglect the effect from the bending stiffness (fourth order spatial derivative term).

| Term | Meaning | Term | Meaning |
|---|---|---|---|
| $\Omega_F$ | Fluid domain | $\Sigma$ | Floating membrane domain |
| $u$ | Fluid velocity | $p$ | Fluid pressure |
| $n$ | Normal vector towards the interface | $d$ | Solid displacement |
| $\rho_f$ | Fluid density | $\rho_s$ | Solid density |
| $h_s$ | Solid thickness | $T$ | Membrane pre-tension |
| g | Acceleration due to gravity | H | Total depth from surface to bed |

TABLE 3.2: Definition of terms relating to the floating structure problem

The equations governing this phenomenon are as follows.

- Fluid domain

$$\rho_f \frac{\partial u}{\partial t} + \nabla p = \rho_f g \text{ on } \Omega_f$$

$$\nabla \cdot u = 0 \text{ on } \Omega_f$$

$$u \cdot n_\Gamma = 0 \text{ on } \Gamma_1$$

$$p = p_{bc} \text{ on } \Gamma_2 \cup \Gamma_3$$

- Solid domain and the interface

$$\rho_s h_s \ddot{d} - \nabla \cdot (T \nabla d) = p \text{ on } \Sigma$$

$$\dot{d} = u \text{ on } \Sigma$$

Similar finite element discretization is done for this problem as we did for the toy problem. Refer to table 3.1 for the terms relating to the finite element discretization. The weak form for the given set of equations is the following.

$$\int (\rho_f \frac{\partial u}{\partial t} \cdot v)\Omega_f - \int (p(\nabla \cdot v))\Omega_f = \int ((\rho_f g)v)d\Omega f - \int (p_{\Gamma 2} v \cdot n)d\Gamma_2 - \int (p_{\Gamma 3} v \cdot n)d\Gamma_3$$

$$\int ((\nabla \cdot u)q)d\Omega_f$$

$$\int (\rho_s h_s \frac{\partial^2 d}{\partial t^2} s + T\nabla d \cdot \nabla s)d\Sigma = \int p s d\Sigma$$

The discrete problem derived is similar to the discrete problem mentioned for the toy problem. Therefore, we use the same preconditioners developed for the toy problem in the previous section.

# Chapter 4

# Implementation, Results and Discussion

All the implementations other than the section on overlapping methods have been implemented using the Gridap package [46] written in Julia. For the parallel implementation, we use the package Gridap Distributed [4] and the library Trilinos [41] for implementing the inner solves.

## 4.1 Non-overlapping domain-decomposition methods

### 4.1.1 Implementation of the problem in Gridap

This code was implemented using Gridap [46]. Block preconditioners and solvers were implemented using the GridapSolvers package [32]. The problem introduced in 2.3.1 is implemented. Preconditioned Richardson iteration is implemented. A direct matrix solver based on LU Decomposition is used to solve the blocks of the preconditioner. The domain is discretized into 20 elements across the x-axis and 20 elements across the y-axis. The interface is moved along the x-axis and its position is measured from the origin (example : interface = 1 implies interface is located one unit from the origin along the x-axis). The parameters used for numerical experiments are shown below.

| Term | Value | Term | Value |
|---|---|---|---|
| b | 5 | h | 5 |
| $a_1$ | Mentioned for each case | $a_2$ | Mentioned for each case |
| $f_1$ | 1 | $f_2$ | 1 |
| Absolute tolerance | 1e-6 | | |

TABLE 4.1: Parameters used in the coupled elliptic PDE problem.

### 4.1.2 Results and inference

We look at how the relaxation parameter affects the convergence for these methods. We also study how the position of the interface affects convergence. For the Robin-Robin methods, we look at how varying the Robin parameters affects the convergence. Detailed results are mentioned in the appendix A.1.

From all the methods that we have studied, it is evident that each method has its own set of unique characteristics. We keep the Dirichlet-Neumann method as the benchmark to compare the other methods. We can observe that the Dirichlet-Robin methods tend to move the convergence plot to the right. This made the method better suited to the problems where $a_1 > a_2$ compared to the Dirichlet-Neumann method. For the Robin-Neumann method, the behavior largely remained similar to the Dirichlet-Neumann method for this particular use case. The convergence of the Robin-Neumann method is inferior to the Dirichlet-Neumann method for this particular problem. The Robin-Robin method produced the most consistent results for all cases. When the robin parameter $\alpha \geq 0.5$, the method converged to the solution in about 10 iterations with no relaxation for all cases. The table below summarizes all the results.

| **Diffusion Contrast** | $a_1 = a_2$ | $a_1 = a_2$ | $a_1 > a_2$ | $a_1 > a_2$ | $a_1 < a_2$ | $a_1 < a_2$ |
|---|---|---|---|---|---|---|
| **Interface Position** | **Center** | **Asymmetric** | **Center** | **Asymmetric** | **Center** | **Asymmetric** |
| **Dirichlet-Neumann Method** | Best | Best | Bad | Bad | Good | Good |
| **Dirichlet-Robin Method** | Good | Good | Good | Good | Good | Good |
| **Robin-Neumann Method** | Good | Good | Bad | Bad | Good | Good |
| **Robin-Robin Method** | Good | Good | Best | Best | Best | Best |

TABLE 4.2: Summary for Non-Overlapping methods applied on the coupled elliptic PDE

## 4.2 Overlapping domain-decomposition methods

In this section, we implement basic overlapping domain-decomposition techniques in Julia and MATLAB on a one-dimensional Poisson problem described in 2.4.1.

### 4.2.1 Implementation of the problem

We have implemented these methods in Julia and MATLAB. The length of the 1-D domain is 1 unit and is divided into 1000 elements. The forcing function ($f(x)$) is kept to be a constant function equal to one. A preconditioned conjugate gradient solver is used to solve with the solver tolerance set to 1e-6. We kept a constant overlap of two cells while forming the overlapping subdomains.

### 4.2.2 One-level additive Schwarz

We plot the number of iterations that were required against the number of subdomains used. From plot 4.1, we observe that as we increase the number of subdomains for a problem, the number of iterations tends to proportionally increase without any upper bound. If we were to use a large number of subdomains to solve the problem, then we would reach a point where the computational cost and time would be so high that it would become unfeasible to use this method. Therefore, we say that this method is not scalable.

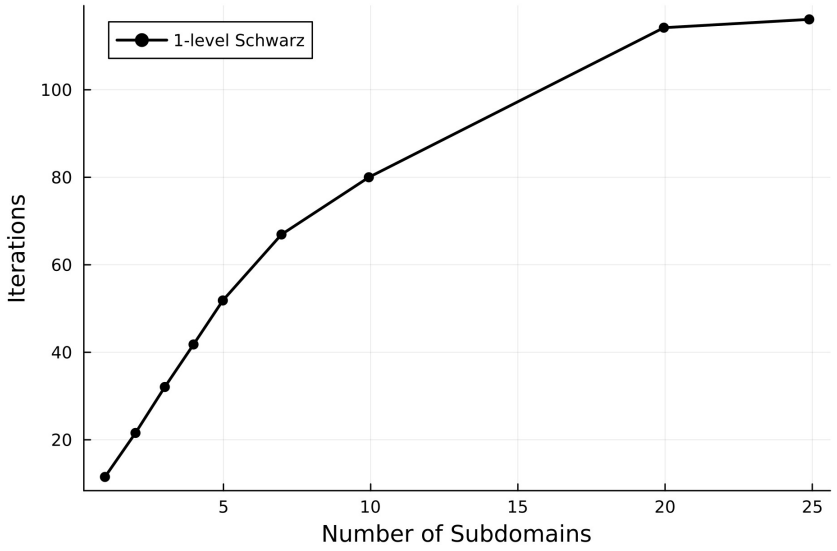



FIGURE 4.1: One-level Schwarz Results

### 4.2.3 Two-level additive Schwarz

To solve the problem of scalability that occurred in the one-level additive Schwarz method, we introduce this method. We plot the number of iterations that were required against the number of subdomains used.

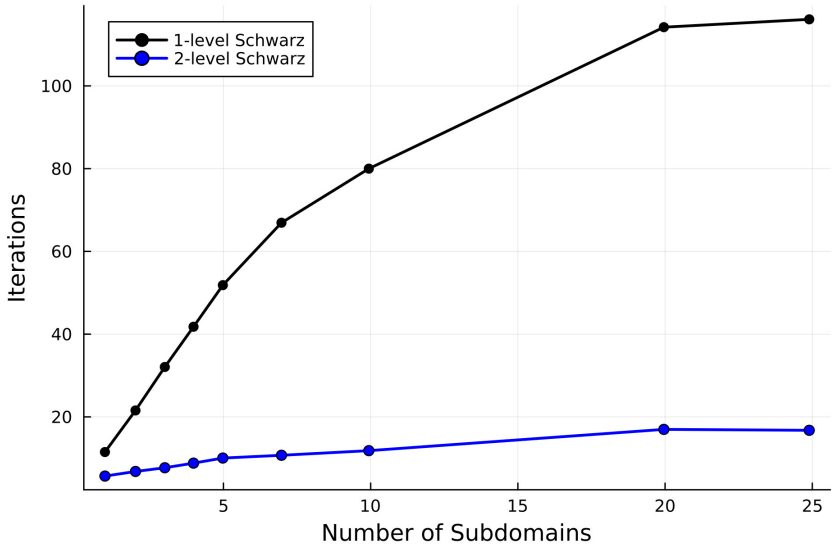



FIGURE 4.2: Two-level Schwarz Results

From the above plot, we can observe that the growth in the number of iterations required to solve the problem does not rapidly increase while increasing the number of subdomains for the two-level method compared to the one-level problem. This makes this method better suited to solve problems when a large number of subdomains are required.

## 4.3 Block-preconditioners for the toy FSI problem

The problem outlined in 3.1 is implemented in Gridap. The matrix for the uncoupled system is first generated in Gridap, and then the system is coupled through matrix modifications that are done in the numerical setup of the solver. The discrete equations of the uncoupled system are shown below.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & 0 \\ 0 & 0 & 0 & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_i \\ p \\ u_\Gamma \\ d \end{bmatrix} = \begin{bmatrix} rf_I \\ rf_p \\ rf_\Gamma \\ rs \end{bmatrix}$$

A new function is introduced that gives us an ordered pair of interface degrees of freedom for the fluid velocity and solid displacement. A unique 'dface' value is generated in Gridap for each geometric identity (example - vertex, edge, face) in the mesh. Using this generated unique geometric id, the required degrees of freedom are extracted and arranged. Refer to appendix B.1 for the function implementation. A solver wrapper is created that uses this data and performs the required matrix modifications before solving the matrix. This process is reported in detail in appendix B.2.

The below table contains the values specific to the toy problem that are implemented in this thesis.

| Term | Value | Term | Value |
|---|---|---|---|
| Length of the domain | 6 $cm$ | Radius of the domain | 1 $cm$ |
| E | 0.75e-6 $dynes/cm^2$ | $\epsilon$ | 0.1 $cm$ |
| $\rho_f$ | 1 $g/cm^3$ | $\rho_s$ | variable $g/cm^3$ |
| Numer of elements along x axis | 40 | Time step | 1e-3 *seconds* |
| Numer of elements along y axis | 5 | | |

TABLE 4.3: Toy FSI problem implementation

To initiate a pressure wave, an overpressure of 2e+4 $dynes/cm^2$ is applied for the first 0.005 seconds on the inlet boundary ($\Gamma_1$). The domain is divided into 40 elements along the $x$-axis and 5 elements along the $y$-axis.

### 4.3.1 Results

To understand the problem we are simulating, we show a pressure contour plot along with the solid displacement at the time of 0.3 seconds.

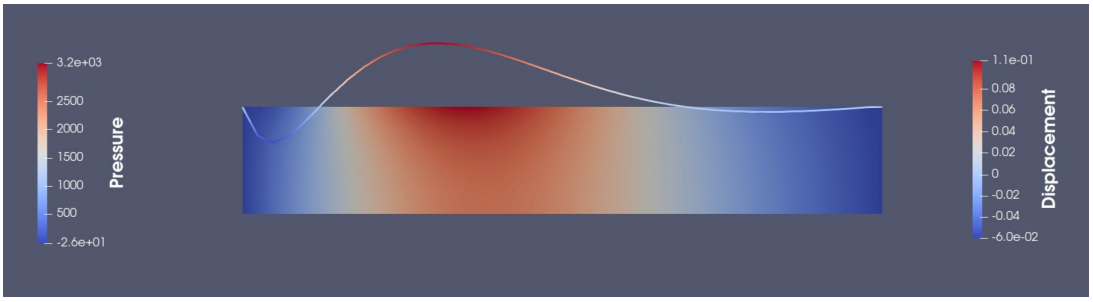



FIGURE 4.3: Pressure Contour plot along with solid displacement at time 0.3 seconds

#### 4.3.1.1 Dirichlet-Neumann preconditioner

We vary the solid density to help us understand how it impacts convergence. By varying the solid density, we get to understand how the added mass effect affects convergence, as the solid density is inversely proportional to the added mass effect. The below plot shows us how the relaxation parameter affects the iteration count.

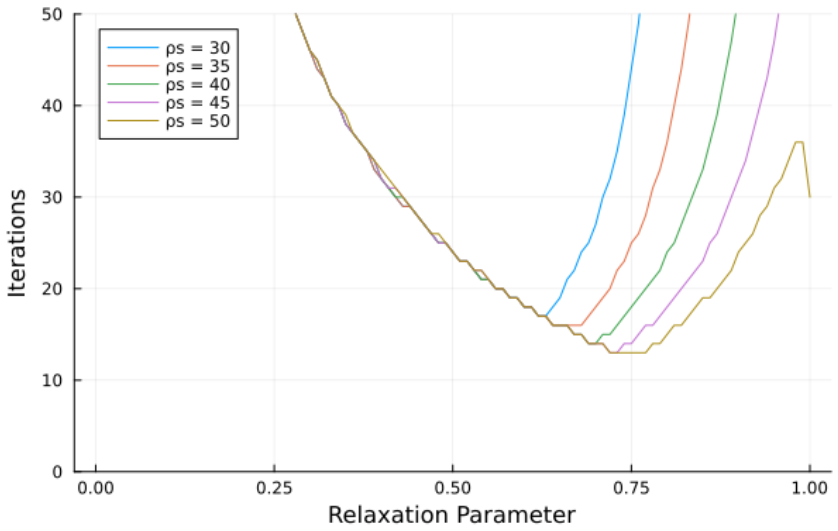



FIGURE 4.4: Numerical experiment-1 — Dirichlet-Neumann preconditioner

From the above plot, we observe that the range of relaxation parameters over which the solution converges reduces as the solid density is reduced. As the solid density increases, the number of iterations required to converge for the system with no relaxation reduces. This result corroborates with the theoretical convergence results we derived in the previous chapter. The glaring limitation of this method is that it fails to converge when the added mass effect is high (solid density is low).

#### 4.3.1.2 Robin-Robin preconditioner

We try to understand how this method performs when compared to the Dirichlet-Neumann method. The below plot shows us how the relaxation parameter affects the iteration count.

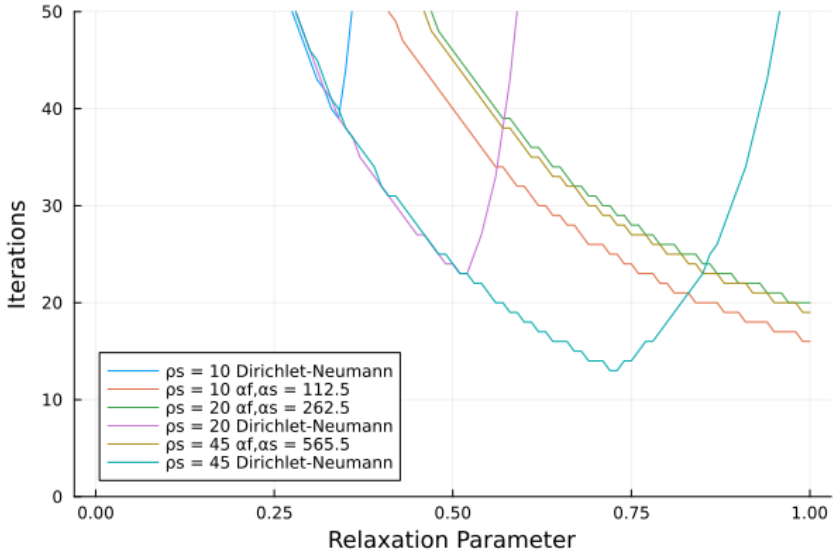



FIGURE 4.5: Numerical experiment-1 — Robin-Robin preconditioner

From the above plot, we can observe that the Robin-Robin method appears to be more immune to the added mass effects when compared to the Dirichlet-Neumann method. Although the convergence of the Dirichlet-Neumann method when the added mass effect is low appears to be better than the Robin-Robin method. It is also evident that no relaxation is required for the Robin-Robin method for optimal convergence.

#### 4.3.1.3 Robin-Neumann preconditioner

We try to understand how this method performs when compared to the Dirichlet-Neumann method. The below plot shows us how the relaxation parameter affects the iteration count.

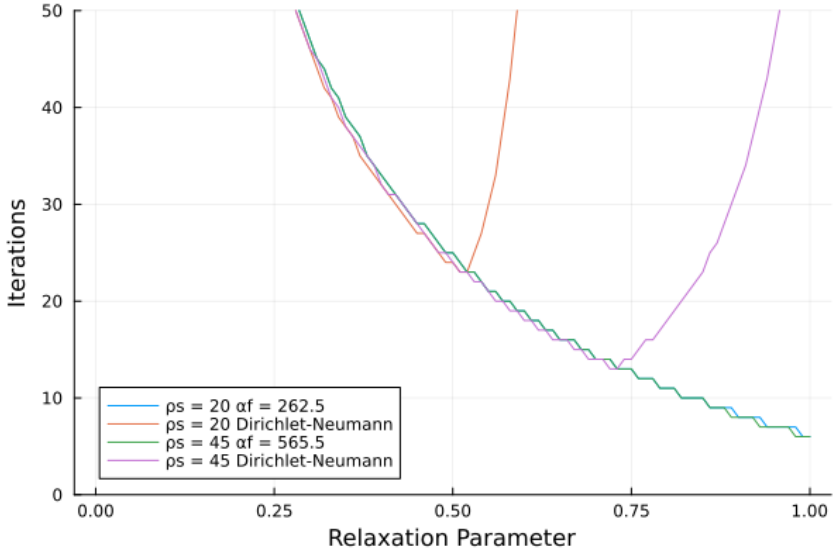



FIGURE 4.6: Numerical experiment-1 — Robin-Robin preconditioner

From the above plot, it is evident that the Robin-Neumann method is immune to the added mass effects, and the convergence properties of the Robin-Neumann method appear to be consistently better than the Dirichlet-Neumann method.

### 4.3.2 Inference

We summarize the results into a table for the purpose of easy understanding.

| Method | Low added mass effect | High added mass effect |
|---|---|---|
| Dirichlet-Neumann method | Good | Poor |
| Robin-Robin method | Poor | Good |
| Robin-Neumann method | Best | Best |

TABLE 4.4: Toy FSI serial results summary

## 4.4 Parallel block preconditioners for the toy problem

### 4.4.1 FSI parallel implementation in Gridap

The finite element parallel assembly is done using the GridapDistributed package. The finite element matrices are assembled locally on each separate MPI process. The matrix modifications to couple the fluid and solid systems take place locally on each process on the local matrix without any communication with other processes. The Julia package PartitionedArrays.jl provides the distributed linear algebra framework within GridapDistributed [4].

While storing data for the distributed matrices, each local process stores the local matrix along with the associated row map and column map. Similarly, for distributed vectors, each local process stores the local vector and its associated row map. A row map is a vector that maps the local indices of the local matrix stored in the process to the global matrix.

A ghost node is a position on a local matrix in a process that is actually owned by another process. These duplicate data entries are there to minimize communication among processes during the matrix assembly and certain matrix operations. These ghost nodes generally store the data of the adjacent cells just outside the local subdomain.

Generally, there is a domain map and a range map. In the domain map, all the possible ghost cells are included in each mpi process. In the range map, there are no ghost cells included or all the local nodes are owned by the local process.

As we implemented the matrix modifications for the serial problem, a similar process is followed in the parallel problem. In the parallel problem, the matrix modifications are performed on the local matrices independently using the local DOF IDs extracted. The additional complexity involved here is to build efficient maps that map a local index defined on a domain map to a row map or column map. The DOF IDs extracted give the local DOF IDs that are mapped to the domain map of the FE Space. Then we have to convert these DOF IDs to local row DOF IDs that are mapped to row maps and local column DOF IDs that are mapped to column maps.

### 4.4.2 Gridap-Trilinos interface

During this thesis project, a solver interface between Gridap and Trilinos is developed. Trilinos, being a library written in C++ and Gridap being written in Julia, we used the library CxxWrap.jl to run the C++ binaries from Julia.

There are three parts to the interface. The first is assembling the required linear system in Trilinos using the appropriate data structures. The second part is solving the linear system

using the packages present in the Trilinos library. The third is the transfer of the solution from Trilinos to Gridap.

#### 4.4.2.1 Matrix assembly

The distributed matrices in GridapDistributed are stored as a psparse matrix, and the distributed vector is stored as a pvector. These data types are implemented in the PartitionedArrays.jl package. For the psparse matrix, a local sparse matrix is stored in a CSC (Compressed Sparse Column) format along with its associated row map and column map. For the pvector, a local vector along with its associated row map is stored.

In Trilinos, we use the Tpetra back end to store the distributed linear system. Here, the matrix is stored as a Tpetra CRS (Compressed Row Sparse) matrix, and the vector is stored as a Tpetra vector. While assembling the system, it is important for the row map to be a one-to-one map. This means no ghost nodes must be there for the row map.

From Gridap, all the required data such as these maps and the local components of the vectors and matrices are passed as pointers to the C++ executable. Using this data, we build the Trilinos objects of the linear system.

#### 4.4.2.2 Solver implementation

For this project, we are looking to implement preconditioners from the FROSch framework [27] and implement iterative matrix solvers from the Belos package [8]. FROSch uses the Xpetra backend [42] to store the distributed linear system. The Xpetra backend is a wrapper that allows backward compatibility with the older Epetra backend along with the newer Tpetra backend. We therefore convert the Tpetra objects created to Xpetra objects. For the scope of this thesis, we limit ourselves to using the one-level additive Schwarz preconditioner. Using these Xpetra objects, the FROSch preconditioner and Belos solver objects are created. These objects can be configured through an input XML file. We therefore maintain some generality for this implementation. A sample XML file is provided in the appendix B.4.

#### 4.4.2.3 Solution transfer

The solution is first stored as a Tpetra vector. Along with this Tpetra vector, a corresponding Tpetra map is created. Pointers to an initialized solution distributed vector from Gridap are passed to the interface script. Using the information stored in the Tpetra map and the Tpetra vector, the Gridap solution vector is updated. All these Trilinos objects created are destroyed once the solve operation is completed as all of them are present within a local scope.

Julia wrappers for initializing and finalizing Kokkos [44] are created. Using these functions, Kokkos is initialized before any implementing Trilinos solve operation from the Julia environment. After executing all the Trilinos solve operations, Kokkos is finalized from the Julia environment.

### 4.4.3 Results

In this section, we will discuss convergence and performance results of the parallel block preconditioner implementation. We first present results showcasing the robustness of the proposed block preconditioners, and then we proceed to showcase weak and strong scaling of the system. All these tests have been conducted on the DelftBlue super computer [20]. We consider the toy problem with the same parameters as mentioned in section 4.3. For all our results on the matrix solver, we showcase the results for the second timestep. Since the inner solves that are used in the preconditioner are iterative solvers, it is to be noted that the convergence tolerance set for these solvers is 1e-8. The outer solver's relative tolerance is set as 1e-6 and its absolute tolerance is set to be 1e-8.

For the first numerical experiment, we try to understand the convergence of the outer solver while applying the Dirichlet-Neumann preconditioner compared with the Robin-Neumann preconditioner. As mentioned in 3.21, we consider the cases of the Robin-Neumann preconditioner with $\gamma$ being 0.1, 1, and 10. We repeat this experiment by varying the solid density to understand how this affects convergence.

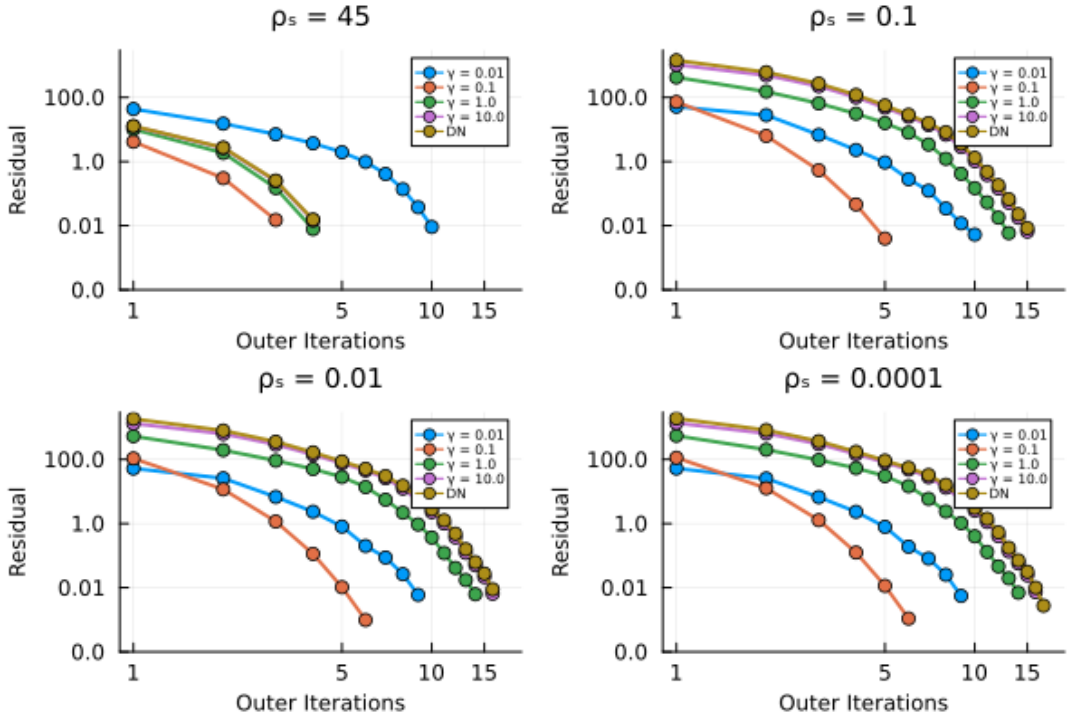



FIGURE 4.7: Convergence plots for different FSI block preconditioners and solid densities

From plot 4.7, the Robin-Neumann preconditioner with $\gamma$ being 0.1 exhibits the best convergence for all solid densities. This result shows us that choosing the Robin parameter is very important to achieve optimal convergence. Through this experiment, we have identified that the optimal value for $\gamma$ is 0.1 for the given FSI toy problem. For further experiments in this section, we will be using this optimal Robin parameter while defining the Robin-Neumann preconditioner.

The number of outer iterations that are required for the Dirichlet-Neumann preconditioner increases from 4 to 17 as we reduce the solid density. Similarly, the number of outer iterations that are required for the optimal Robin-Neumann preconditioner only increases from 3 to 6. This result clearly demonstrates the superior convergence that is exhibited by the Robin-Neumann preconditioner when compared to the Dirichlet-Neumann preconditioner, especially when the added mass effect is high.

For the next numerical experiment, we demonstrate the convergence of the Robin Neumann preconditioner when we increase the length of the solid domain. It is to be noted that as we increase the length of the solid domain, the added mass effects increase. The solid density is assumed to be 1e-4 $g/cm^3$ for this experiment.

| | L = 6 cm | L = 50 cm | L = 100 cm | L = 1000 cm |
|---|---|---|---|---|
| Dirichlet-Neumann | 17 | 38 | 52 | 157 |
| Robin-Neumann | 6 | 6 | 6 | 6 |

TABLE 4.5: Number of outer iterations for FSI block preconditioners

From table 4.5, it is very evident that the number of iterations to convergence for the Dirichlet-Neumann preconditioner deteriorates significantly as the length of the domain is increased. On the other hand, we notice that the number of iterations to convergence for the Robin-Neumann preconditioner remains constant as the length of the domain is increased.

The next numerical experiments move on to assess the parallel scalability of the system. For these tests, we limit ourselves to testing only the Robin-Neumann preconditioner.

Strong scaling [2] is a test where the problem size remains fixed and we increase the number of MPI ranks required to simulate the problem. A system is said to demonstrate strong scaling if the time taken to solve the problem proportionately decreases as the MPI processes used to simulate increases. The next numerical experiment is designed to test the strong scaling of the proposed system. For this experiment, we set the domain length to be 4000 $cm$, with each centimeter containing seven elements. The domain height is set to be 5 $cm$. The solid density is taken to be 1e-4 $g/cm^3$.

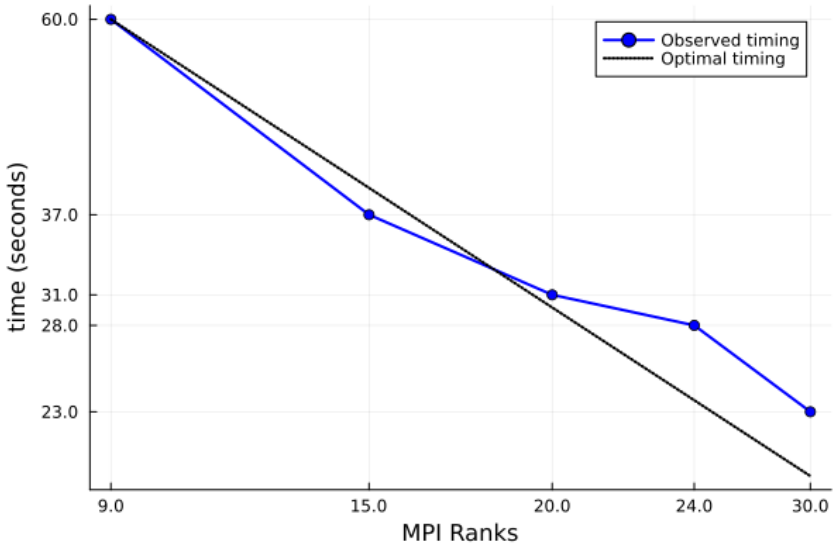



FIGURE 4.8: Strong Scaling for the proposed parallel block preconditioner system

From plot 4.8, we observe that the proposed system and its implementation demonstrate behavior close to strong scaling.

From the above problem, we also try to analyze the inner matrix solvers that are used to solve the fluid and solid blocks within the preconditioner. A one-level Schwarz preconditioner is implemented using FROSch along with a GMRES solver implemented using the Belos package. The local solve operation on each subdomain is carried out by the UMFPACK direct matrix solver [18] implemented using the Amesos 2 package. The overlap that is used to build the overlapping subdomains is set to one layer. The convergence tolerance of this inner solver is set to 1e-8.

Detailed convergence analysis for the one-level Schwarz preconditioner, when used within the block preconditioner, has not been done due to time constraints. This analysis is important to ascertain whether a two-level Schwarz preconditioner is needed for the problem or not. There are both benefits and drawbacks of using the two-level preconditioner. The benefit is that we achieve superior convergence with an upper bound to the condition number of the system. The drawback is that it takes computational resources to construct the coarse space. For a specific problem, we would have these benefits and drawbacks to decide what to use. For this thesis project, we have stuck to using a one-level Schwarz preconditioner.

Weak scaling [25] is a test, where for each process a similar size problem is given. So as we increase the number of MPI processes used to simulate, the problem size proportionally increases. A problem demonstrates weak scaling if the time taken to solve remains constant as the number of MPI processes is increased. For the next numerical experiment, we try to demonstrate the weak scaling behavior of the proposed system. Here we take the length of the subdomain to be a hundred times the number of MPI processes used to simulate the problem.

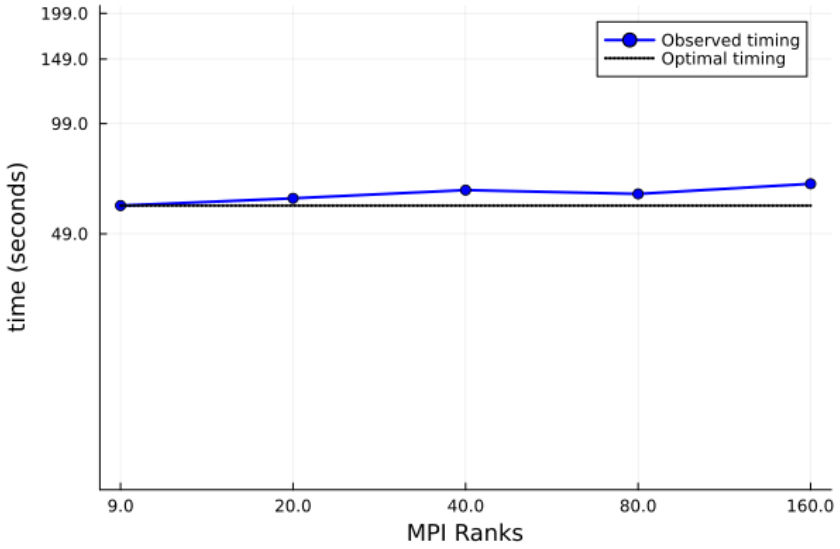



FIGURE 4.9: Weak Scaling for the proposed parallel block preconditioner system

From plot 4.9, we observe that the proposed system and its implementation demonstrate behavior close to weak scaling.

## 4.5 Floating structure problem

In this section, we demonstrate the simulation of two test cases of the floating structure problem. In the first test case, we introduce a non-reflecting boundary condition at one end to damp the incoming wave to prevent reflections. For the second case, we introduce periodic boundary conditions for simulating the response of an infinite floating membrane. We refer to section 3.4.1 for the problem.

### 4.5.1 Test case - 1

In this test case, we simulate a floating membrane object by forcing a wave at the inlet boundary and introducing a non-reflective boundary condition at the outlet boundary. Refer to table 4.6 for the terms used in this simulation. The following forcing Dirichlet condition on the wave equation for the solid displacement is imposed at the inlet.

$$d = 0.01sin(\omega t)$$

| Term | Value | Term | Value |
|---|---|---|---|
| $\rho_s$ | 100 $kg/m^3$ | $\rho_f$ | 1000 $kg/m^3$ |
| $h_s$ | 0.01 m | H | 1.1 m |
| $T$ | $0.9\rho_f g$ $1/m^2$ | $k_\lambda$ (wave number) | 10 $rad/m$ |
| Number of elements along x axis | 219 | Time step | 0.01 s |
| Number of elements along y axis | 77 | g | 9.81 $m/s^2$ |
| L | 31.4 m | $\omega$ | $sqrt(gk_\lambda tanh(k_\lambda H))$ |

TABLE 4.6: Values used to simulate the floating structure problem (Test Case-1)

Free slip boundary condition is applied on the bed for the velocity. No boundary condition is imposed on the pressure field. A non-reflective boundary condition is imposed on the outlet boundary. The dynamic pressure, velocity, and solid displacement are damped to zero using a damping function.

Let us say that we will introduce this damping from a length $L_d$ which is set to be 15.7 m. Then we introduce a relation $f = \frac{x-L_d}{Lf-Ld}$ when $x > L_d$. For all other regions, this variable is set to zero. Now we introduce the damping function as the following. We have adapted this damping function from [47].

$$damp(x) = (1 - c_1 f^2)(\frac{1 - \exp{(c_2 f^2)}}{1 - \exp{c_2}})$$

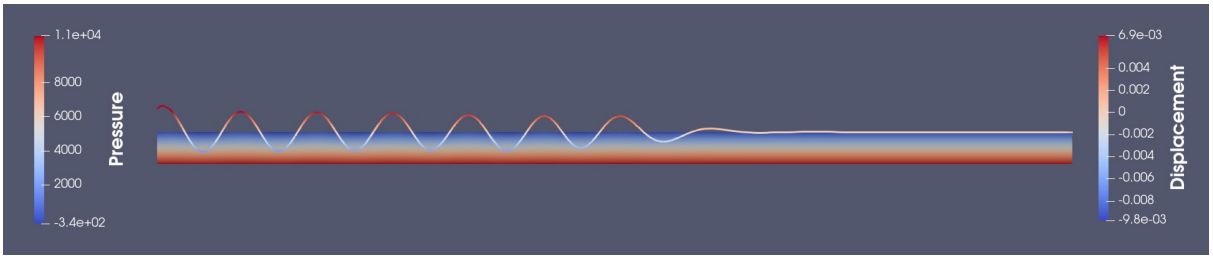



FIGURE 4.10: Pressure Contour plot along with solid displacement at time 3.4 seconds

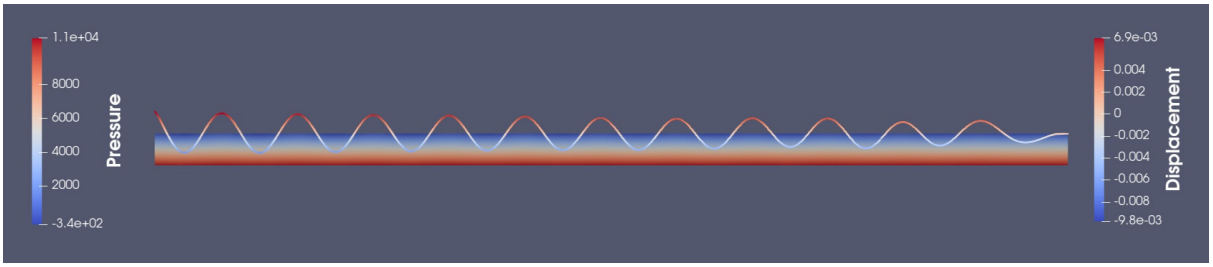



FIGURE 4.11: Pressure Contour plot along with solid displacement at time 5.8 seconds

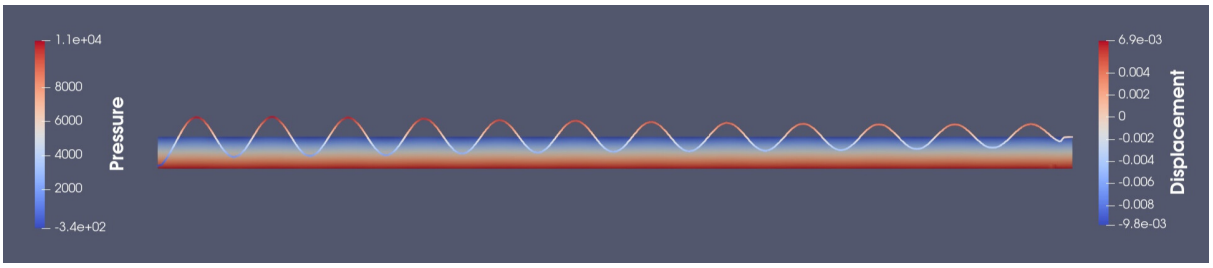



FIGURE 4.12: Pressure Contour plot along with solid displacement at time 10.0 seconds

### 4.5.2 Test case - 2

In this test case, we implement periodic boundary conditions to simulate an infinite floating membrane. The methodology of implementing this problem has been adapted from [15].

| Term | Value | Term | Meaning |
|---|---|---|---|
| $\rho_s$ | 100 $kg/m^3$ | $\rho_f$ | 1000 $kg/m^3$ |
| $h_s$ | 0.01 m | H | 1.1 m |
| Number of elements along x axis | 219 | $k_\lambda$ (wave number) | 1 $rad/m$ |
| Number of elements along y axis | 77 | Time step | 0.01 s |
| L | $10\pi$ m | g | 9.81 $m/s^2$ |
| $\omega$ | $sqrt(gk_\lambda tanh(k_\lambda H))$ | $\eta_0$ | 0.01 |

TABLE 4.7: Values used to simulate the floating structure problem (Test Case-2)

A traveling wave solution for free surface flow in terms of the fluid potential is given as the following relation. This relation is prescribed using the linear Airy's wave theory.

$$\phi\left(\left(x,y\right),t\right) = -\frac{\eta_0\omega}{k_\lambda}\frac{\cosh\left(k_\lambda y\right)}{\sinh\left(k_\lambda H\right)}\sin\left(k_\lambda x-\omega t\right)$$

Similarly, the following relation gives us the solid displacement solution (wave equation) on the membrane.

$$d\left(\left(x\right),t\right) = \eta_0\cos\left(k_\lambda x-\omega t\right)$$

Using the fluid potential relation, we can formulate relations for the fluid velocity and pressure. These fields are used as initial boundary conditions for this problem. It is also important to note that the length of the domain be an integral multiple of the wavelength of the wave. Using these relations, we find a relation to the pre-tension parameter in the membrane equation.

$$T = \frac{\omega^2}{k_\lambda^2}\left(\frac{\rho_f}{k_\lambda\tanh(k_\lambda H)}+\rho_s h_s\right)$$

Free slip boundary condition on the fluid velocity is applied on the bed. The fluid velocity relations derived earlier are applied on both the inlet and outlet boundaries as Dirichlet boundary conditions. The solid displacement relation is applied as a Dirichlet boundary condition on the two ends of the domain.

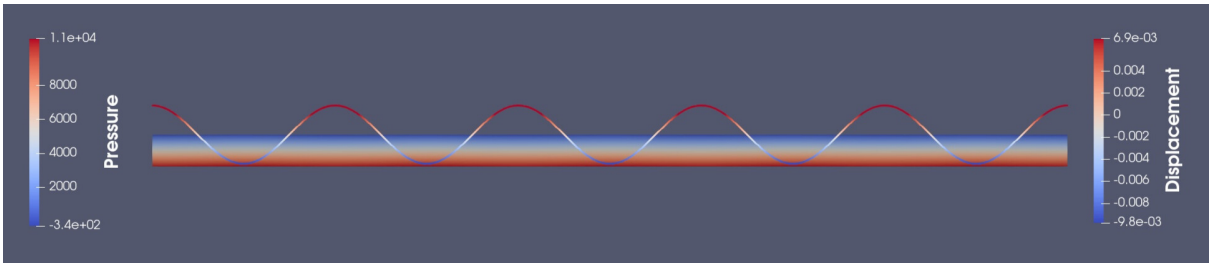



FIGURE 4.13: Pressure Contour plot along with solid displacement at time 0 seconds

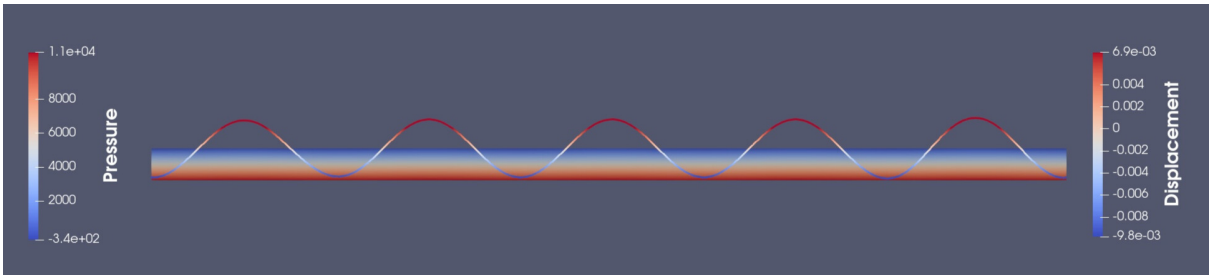



FIGURE 4.14: Pressure Contour plot along with solid displacement at time 1.1 seconds

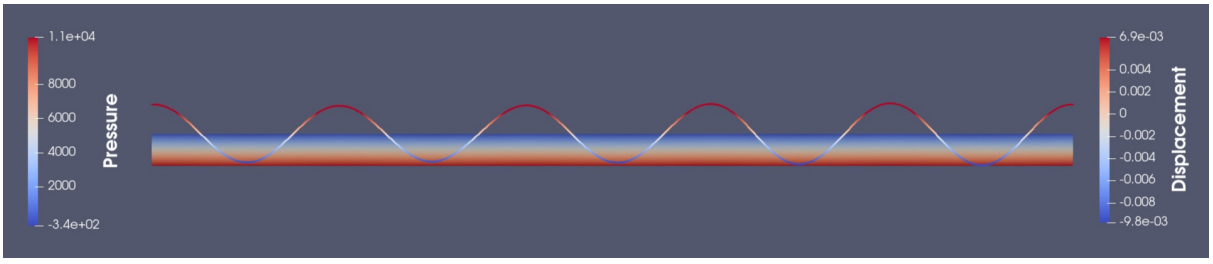



FIGURE 4.15: Pressure Contour plot along with solid displacement at time 2.2 seconds

# Chapter 5

# Conclusion

This thesis addresses the challenge of efficiently simulating problems that are similar to simulating very large floating structures on large parallel computing hardware. To achieve this, we develop parallel preconditioners based on overlapping and non-overlapping domain decomposition methods.

In chapter 2, we introduce these domain decomposition methods and their application to simple elliptic PDE problems. Block preconditioners are developed using the concepts of non-overlapping domain decomposition methods. We discuss the convergence of these block preconditioners when used with the Richardson iteration as this scheme is similar to the continuous algorithm behind the non-overlapping methods. Overlapping methods such as the one-level Schwarz and the two-level Schwarz based on the Lagrangian coarse space are discussed.

In chapter 3, we introduce the FSI problems and develop preconditioners for them. Based on the concept of non-overlapping methods, we investigate various coupling schemes that couple the fluid and solid domains for this coupled problem. We investigate the added-mass effect and how this impacts the convergence while solving. We then present various block preconditioners based on these coupling strategies. It is seen that the Robin-Neumann preconditioner is not affected by the added-mass effect, while the other preconditioners, such as the Dirichlet-Neumann preconditioner, are adversely affected by the added mass effect.

In chapter 4, we discuss how we implement the methods discussed in the previous chapters and the associated results. We first show implementations and results of the DD methods implemented for the simple elliptic problems discussed in chapter 2. We then proceed to show our implementation for the FSI coupling through matrix modifications in Gridap. To achieve this, we introduce a new function that provides the DOF IDs of all the nodes present on the interface. We also introduce the methodology used to impose the interface conditions while coupling the system via matrix modifications. This is introduced as a solver wrapper in Gridap.

The proposed FSI block preconditioner is implemented on a toy FSI problem to understand its convergence properties. We first implement these block preconditioners on a Richardson iteration matrix solver as this is similar to the continuous algorithm.

We then move on to the parallel implementation of these block preconditioners. To solve the inner blocks in these block preconditioners, we use a Krylov iterative solver with a one-level Schwarz preconditioner. These inner solves are implemented using the Trilinos library. To achieve this, we implement a Gridap-Trilinos interface and a solver wrapper through which we can solve linear systems in Gridap using Trilinos. We discuss this implementation in chapter 4. We also discuss the modifications required to implement the FSI coupling through matrix modification for the parallel case. Krylov solvers are implemented as outer solvers.

Using this developed preconditioning system, we first try to understand how its convergence is affected when we change certain parameters in the Toy FSI problem such as the solid density or the domain length. These results clearly show us that the preconditioner system based on the Robin-Neumann method offers the best convergence properties for this problem. We successfully demonstrated the strong and weak scaling behavior of the system and its implementation.

We finally showcase a floating structure-based problem using the implementations we have developed. Through this thesis, we have introduced efficient preconditioning systems through which we can simulate large problems similar to the floating structure problem.

Future research can explore developing similar preconditioners for more complex floating structure problems involving nonlinearities as we have restricted ourselves to linear problems. This thesis project was also limited to a conformal fluid-solid grid. Coupling implementations for non-conformal grids can be implemented to increase the generality of the implementation in Gridap.

# Bibliography


[1] Shagun Agarwal, Oriol Colomés, and Andrei V. Metrikine. "Dynamic analysis of viscoelastic floating membranes using monolithic Finite Element method". In: *Journal of Fluids and Structures* 129 (July 2024), p. 104167. DOI: 10.1016/j.jfluidstructs.2024.104167. URL: https://doi.org/10.1016/j.jfluidstructs.2024.104167.

[2] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, Reprinted from the AFIPS Conference Proceedings, Vol. 30 (Atlantic City, N.J., Apr. 18–20), AFIPS Press, Reston, Va., 1967, pp. 483–485, when Dr. Amdahl was at International Business Machines Corporation, Sunnyvale, California". In: *IEEE Solid-State Circuits Society Newsletter* 12.3 (2007), pp. 19–20. DOI: 10.1109/N-SSC.2007.4785615.

[3] Ivo Babuška. "Error-Bounds for Finite Element Method". In: *Numerische Mathematik* 16 (1971), pp. 322–333. URL: http://eudml.org/doc/132037.

[4] Santiago Badia, Alberto F Martín, and Francesc Verdugo. "GridapDistributed: a massively parallel finite element toolbox in Julia". In: *J. Open Source Softw.* 7.74 (June 2022), p. 4157.

[5] Santiago Badia, Fabio Nobile, and Christian Vergara. "Fluid–structure partitioned procedures based on Robin transmission conditions". en. In: *J. Comput. Phys.* 227.14 (July 2008), pp. 7027–7051.

[6] Santiago Badia, Fabio Nobile, and Christian Vergara. "Robin–Robin preconditioned Krylov methods for fluid–structure interaction problems". In: *Computer Methods in Applied Mechanics and Engineering* 198.33–36 (Apr. 2009), 2768–2784. DOI: 10.1016/j.cma.2009.04.004. URL: https://doi.org/10.1016/j.cma.2009.04.004.

[7] Santiago Badia, Annalisa Quaini, and Alfio Quarteroni. "Modular vs. non-modular preconditioners for fluid–structure systems with large added-mass effect". In: *Computer Methods in Applied Mechanics and Engineering* 197.49–50 (May 2008), 4216–4232. DOI: 10.1016/j.cma.2008.04.018. URL: https://doi.org/10.1016/j.cma.2008.04.018.

[8] Eric Bavier et al. “Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems”. In: *Scientific Programming* 20.3 (2012), p. 243875. DOI: https://doi.org/10.3233/SPR-2012-0352. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.3233/SPR-2012-0352. URL: https://onlinelibrary.wiley.com/doi/abs/10.3233/SPR-2012-0352.

[9] Jean-David Benamou. “A Domain Decomposition Method with Coupled Transmission Conditions for the Optimal Control of Systems Governed by Elliptic Partial Differential Equations”. In: *SIAM Journal on Numerical Analysis* 33.6 (Dec. 1996), 2401–2416. DOI: 10.1137/s0036142994267102. URL: https://doi.org/10.1137/s0036142994267102.

[10] Michele Benzi and Maxim A. Olshanskii. “An augmented Lagrangian-Based approach to the Oseen problem”. In: *SIAM Journal on Scientific Computing* 28.6 (Jan. 2006), 2095–2113. DOI: 10.1137/050646421. URL: https://doi.org/10.1137/050646421.

[11] Petter E. Bjørstad et al. *Domain Decomposition Methods in Science and Engineering XXIV*. Springer, Jan. 2019.

[12] P Causin, J F Gerbeau, and F Nobile. “Added-mass effect in the design of partitioned algorithms for fluid–structure problems”. en. In: *Comput. Methods Appl. Mech. Eng.* 194.42-44 (Oct. 2005), pp. 4506–4527.

[13] Wenbin Chen. “On the Optimal Convergence Rate of a Robin-Robin Domain Decomposition Method”. In: *Journal of Computational Mathematics* 32.4 (June 2014), 456–475. DOI: 10.4208/jcm.1403-m4391. URL: https://doi.org/10.4208/jcm.1403-m4391.

[14] J. Chung and G. M. Hulbert. “A Time Integration Algorithm for Structural Dynamics With Improved Numerical Dissipation: The Generalized- Method”. In: *Journal of Applied Mechanics* 60.2 (1993), pp. 371–375. DOI: 10.1115/1.2900803.

[15] Oriol Colomés, Francesc Verdugo, and Ido Akkerman. “A monolithic finite element formulation for the hydroelastic analysis of very large floating structures”. In: *International Journal for Numerical Methods in Engineering* 124.3 (Oct. 2022), 714–751. DOI: 10.1002/nme.7140. URL: https://doi.org/10.1002/nme.7140.

[16] Melvin Creff and Jean-Luc Guermond. “Preconditioning of the generalized Stokes problem arising from the approximation of the time-dependent Navier-Stokes equations”. In: *arXiv preprint arXiv:2407.01783* (2024). URL: https://arxiv.org/abs/2407.01783.

[17] Paolo Crosetto. “Fluid-Structure Interaction Problems in Hemodynamics: Parallel Solvers, Preconditioners, and Applications”. In: (Jan. 2011).

[18] Timothy A. Davis. “Algorithm 832: UMFPACK V4.3—an Unsymmetric-Pattern Multifrontal Method”. In: *ACM Transactions on Mathematical Software* 30.2 (2004), pp. 196–199. DOI: 10.1145/992200.992206. URL: https://doi.org/10.1145/992200.992206.

[19] Simone Deparis et al. “Fluid–structure algorithms based on Steklov–Poincaré operators”. In: *Computer Methods in Applied Mechanics and Engineering* 195.41–43 (Feb. 2006), 5797–5812. DOI: 10.1016/j.cma.2005.09.029. URL: https://doi.org/10.1016/j.cma.2005.09.029.

[20] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024.

[21] Miguel A. Fernández, Jimmy Mullaert, and Marina Vidrascu. “Explicit Robin–Neumann schemes for the coupling of incompressible fluids with thin-walled structures”. In: *Computer Methods in Applied Mechanics and Engineering* 267 (Oct. 2013), 566–593. DOI: 10.1016/j.cma.2013.09.020. URL: https://doi.org/10.1016/j.cma.2013.09.020.

[22] Martin J. Gander. “Schwarz methods over the course of time”. In: *Electronic Transactions on Numerical Analysis* 31 (2008). Available at https://www.unige.ch/~gander/Preprints/SchwarzHistorical.pdf, pp. 228–255.

[23] Luca Gerardo-Giorda, Fabio Nobile, and Christian Vergara. “Analysis and optimization of Robin–Robin partitioned procedures in Fluid-Structure Interaction Problems”. In: *SIAM Journal on Numerical Analysis* 48.6 (Jan. 2010), 2091–2116. DOI: 10.1137/09076605x. URL: https://doi.org/10.1137/09076605x.

[24] W. Guo and L. S. Hou. “Generalizations and Accelerations of Lions’ Nonoverlapping Domain Decomposition Method for Linear Elliptic PDE”. In: *SIAM Journal on Numerical Analysis* 41.6 (Jan. 2003), 2056–2080. DOI: 10.1137/s0036142902407150. URL: https://doi.org/10.1137/s0036142902407150.

[25] John L. Gustafson. “Reevaluating Amdahl’s law”. In: *Commun. ACM* 31.5 (May 1988), 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415. URL: https://doi.org/10.1145/42411.42415.

[26] Alexander Heinlein, Christian Hochmuth, and Axel Klawonn. “Reduced dimension GDSW coarse spaces for monolithic Schwarz domain decomposition methods for incompressible fluid flow problems”. In: *International Journal for Numerical Methods in Engineering* 121.6 (Oct. 2019), 1101–1119. DOI: 10.1002/nme.6258. URL: https://doi.org/10.1002/nme.6258.

[27] Alexander Heinlein, Axel Klawonn, and Oliver Rheinbach. “A parallel implementation of a two-level overlapping Schwarz method with energy-minimizing coarse space based on Trilinos”. In: *SIAM J. Sci. Comput.* 38.6 (2016). Preprint http://tu-freiberg.de/sites/default/files/media/fakultaet-fuer-mathematik-und-informatik-fakultaet-1-9277/prep/2016-04_fertig.pdf, pp. C713–C747. DOI: 10.1137/16M1062843.

[28] JuliaInterop. *CxxWrap.jl: Package to make C++ libraries accessible from Julia*. https://github.com/JuliaInterop/CxxWrap.jl. Accessed: 2025-05-06. 2015.

[29] Axel Klawonn and Luca F. Pavarino. "Overlapping Schwarz methods for mixed linear elasticity and Stokes problems". In: *Computer Methods in Applied Mechanics and Engineering* 165.1–4 (Nov. 1998), 233–245. DOI: 10.1016/s0045-7825(98)00059-0. URL: https://doi.org/10.1016/s0045-7825(98)00059-0.

[30] Ulrich Küttler and Wolfgang A. Wall. "Fixed-point fluid–structure interaction solvers with dynamic relaxation". In: *Computational Mechanics* 43.1 (Feb. 2008), 61–72. DOI: 10.1007/s00466-008-0255-5. URL: https://doi.org/10.1007/s00466-008-0255-5.

[31] Mikel Landajuela et al. "Coupling schemes for the FSI forward prediction challenge: Comparative study and validation". en. In: *Int. J. Numer. Method. Biomed. Eng.* 33.4 (Apr. 2017), e2813.

[32] Jordi Manyer, Alberto F Martín, and Santiago Badia. "GridapSolvers.jl: Scalable multiphysics finite element solvers in Julia". In: *J. Open Source Softw.* 9.102 (Oct. 2024), p. 7162.

[33] L D Marini and A Quarteroni. "A relaxation procedure for domain decomposition methods using finite elements". en. In: *Numer. Math. (Heidelb.)* 55.5 (Sept. 1989), pp. 575–598.

[34] L. D. Marini and A. Quarteroni. "A relaxation procedure for domain decomposition methods using finite elements". In: *Numerische Mathematik* 55.5 (Sept. 1989), 575–598. DOI: 10.1007/bf01398917. URL: https://doi.org/10.1007/bf01398917.

[35] Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for partial differential equations*. May 1999. DOI: 10.1093/oso/9780198501787.001.0001. URL: https://doi.org/10.1093/oso/9780198501787.001.0001.

[36] Sivasankaran Rajamanickam, Erik G. Boman, and Michael A. Heroux. "ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms". In: *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IPDPS '12. USA: IEEE Computer Society, 2012, 631–643. ISBN: 9780769546759. DOI: 10.1109/IPDPS.2012.64. URL: https://doi.org/10.1109/IPDPS.2012.64.

[37] M ur Rehman et al. "On iterative methods for the incompressible Stokes problem". en. In: *Int. J. Numer. Methods Fluids* 65.10 (Apr. 2011), pp. 1180–1200.

[38] Lewis Fry Richardson and Richard Tetley Glazebrook. "IX. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam". In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 210.459-470 (1911), pp. 307–357. DOI: 10.1098/rsta.1911.0009. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.1911.0009. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1911.0009.

[39] Youcef Saad. “A flexible Inner-Outer Preconditioned GMRES algorithm”. In: *SIAM Journal on Scientific Computing* 14.2 (Mar. 1993), 461–469. DOI: 10.1137/0914028. URL: https://doi.org/10.1137/0914028.

[40] Youcef Saad and Martin H Schultz. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM J. Sci. Stat. Comput.* 7.3 (July 1986), pp. 856–869.

[41] The Trilinos Project Team. *The Trilinos Project Website*.

[42] The Xpetra Project Team. *The Xpetra Project Website*. 2024 (accessed May 7, 2025). URL: https://trilinos.github.io/xpetra.html.

[43] Andrea Toselli and Olof Widlund. *Domain Decomposition Methods - Algorithms and Theory*. Springer Science Business Media, 2006. URL: http://books.google.ie/books?id=h7EVoI2g1nkC&printsec=frontcover&dq=Domain+Decomposition+Methods:+Algorithms+and+Theory&hl=&cd=1&source=gbs_api.

[44] Christian R. Trott et al. “Kokkos 3: Programming Model Extensions for the Exascale Era”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 805–817. DOI: 10.1109/TPDS.2021.3097283.

[45] A. M. TURING. “ROUNDING-OFF ERRORS IN MATRIX PROCESSES”. In: *The Quarterly Journal of Mechanics and Applied Mathematics* 1.1 (Jan. 1948), pp. 287–308. ISSN: 0033-5614. DOI: 10.1093/qjmam/1.1.287. eprint: https://academic.oup.com/qjmam/article-pdf/1/1/287/5323145/1-1-287.pdf. URL: https://doi.org/10.1093/qjmam/1.1.287.

[46] Francesc Verdugo and Santiago Badia. “The software design of Gridap: A Finite Element package based on the Julia JIT compiler”. In: *Computer Physics Communications* 276 (July 2022), p. 108341. DOI: 10.1016/j.cpc.2022.108341. URL: https://doi.org/10.1016/j.cpc.2022.108341.

[47] B. Wasistho, B.J. Geurts, and J.G.M. Kuerten. “Simulation techniques for spatially evolving instabilities in compressible flow over a flat plate”. In: *Computers Fluids* 26.7 (Sept. 1997), 713–739. DOI: 10.1016/s0045-7930(97)00021-2. URL: https://doi.org/10.1016/s0045-7930(97)00021-2.

# Appendix

## A Supplementary results

### A.1 Non-overlapping domain-decomposition methods

#### A.1.1 Dirichlet-Neumann preconditioner

In this section, we discuss the convergence results for the problem when we apply the Dirichlet-Neumann preconditioner.
For the first experiment, we vary the interface position, keeping all other parameters constant. Here, the diffusion coefficients ($a_1$ and $a_2$) are kept to be one. The below plot shows us how the relaxation parameter affects the iteration count.

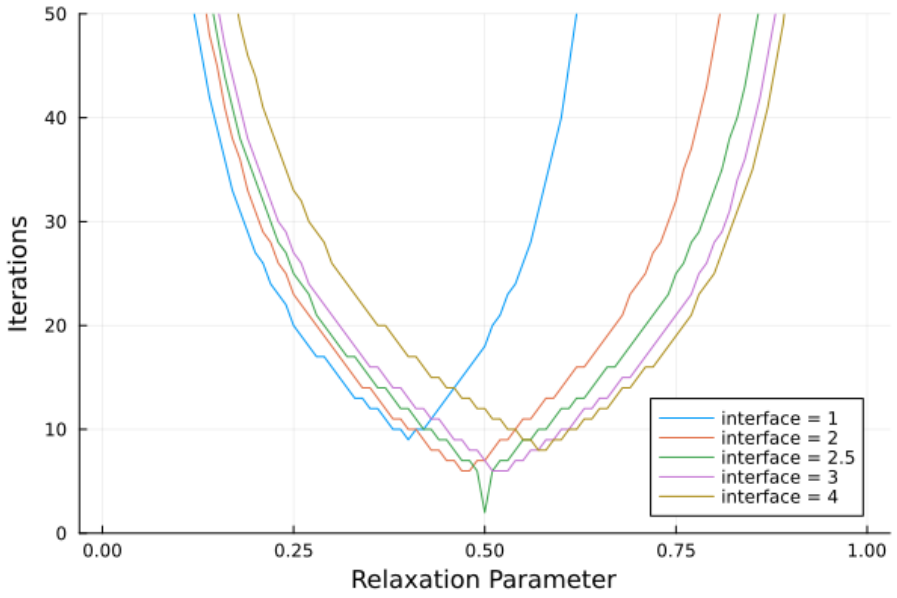



FIGURE 1: Numerical Experiment-1 — Dirichlet-Neumann iteration — Coupled Poisson Problem

We can observe that the plot moves to the left of 0.5 if the interface is moved to the left of center, and the plot moves to the right if the interface is moved to the right of the center. Also, we can

observe that if the interface is not placed symmetrically, then the range of relaxation parameters where the solution converges decreases proportionally.

For the second numerical experiment conducted on the Dirichlet-Neumann method, we vary the diffusion coefficients ($a_1$ and $a_2$) keeping the interface at the center. The below plot shows us how the relaxation parameter affects the iteration count.

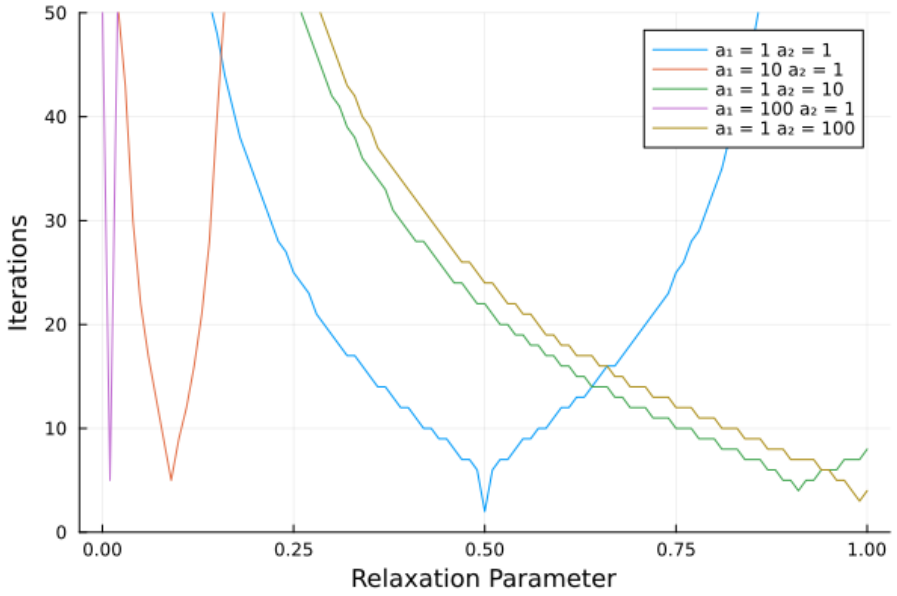



FIGURE 2: Numerical Experiment-2 — Dirichlet Neumann iteration — Coupled Poisson Problem

In section 2.3.2.2, we have derived the optimum relaxation parameter that is possible if the interface is kept at the center. The above plot validates these findings. We can observe that the range of relaxation parameters that converges to the solution drastically reduces as $a_1 >> a_2$. Another observation is that, if $a_2 > a_1$, then the solution converges even if we have no relaxation. We can deduce that the Dirichlet-Neumann method's convergence is very sensitive to the relaxation parameter chosen.

For the third numerical experiment conducted on the Dirichlet-Neumann method, we vary the interface position for two cases where the diffusion coefficients are kept different in each subdomain. The below plot shows us how the relaxation parameter affects the iteration count.

From the plot 3 shown below, we can observe that the Dirichlet-Neumann method does not do very well when $a_1 > a_2$ as the choice of relaxation parameters necessary for convergence reduces. We can also observe that the plot moves as we move the interface, but the impact of moving

the interface on convergence is significantly less than changing the diffusion coefficients of the problem.

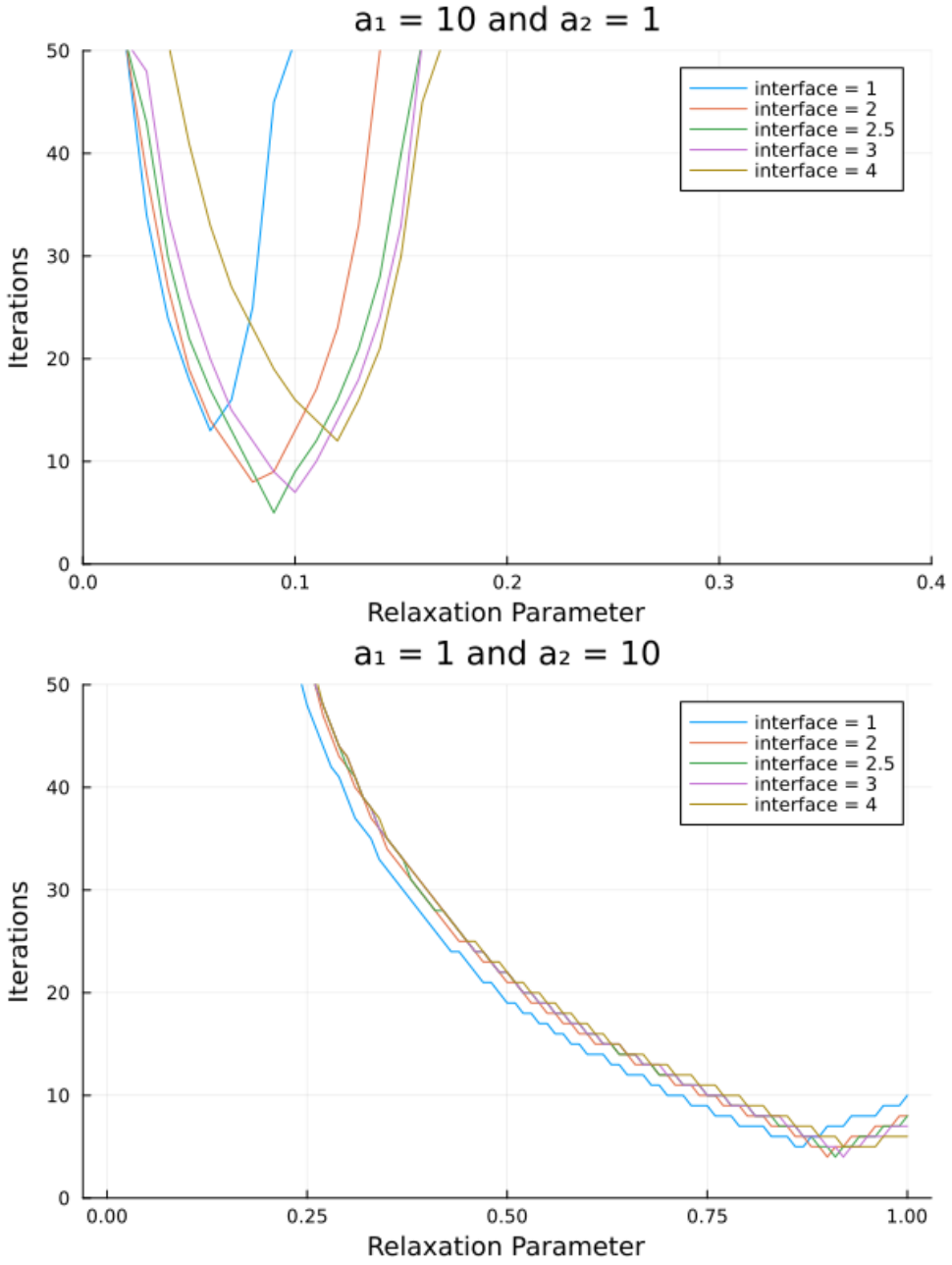



FIGURE 3: Numerical Experiment-3 — Dirichlet Neumann iteration — Coupled Poisson Problem

### A.1.2 Robin-Neumann preconditioner

In this section, we discuss the convergence results for the problem when we apply the Robin-Neumann preconditioner.

For the first experiment, we try to compare how the Robin-Neumann preconditioner differs from the Dirichlet-Neumann preconditioner for the standard problem. Here, the interface is kept at

the center and the diffusion coefficients are kept to be one. The below plot shows us how the relaxation parameter affects the iteration count.

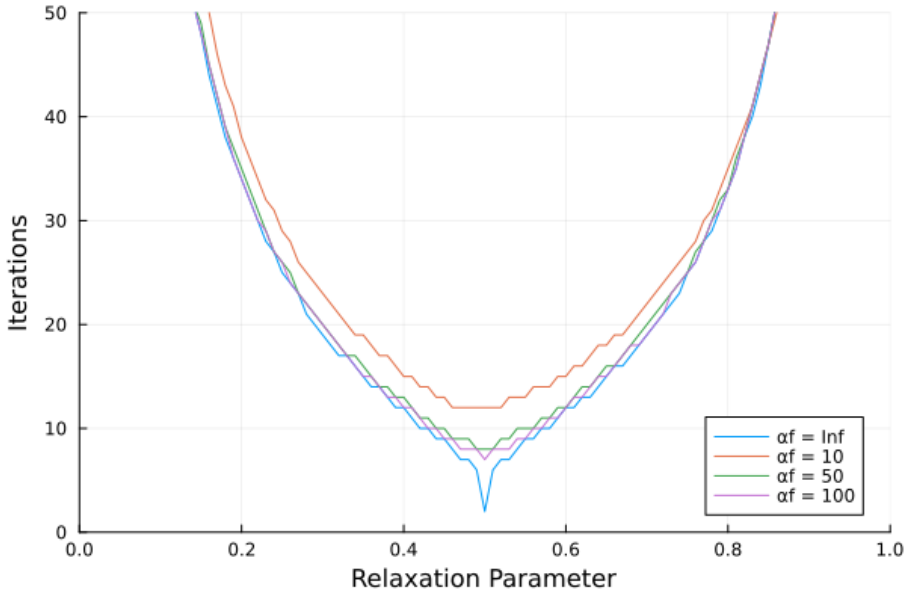



FIGURE 4: Numerical Experiment-1 —Robin-Neumann iteration — Coupled Poisson Problem

We can observe that the convergence plot largely remains similar to the Dirichlet-Neumann preconditioner when $\alpha_f$ is greater than one. It can also be observed that the convergence of all Robin-Neumann cases is worse than the Dirichlet-Neumann case ($\alpha_f = \inf$).

For the second numerical experiment conducted on the Robin-Neumann method, we keep the diffusion constants to be one and vary the interface position to understand how that impacts the preconditioner. The below plot shows us how the relaxation parameter affects the iteration count.

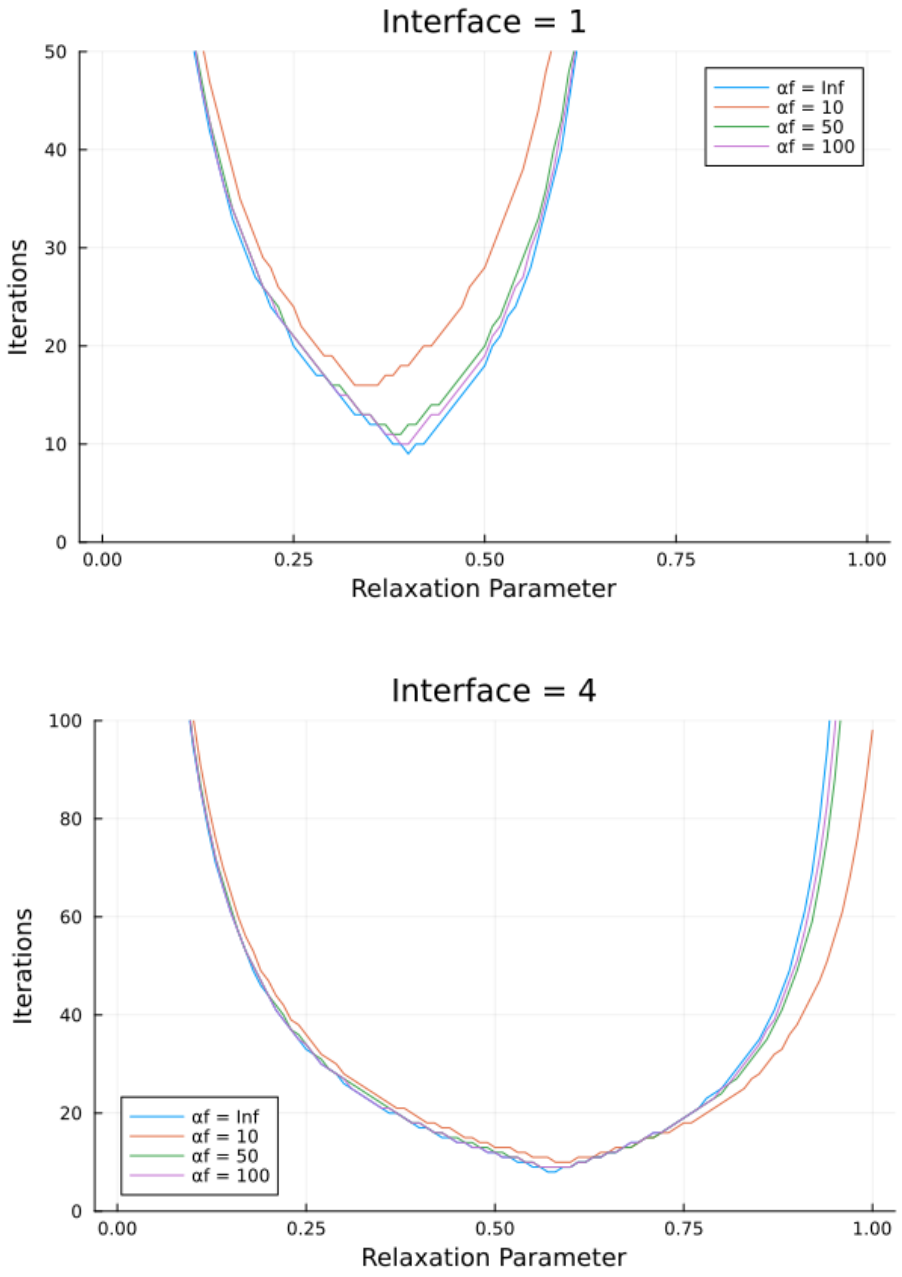



FIGURE 5: Numerical Experiment-2 — Robin-Neumann iteration — Coupled Poisson Problem

We observe that the convergence plot largely remains similar to the Dirichlet-Neumann method. We can therefore say that this method is not superior to the Dirichlet-Neumann method when the diffusion coefficients of the subdomains are the same.

For the third numerical experiment conducted on the Robin-Neumann method, we keep the interface in the center and we vary the diffusion coefficients to understand how that affects the convergence criteria of the Robin-Neumann method. The below plot shows us how the relaxation parameter affects the iteration count.

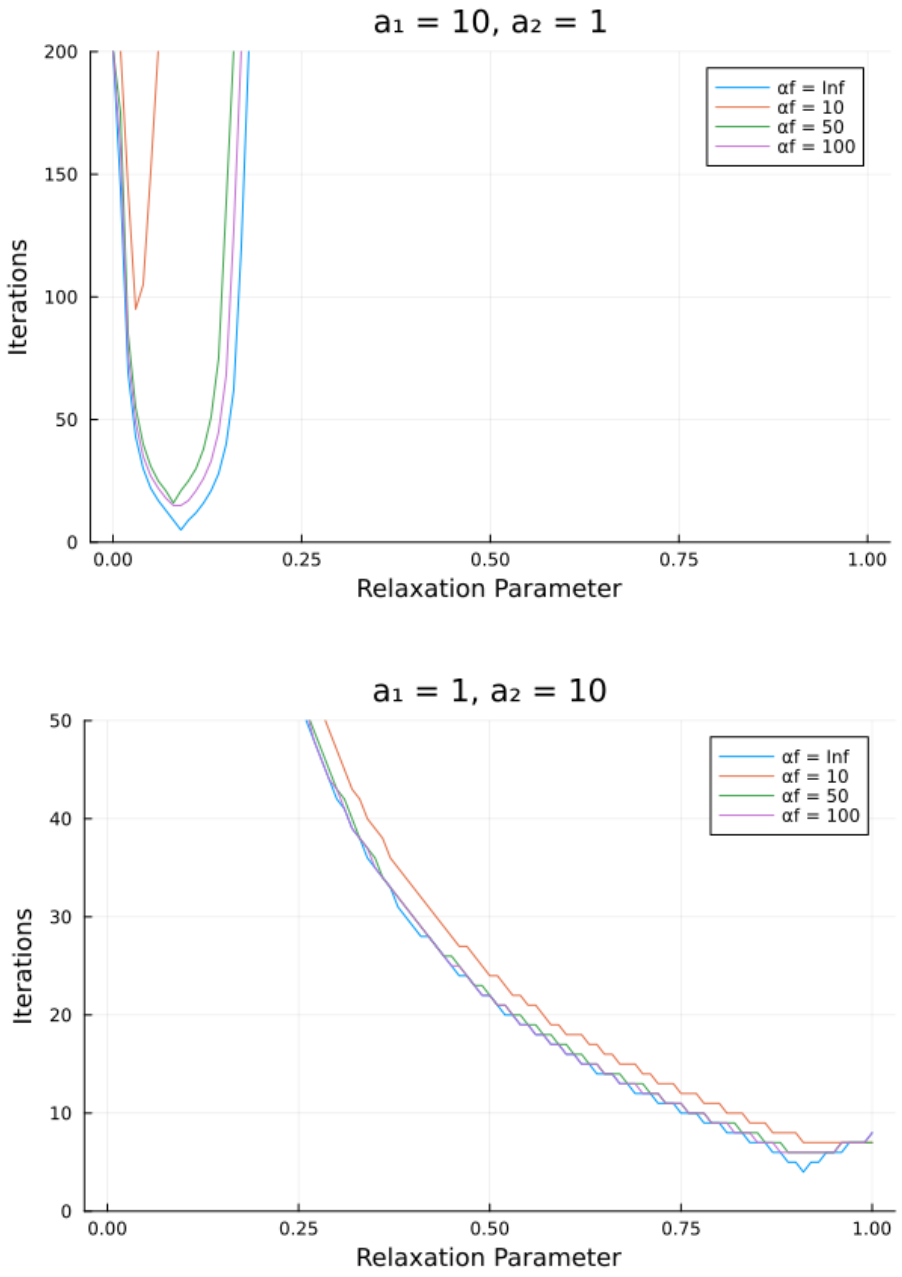



FIGURE 6: Numerical Experiment-3 — Robin-Neumann iteration — Coupled Poisson Problem

We can observe that when $a_1 > a_2$, then it is very clearly evident that the Robin-Neumann method significantly becomes worse than the Dirichlet-Neumann method as $\alpha_f$ is reduced. When $a_2 > a_1$, then we can observe that the convergence properties of these two methods remain similar.

For the fourth numerical experiment conducted on the Robin-Neumann method, we change the location of the interface to the two extremes and then we vary the diffusion coefficients to understand how that affects the convergence criteria of the Robin-Neumann method. The below plot shows us how the relaxation parameter affects the iteration count.

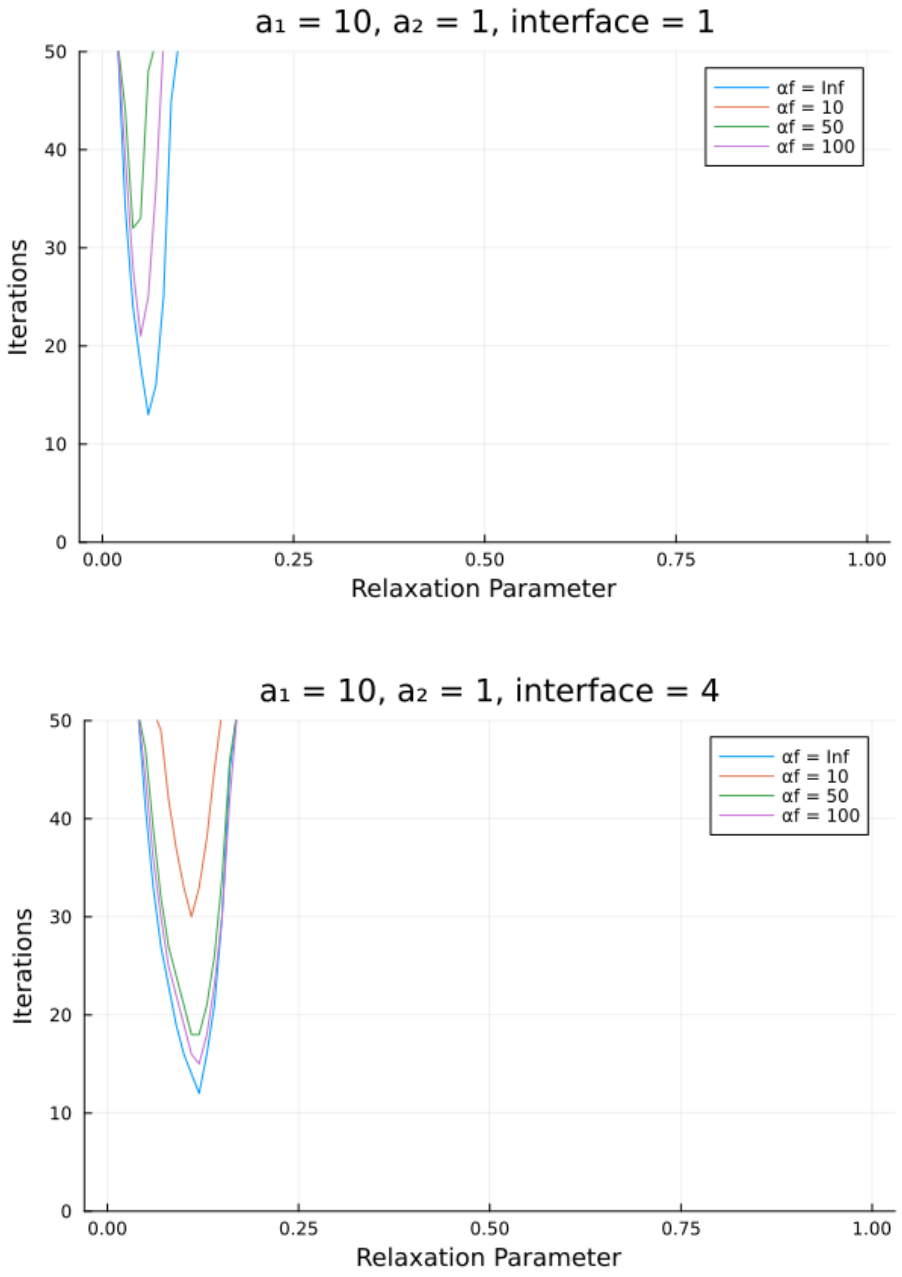



FIGURE 7: Numerical Experiment-4 (a) — Robin-Neumann iteration — Coupled Poisson Problem

From the above plots, we can observe that the convergence remains largely similar (when $a_1 > a_2$) between the Dirichlet-Neumann method and the Robin-Neumann method. The convergence has also significantly improved when compared to the numerical experiment 3. We can say that both the position of the interface and the diffusion coefficients impact the convergence criteria.

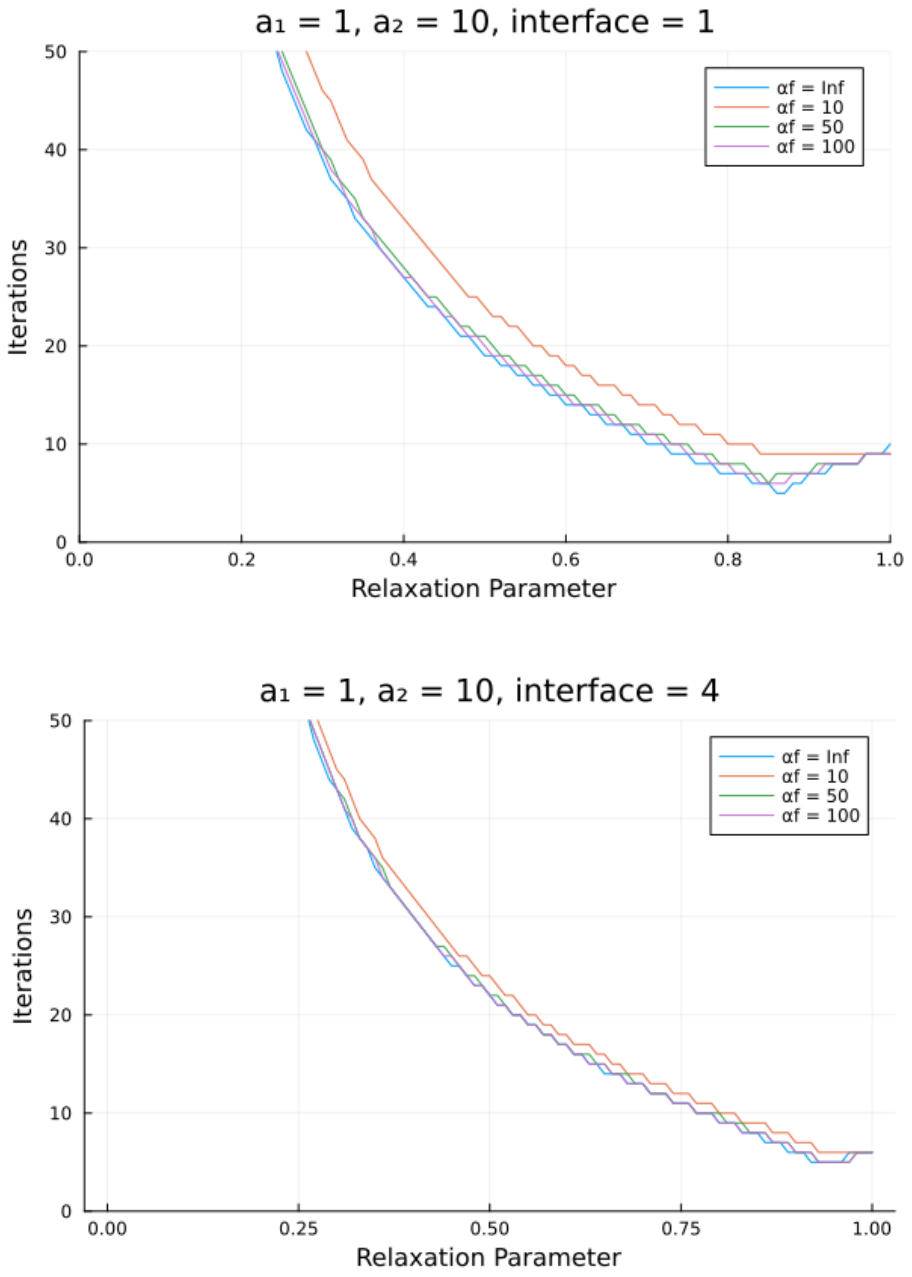



FIGURE 8: Numerical Experiment-4 (b) — Robin-Neumann iteration — Coupled Poisson Problem

From the above plots (when $a_2 > a_1$) we can say that the convergence of the Robin-Neumann method largely remains similar to the Dirichlet-Neumann method. This result is similar to what was observed in the numerical experiment 3.

### A.1.3 Dirichlet-Robin preconditioner

In this section, we discuss the convergence results for the problem when we apply the Dirichlet-Robin preconditioner.

For the first numerical experiment, we try to compare how the Dirichlet-Robin preconditioner differs from the Dirichlet-Neumann preconditioner for the standard problem. Here the interface is kept at the center, and the diffusion coefficients are kept to be one. The below plot shows us how the relaxation parameter affects the iteration count.

From plot 9, we can observe that as we change $\alpha_s$ from 0 (Dirichlet-Neumann case) to 2, the convergence plot shifts to the right. We can also observe that convergence increases when there is no relaxation as $\alpha_s$ increases.

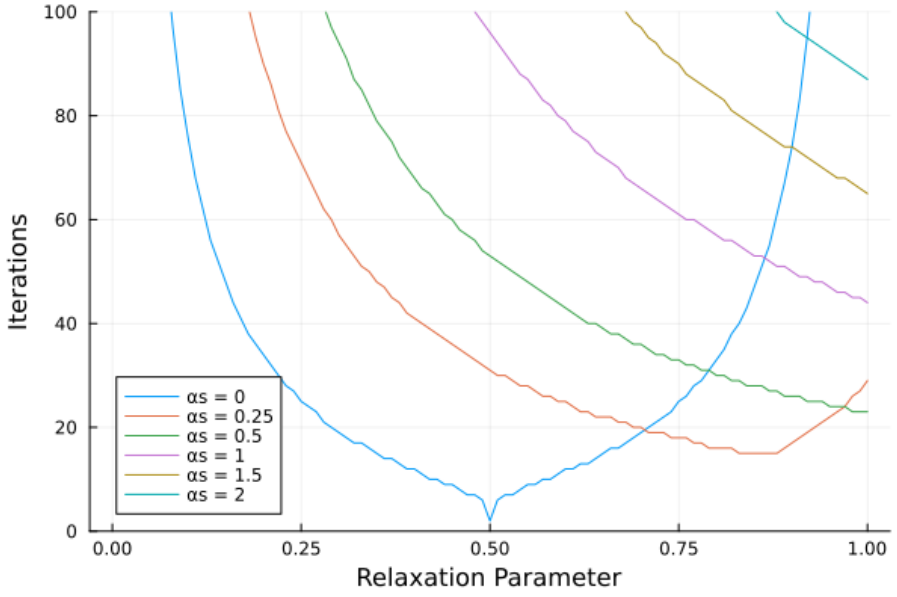



FIGURE 9: Numerical Experiment-1 —Dirichlet-Robin iteration — Coupled Poisson Problem

For the second numerical experiment conducted on the Dirichlet-Robin method, we keep the diffusion coefficient to be one and we vary the interface position to understand how it impacts the preconditioner. The below plot shows us how the relaxation parameter affects the iteration count.

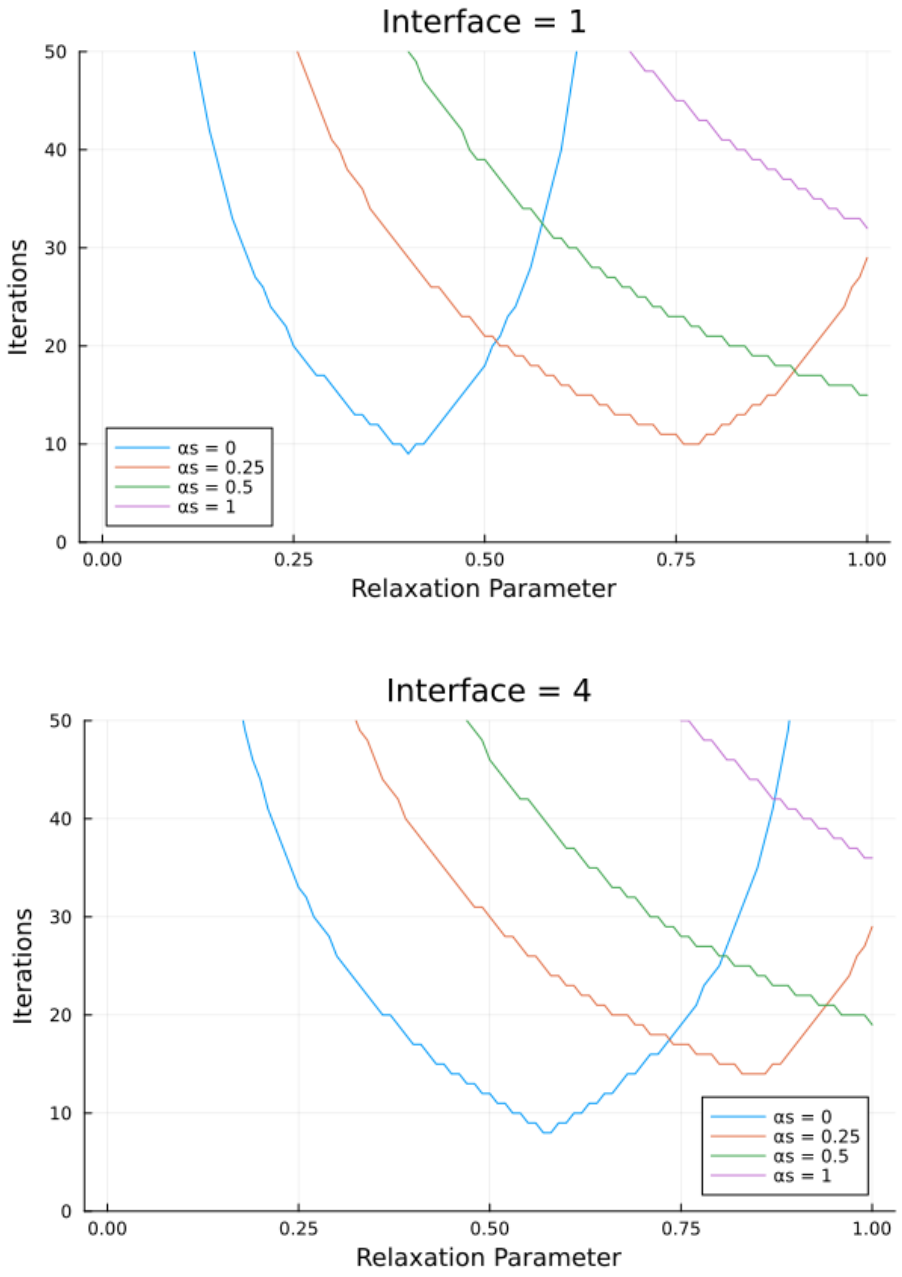



FIGURE 10: Numerical Experiment-2 — Dirichlet-Robin iteration — Coupled Poisson Problem

From the above plots, we can observe that, as we increase $\alpha_s$ from 0 (Dirichlet-Neumann) to 1, the entire convergence plot shifts to the right. The convergence behavior largely remains similar, though the best convergence is observed for the Dirichlet-Neumann method.

For the third numerical experiment conducted on the Dirichlet-Robin method, we keep the interface in the center and we vary the diffusion coefficient to understand how that affects the convergence criteria of the Dirichlet-Robin method. The below plot shows us how the relaxation parameter affects the iteration count.

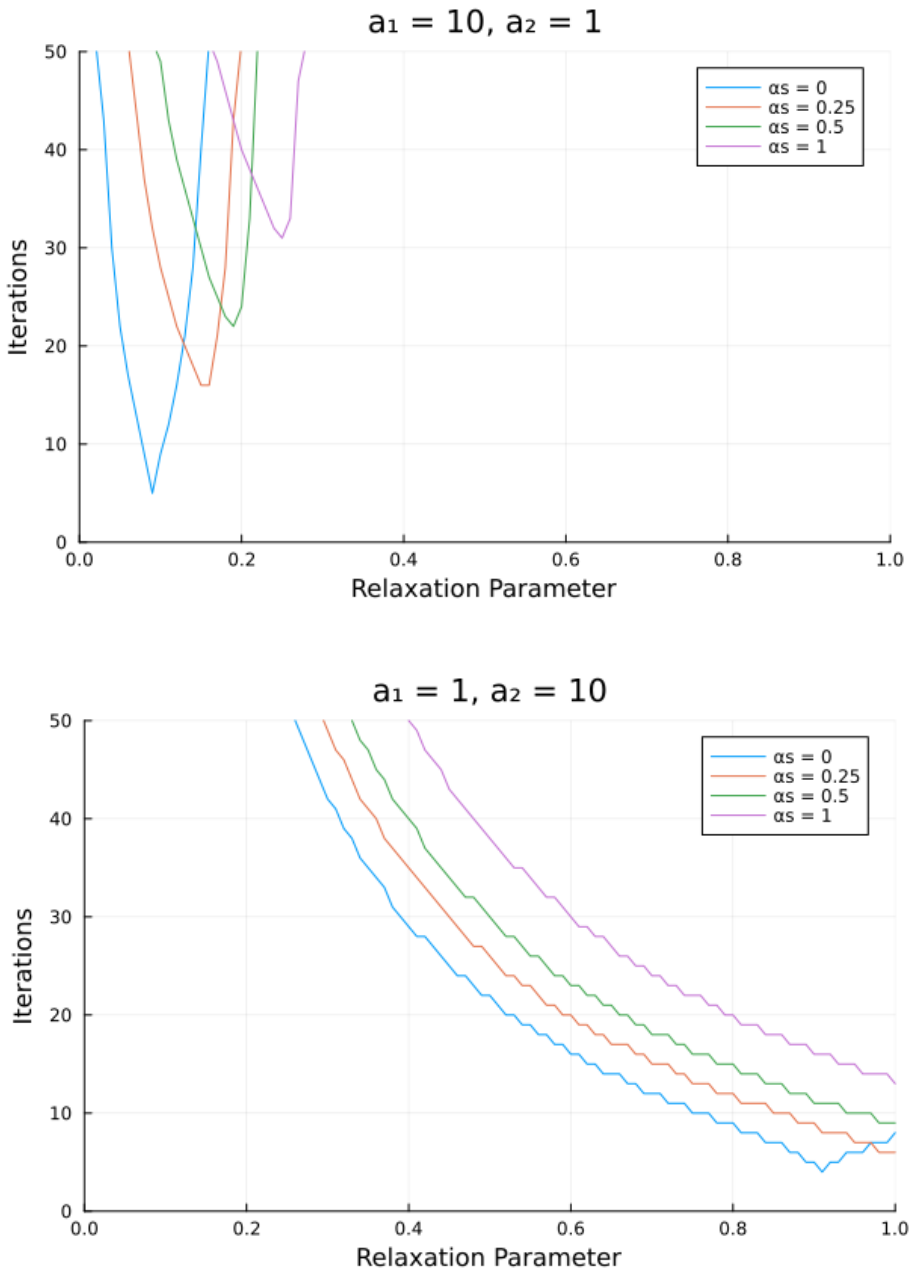



FIGURE 11: Numerical Experiment-3 — Dirichlet-Robin iteration — Coupled Poisson Problem

From the above plots, we can observe that the convergence plot tends to shift to the right with lesser optimal convergence as $\alpha_s$ increases. This is true for both the case when $(a_1 > a_2)$ and when $(a_2 > a_1)$.

For the fourth numerical experiment conducted on the Dirichlet-Robin method, we change the location of the interface to the two extremes and then we vary the diffusion coefficients to understand how that affects the convergence criteria of the Dirichlet-Robin method. The below plot shows us how the relaxation parameter affects the iteration count.

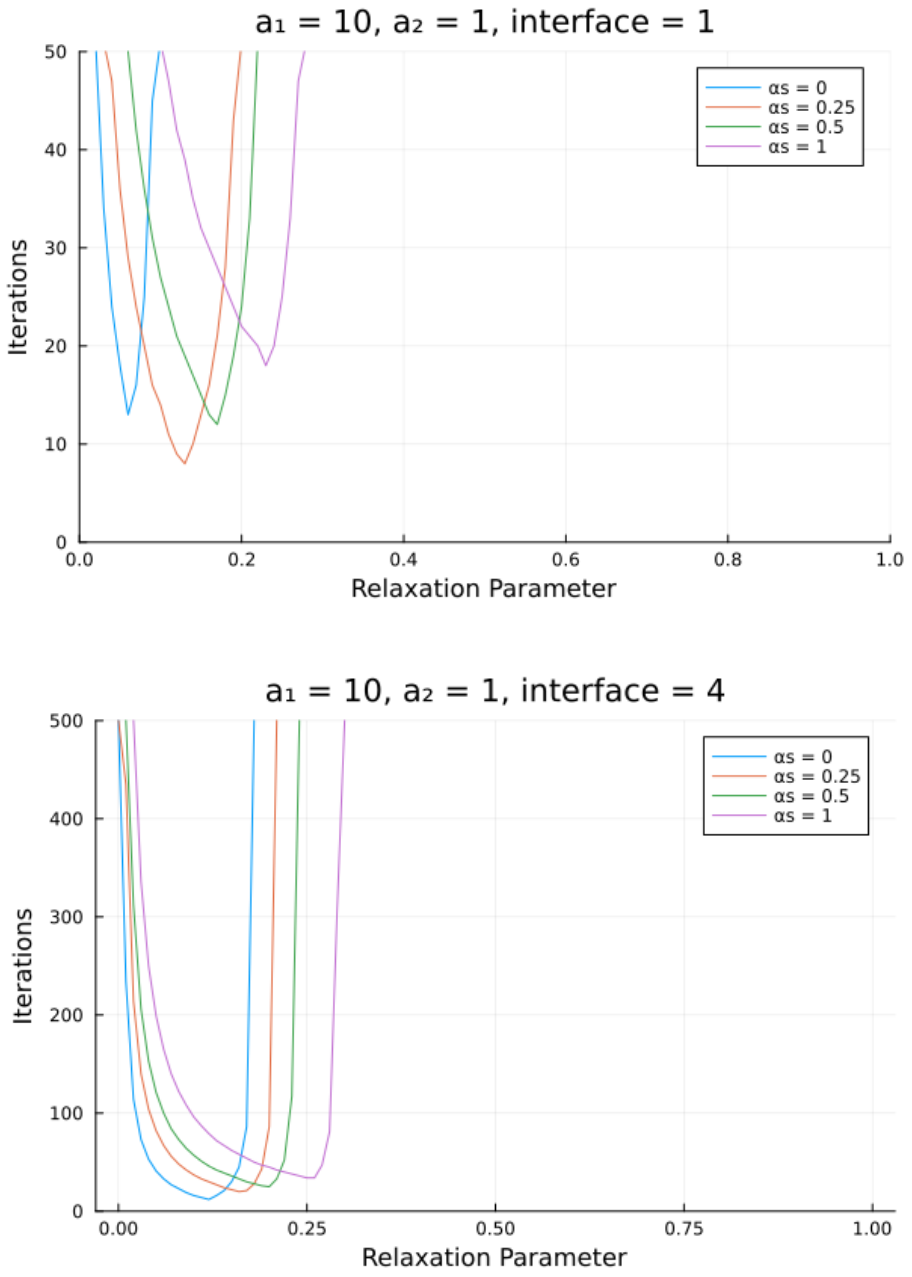



FIGURE 12: Numerical Experiment-4 (a) — Dirichlet-Robin iteration — Coupled Poisson Problem

From the above plots ($a_1 > a_2$), we can make an interesting observation where the Dirichlet-Robin method tends to be superior to the Dirichlet-Neumann method, if $\alpha_s$ is chosen correctly. We can also see the generic behavior of this method where the convergence plot is moved to the right as $\alpha_s$ increases.

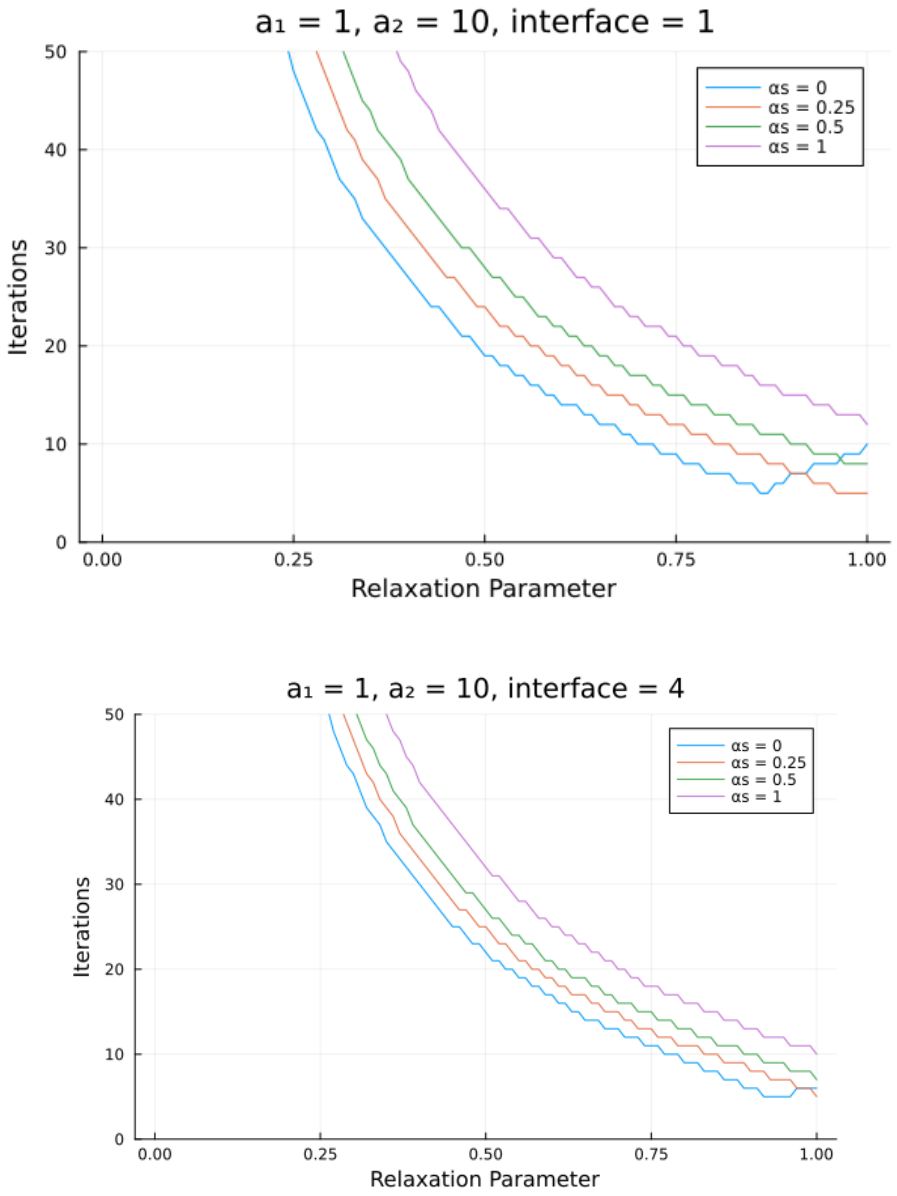



FIGURE 13: Numerical Experiment-4 (b) — Dirichlet-Robin iteration — Coupled Poisson Problem

For the case where $a_2 > a_1$, the convergence of the Dirichlet-Robin method tends to be similar to the Dirichlet-Neumann method.

### A.1.4 Robin-Robin preconditioner

In this section, we discuss the convergence results for the problem when we apply the Robin-Robin preconditioner. We take the special case when $(\alpha_f = \alpha_s)$ for the analysis.
For the first numerical experiment, we try to compare how varying the Robin parameter affects the convergence of the Robin-Robin preconditioner for the standard problem. Here the interface

is kept at the center and the diffusion coefficients are kept to be one. The below plot shows us how the relaxation parameter affects the iteration count.

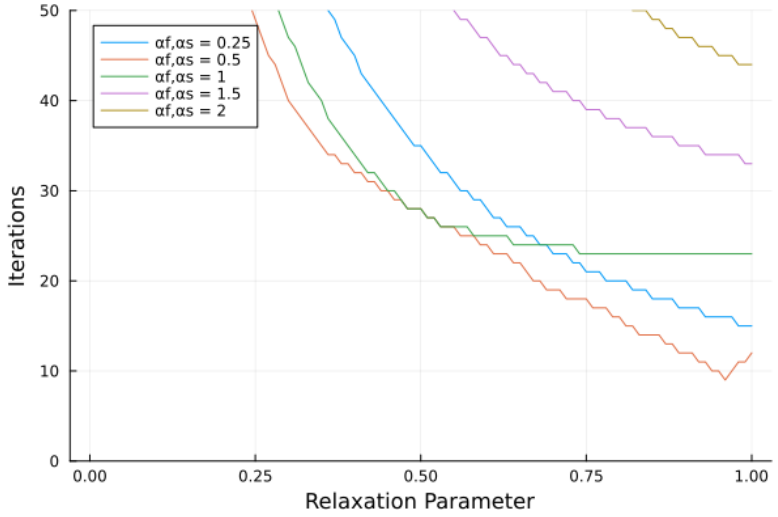
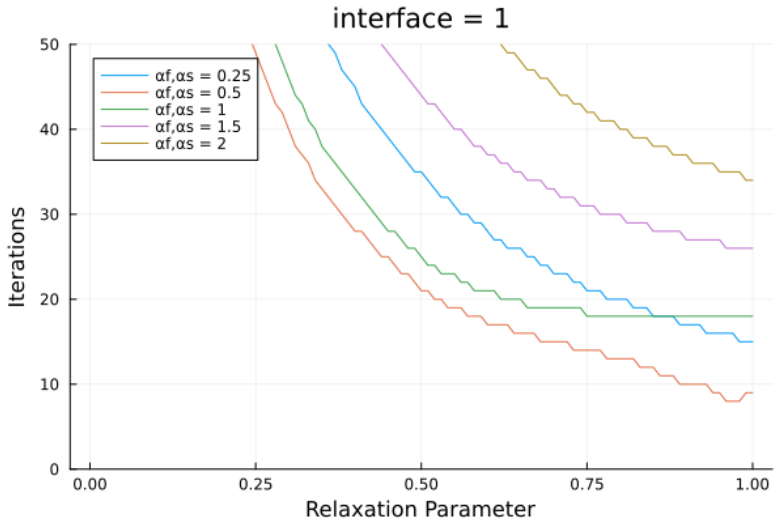



FIGURE 14: Numerical Experiment-1 — Robin-Robin iteration — Coupled Poisson Problem

From the above plot, we can observe that for all cases, the most optimal convergence occurs for the system that has no relaxation. The most optimum Robin parameter turns out to be 0.5 for this case when both the diffusion coefficients are equal.

For the second numerical experiment conducted on the Robin-Robin method, we keep the diffusion coefficient to be one and we vary the interface position to understand how it impacts the preconditioner. The below plot shows us how the relaxation parameter affects the iteration count.



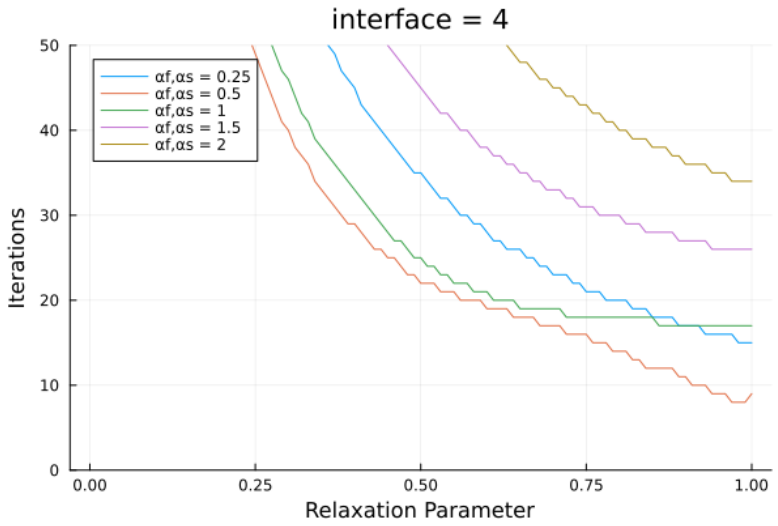
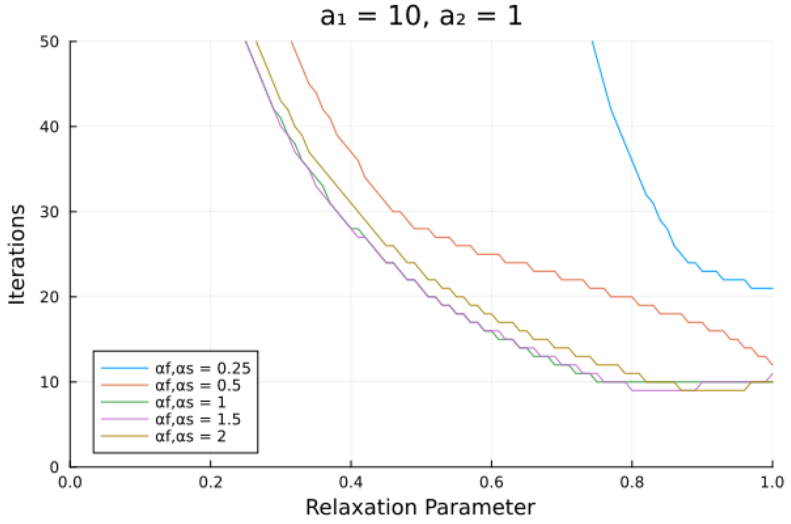



FIGURE 15: Numerical Experiment-2 — Robin-Robin iteration — Coupled Poisson Problem

We can observe that, similar to the last experiment, for each case, the most optimal convergence occurs for the system that has no relaxation. Also, we can observe that $\alpha = 0.5$ is the optimum relaxation parameter, same as the last numerical experiment. This indicates that the Robin parameter is not dictated by the interface position.

For the third numerical experiment conducted on the Robin-Robin method, we keep the interface in the center and we vary the diffusion coefficient to understand how that affects the convergence criteria of the Robin-Robin method. The below plot shows us how the relaxation parameter affects the iteration count.



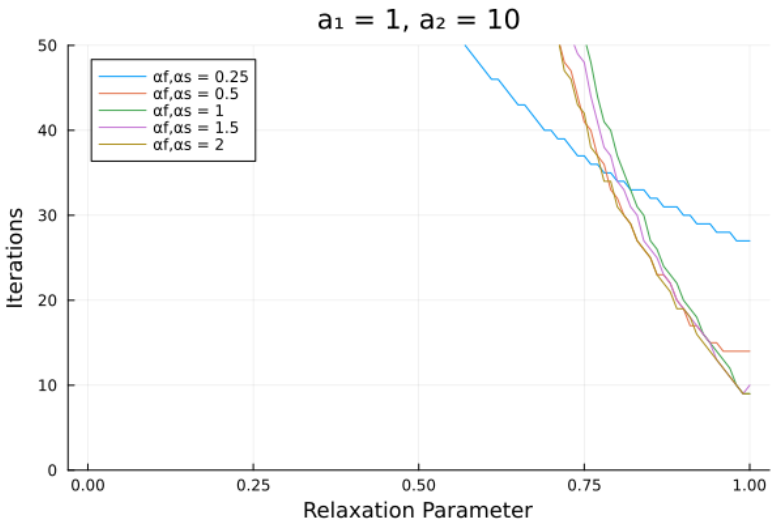
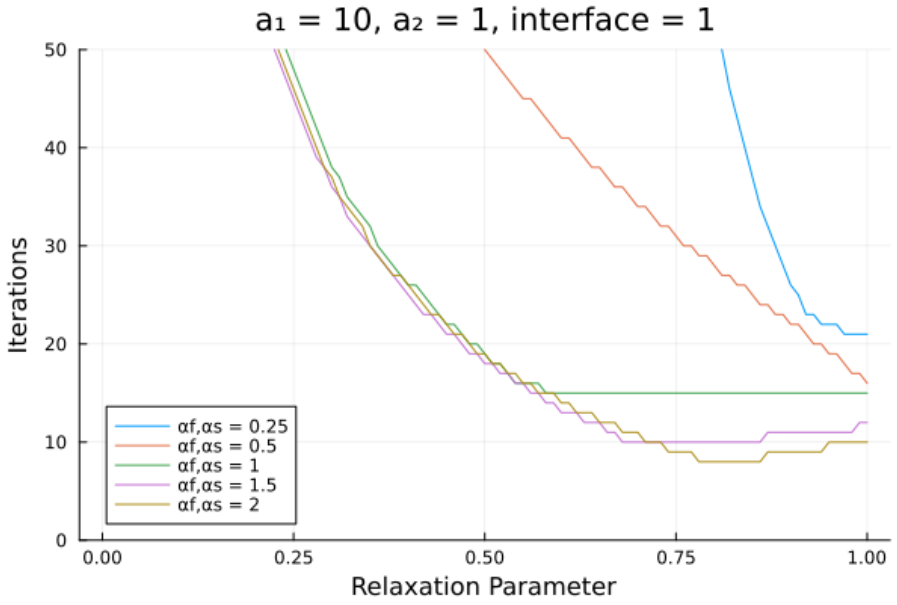



FIGURE 16: Numerical Experiment-3 — Robin-Robin iteration — Coupled Poisson Problem

We can observe that for all cases, the most optimal convergence occurs for the system with no relaxation. Another point of observation is that for almost all cases where $\alpha > 0.5$, the convergence remains the same.

For the fourth numerical experiment conducted, on the Robin-Robin method, we change the location of the interface of the two extremes and then we vary the diffusion coefficients to understand how that affects the convergence criteria of the Robin-Robin method. The below plot shows us how the relaxation parameter affects the iteration count.



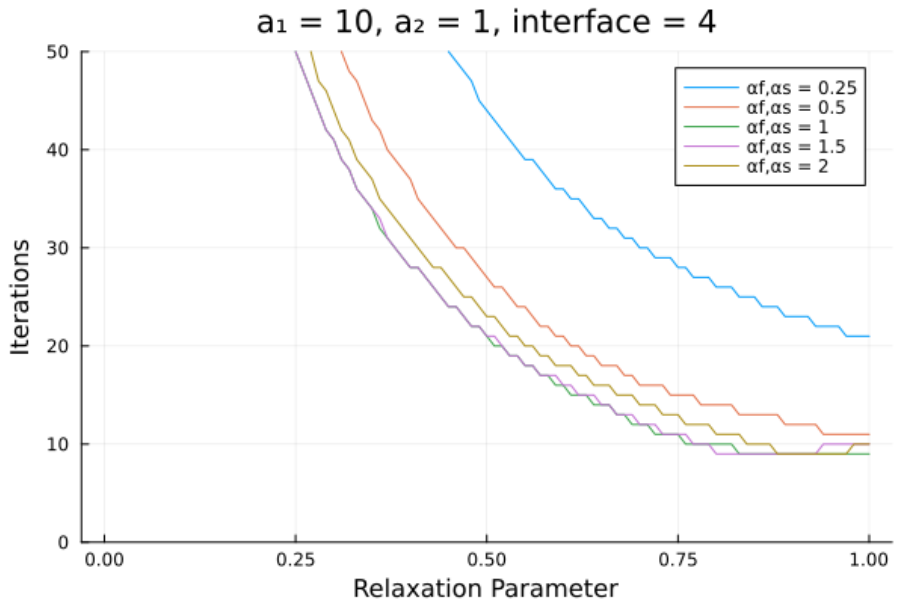
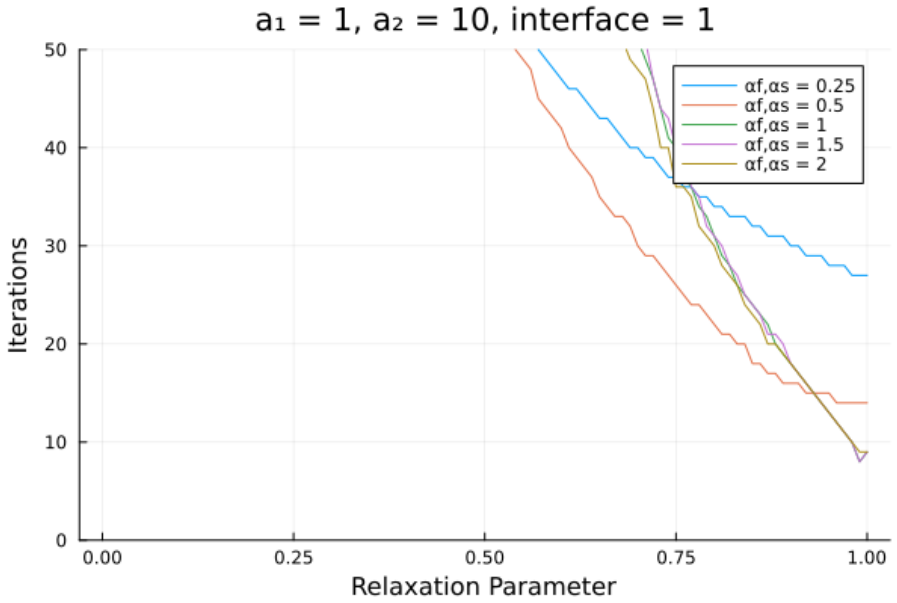



FIGURE 17: Numerical Experiment-4 (a) — Robin-Robin iteration — Coupled Poisson Problem

From the above plots ($a_1 > a_2$), we can make an interesting observation that for all cases, they tend to have similar convergence plots irrespective of the interface position. It is also important to note that the best convergence approximately occurs for the case with no relaxation.



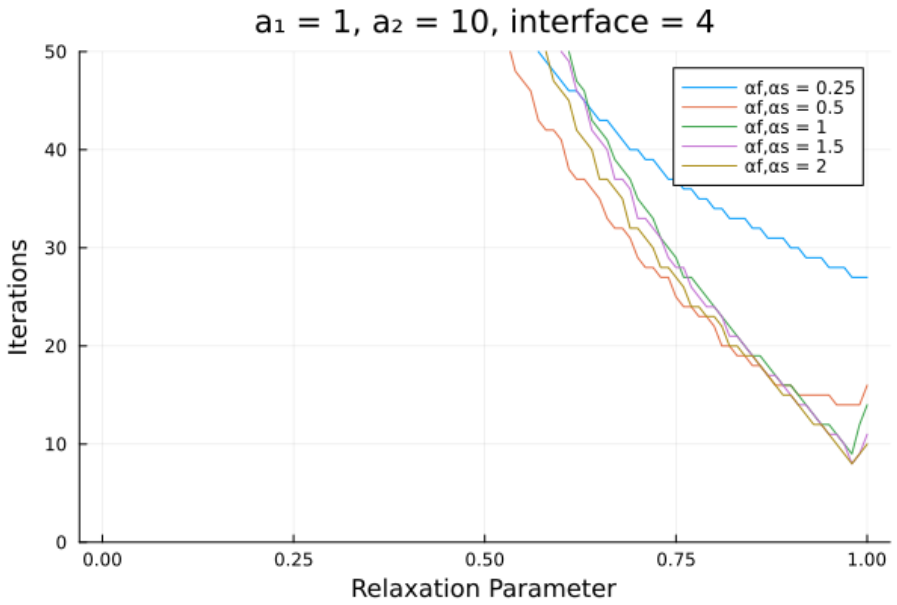



FIGURE 18: Numerical Experiment-4 (b) — Robin-Robin iteration — Coupled Poisson Problem

For the case where $a_2 > a_1$, the convergence of the Robin-Robin method tends to be similar to the case when $a_1 > a_2$. It is also important to note that convergence is worse when the relaxation parameter is less than 0.5 compared to the case $a_1 > a_2$.

# B Specific code implementation

## B.1 DOF extraction function

```
function get_node_dof_ids(reffe::ReferenceFE{}, gmsh_tag::Vector{String}, V::FESpace;
    ↪ vec_comp::Int = 1)
    # Get cell dofs
    cell_dofs = Gridap.FESpaces.get_cell_dof_ids(V)
    # Get the triangulation
    trian = Gridap.FESpaces.get_triangulation(V)
    # Get the Discrete Model from the triangulation
    discrete_model = Gridap.Geometry.get_active_model(trian)
    # Get dimensions
    D = Gridap.Geometry.num_cell_dims(trian)
    # Get the Grid Topology
    grid_topology = Gridap.Geometry.get_grid_topology(discrete_model)
    # Get the cell to d_to_dface mapping along with the offset
    d_to_offset = Gridap.Geometry.get_offsets(grid_topology)
    cell_to_faces = Gridap.Geometry.get_cell_faces(grid_topology)
    # Exit the function if the size of cell_to_faces is zero or the triangulation is
    ↪ not there in the local subdomain
    if (size(cell_to_faces)[1] == 0)
        return [],[]
    end
    # Get the facelabeling of the model
```

```
facelabeling = Gridap.Geometry.get_face_labeling(discrete_model)
# Get the bool mask vectors for the tag given
d_to_dface_to_tag = [ Gridap.Geometry.get_face_tag_index(facelabeling,gmsh_tag,d)
↪ for d in 0:D]
# Get the number of cells
numcells = size(cell_to_faces)[1]
@assert size(cell_to_faces)[1] == size(cell_dofs)[1] "Inconsistant number of cells;
↪ quitting program"
numfaces = size(cell_to_faces[1])[1]
# Understand the dof arrangement in reffe
dof_to_node = Gridap.ReferenceFEs.get_dof_to_node(reffe)
numnodes = maximum(dof_to_node)
numelem = size(dof_to_node)[1]
noderept = numelem/numnodes # To know number of components of the vector
@assert mod(numelem,numnodes) == 0 "Inconsistent number of components in the vector"
# Create a cellfield of bools to know if a element has a tag
bool_cell_field = SparseArrays.spzeros(numcells,numnodes)
# Looping over all cells
for cell = 1:numcells
    for node = 1:numfaces
        node_id = cell_to_faces[cell][node]
        for j = 1:size(d_to_offset)[1]
            if (node_id > d_to_offset[j] && j == size(d_to_offset)[1]) # Last face
↪ node check
                node_id = node_id - d_to_offset[j] # offset node_id
                bool_cell_field[cell,node] = d_to_dface_to_tag[j][node_id];
↪ continue;
            end
            if (node_id > d_to_offset[j] && node_id <= d_to_offset[j+1])
                node_id = node_id - d_to_offset[j] # offset node_id
                bool_cell_field[cell,node] = d_to_dface_to_tag[j][node_id];
↪ continue;
            end
        end
    end
end
# New Comparing the bool matrix to the cell dof matrix and add the intface DOFs
face_own_nodes = vcat([Gridap.ReferenceFEs.get_face_own_nodes(reffe,d) for d in
↪ 0:D]...)
@assert size(face_own_nodes)[1] == numfaces "Inconsistent number of faces"
VGamma = Int[]
VGamma_dface = Int[]
for cell = 1:numcells
    for node = 1:numfaces
        if(bool_cell_field[cell,node] == 1)
            dof_node = face_own_nodes[node] .+ numnodes*(vec_comp - 1)
            dface = cell_to_faces[cell][node].*(ones(size(dof_node)[1]))
            loc_dof = cell_dofs[cell][dof_node]
            append!(VGamma,loc_dof)
            append!(VGamma_dface,dface)
        end
    end
end
indices = [findfirst(==(v), VGamma) for v in sort(unique(VGamma))]
VGamma = VGamma[indices]
```

```
    VGamma_dface = VGamma_dface[indices]
    return VGamma, VGamma_dface
end
```

## B.2 Matrix modification wrapper (FSI problem)

In this section, we explain the process of matrix modification required to couple the fluid and solid domains.

Let us consider, we are trying to strongly impose the Dirichlet condition between the fluid velocity and the solid displacement ($u = \dot{d}$). This would be equivalent to the following matrix equation.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u} \\ \ddot{d} \end{bmatrix} + \begin{bmatrix} 0 & -I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{d} \end{bmatrix} + \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ d \end{bmatrix} = 0$$

We use the generalized alpha method [14] to discretize and solve this ordinary differential equation. We can rewrite the equation as the following with $a, b, c$ as constant floating point numbers.

$$a \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u}^{n+1} \\ \ddot{d}^{n+1} \end{bmatrix} + b \begin{bmatrix} 0 & -I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u}^{n+1} \\ \ddot{d}^{n+1} \end{bmatrix} + c \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u}^{n+1} \\ \ddot{d}^{n+1} \end{bmatrix} = \begin{bmatrix} r_1(\ddot{u}^n, \dot{u}^n, u^n) - r_2(\ddot{d}^n, \dot{d}^n, d^n) \\ 0 \end{bmatrix}$$

In Gridap, the data $a, b, c$ and $r_1(\ddot{u}^n, \dot{u}^n, u^n), r_2(\ddot{d}^n, \dot{d}^n, d^n)$ are readily available in the linear stage operator struct as $ws$ (tuple) and $usx$ (tuple) respectively and are used in the matrix modification process. The final modified matrix structure would be the following.

$$\begin{bmatrix} c & -b \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u}^{n+1} \\ \ddot{d}^{n+1} \end{bmatrix} = \begin{bmatrix} r_1(\ddot{u}^n, \dot{u}^n, u^n) - r_2(\ddot{d}^n, \dot{d}^n, d^n) \\ 0 \end{bmatrix}$$

Imposition of the Neumann interface condition weakly is straightforward as we are equating the residuals of the fluid momentum equation with the solid equation along the trace of the interface. Consider the uncoupled discrete problem.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & 0 \\ 0 & 0 & 0 & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \ddot{u}_i \\ \ddot{p} \\ \ddot{u_\Gamma} \\ \ddot{d} \end{bmatrix} = \begin{bmatrix} rf_I \\ rf_p \\ rf_\Gamma \\ rs \end{bmatrix}$$

The coupled discrete problem after performing the matrix modifications would be the following.

$$\begin{bmatrix} C_{II} & G_I & C_{I\Gamma} & 0 \\ D_I & 0 & D_\Gamma & 0 \\ 0 & 0 & cI & -bI \\ C_{\Gamma I} & G_\Gamma & C_{\Gamma\Gamma} & N_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \ddot{u}_i \\ \ddot{p} \\ \ddot{u_\Gamma} \\ \ddot{d} \end{bmatrix} = \begin{bmatrix} r_f \\ r_p \\ r_1(\ddot{u}^n, \dot{u}^n, u^n) - r_2(\ddot{d}^n, \dot{d}^n, d^n) \\ r_{f\Gamma} + r_s \end{bmatrix}$$

A new solver of the type linear solver is written to handle this matrix modification in Gridap. The structure of the solver is the following. The DOF data is stored as a dictionary.

```
struct MatrixModifier <: LinearSolver
    params::Dict
    ls::LinearSolver
end
```

The matrix (left hand side) modification happens in the numerical setup function of the Matrix Modifier solver. The vector (right hand side) modifications occur in the solve! function called by the ODE solver.

## B.3 Gridap Trilinos Interface

In this section, we provide the code to the C++ Gridap-Trilinos executable.

```
void TrilinosParallel(jlcxx::ArrayRef<double> A_nzval, jlcxx::ArrayRef<int64_t>
    ↪ A_rowval,
  jlcxx::ArrayRef<int64_t> A_colptr, jlcxx::ArrayRef<double> RhsVec,
    ↪ jlcxx::ArrayRef<double> LocSoln, jlcxx::ArrayRef<int64_t> RowPartition,
  jlcxx::ArrayRef<int64_t> ColPartition, int64_t LinSysSize, int64_t LocRowSize,
    ↪ int64_t LocColSize,
  jlcxx::ArrayRef<int64_t> OwnToValSol, jlcxx::ArrayRef<int64_t> OwnToValRow) {
  // Setting the Global system size
  Tpetra::global_size_t numGblIndices = LinSysSize;
  MPI_Comm yourComm = MPI_COMM_WORLD;
  {
    // Passing MPI comm to Trilinos
    RCP<const Comm<int> > comm (new MpiComm<int> (yourComm));
    const int myRank = comm->getRank ();
    const int numProcs = comm->getSize ();
    // Initialise FancyOStream
    RCP<Teuchos::FancyOStream> out =
    ↪ Teuchos::fancyOStream(Teuchos::rcpFromRef(std::cout));
    const bool verbose = (myRank == 0); // Only print on rank 0
    //// Begin custom Tpetra map
    Teuchos::ArrayView<const global_ordinal_type> rowIndices(
      reinterpret_cast<const global_ordinal_type*>(RowPartition.data()),
      RowPartition.size());
    Teuchos::ArrayView<const global_ordinal_type> colIndices(
      reinterpret_cast<const global_ordinal_type*>(ColPartition.data()),
      ColPartition.size());
    // Setting the IndexBase
```

```
const global_ordinal_type indexBase = 0;
// Row Map Construction
RCP<const Tpetra_map> rowMap =
        rcp(new Tpetra_map(numGblIndices, rowIndices, indexBase, comm));
// Col Map Construction
RCP<const Tpetra_map> colMap =
        rcp(new Tpetra_map(numGblIndices, colIndices, indexBase, comm));
//// End custom Tpetra map
//// Begin Tpetra Matrix Assembly
const size_t maxNumEntriesPerRow = 100;
RCP<crs_matrix_type> A = rcp(new crs_matrix_type (rowMap,colMap,
↪ maxNumEntriesPerRow)); // Initialization
// CSC Matrix Insertion using LocalInsert
for(int64_t col = 0; col < LocColSize; ++col){
  int64_t start = A_colptr[col];
  int64_t end = A_colptr[col + 1];
  for(int64_t j = start; j<end;j++){
    double value = A_nzval[j];
    local_ordinal_type row = static_cast<local_ordinal_type>(A_rowval[j]);
    local_ordinal_type Col = static_cast<local_ordinal_type>(col);
    if(value == 0.0){continue;}
    A->insertLocalValues(row,Teuchos::ArrayView<const
↪ local_ordinal_type>(&Col,1),Teuchos::ArrayView<const double>(&value,1));
  }
}
A->fillComplete();
size_t locSize = A->getLocalNumRows();
//// End Tpetra Matrix assembly
///// Begin Initializing Solution Vector
RCP<vec_type> x(new vec_type(A->getDomainMap()));
↪ x->putScalar(Teuchos::ScalarTraits<scalar_type>::zero());
///// End Initializing Solution Vector
//// Begin Right Hand side vector
RCP<Tpetra::Vector<>> b = rcp(new Tpetra::Vector<>(rowMap));
for(int64_t i = 0; i < LocRowSize; ++i){
  local_ordinal_type row = static_cast<local_ordinal_type>(i);
  double value = RhsVec[OwnToValRow[i]];
  b->sumIntoLocalValue(row, value);
}
//// End Right Hand side vector
//// Begin converting Tpetra objects to Xpetra objects
RCP<Xpetra::Matrix<scalar_type, local_ordinal_type, global_ordinal_type, NO>>
↪ xpetra_A = MueLu::TpetraCrs_To_XpetraMatrix<scalar_type, local_ordinal_type,
↪ global_ordinal_type, NO>(A);
RCP<Xpetra::Vector<scalar_type, local_ordinal_type, global_ordinal_type, NO>>
↪ xpetra_b = Xpetra::toXpetra(b);
RCP<Xpetra::Vector<scalar_type, local_ordinal_type, global_ordinal_type, NO>>
↪ xpetra_x = Xpetra::toXpetra(x);
//// End converting Tpetra objects to Xpetra objects
//// Begin Reading parameters list from input XML file
RCP<ParameterList> parameterList =
↪ getParametersFromXmlFile("src/parameters-2d.xml");
RCP<ParameterList> belosList = sublist(parameterList,"Belos List");
RCP<ParameterList> precList = sublist(parameterList,"Preconditioner List");
//// End Reading parameters list from input XML file
```

```
    //// Begin constructing the preconditioner
    RCP<onelevelpreconditioner_type> prec(new
    ↪ onelevelpreconditioner_type(xpetra_A,precList));
    prec->initialize(false);
    prec->compute();
    RCP<operatort_type> belosPrec = rcp(new xpetraop_type(prec));
    //// End constructing the preconditioner
    //// Begin solving the system using Belos
    // Constructing the linear problem
    RCP<operatort_type> belosA = rcp(new xpetraop_type(xpetra_A));
    RCP<linear_problem_type> linear_problem (new
    ↪ linear_problem_type(belosA,xpetra_x,xpetra_b));
    linear_problem->setProblem(xpetra_x,xpetra_b);
    if(locSize != 0){
      linear_problem->setRightPrec(belosPrec); // Specify the preconditioner
    }
    // Sending the parameters to the solver
    solverfactory_type solverfactory;
    RCP<solver_type> solver = solverfactory.create(parameterList->get("Belos Solver
    ↪ Type","GMRES"),belosList);
    solver->setProblem(linear_problem);
    // Solve
    Belos::ReturnType ret = solver->solve();
    bool success = false; success = (ret==Belos::Converged);
    if (success) {
      if (verbose)
        std::cout << "\nEnd Result: Problem Solved!" << std::endl;
    } else {
      if (verbose)
        std::cout << "\nEnd Result: Error in solving" << std::endl;
    }
    //// End solving the system using Belos
    //// Begin copying the solution
    auto x_data_host = x->getLocalViewHost(Tpetra::Access::ReadOnly);
    for(size_t i = 0; i< LocRowSize; ++i){
      LocSoln[OwnToValSol[i]] = x_data_host(i, 0);
    }
    //// End copying the solution
  }
  return;
}

void KokkosInitialize(){
  Kokkos::initialize();
  return;
}

void KokkosFinalize(){
  Kokkos::finalize();
  return;
}

JLCXX_MODULE define_julia_module(jlcxx::Module& mod) {
    mod.method("TrilinosParallel", &TrilinosParallel);
    mod.method("KokkosInitialize", &KokkosInitialize);
```

```
    mod.method("KokkosFinalize", &KokkosFinalize);
}
```

### B.4 Sample input xml file

```
<ParameterList name="Sample Input File">
    <Parameter name="Belos Solver Type"                    type="string" value="GMRES"/>
    <ParameterList name="Belos List">
        <Parameter name="Block Size"                       type="int"    value="1"/>
        <Parameter name="Convergence Tolerance"            type="double" value="1e-8"/>
        <Parameter name="Maximum Iterations"               type="int"    value="1000"/>
        <Parameter name="Verbosity"                        type="int"    value="33" />
        <Parameter name="Output Style"                     type="int"    value="1" />
        <Parameter name="Output Frequency"                 type="int"    value="1"/>
        <Parameter name="Explicit Residual Test"           type="bool"   value="false"/>
    </ParameterList>
    <ParameterList name="Preconditioner List">
        <Parameter name="OverlappingOperator Type"         type="string"
    ↪ value="AlgebraicOverlappingOperator"/>
        <Parameter name="Dimension"                        type="int"    value="2"/>
        <Parameter name="Overlap"                          type="int"    value="1"/>
        <ParameterList name="AlgebraicOverlappingOperator">
            <Parameter name="Combine Values in Overlap"    type="string" value="Full"/>
            <ParameterList name="Solver">
                <Parameter name="SolverType"               type="string"
    ↪ value="Amesos2"/>
                <Parameter name="Solver"                   type="string"
    ↪ value="Umfpack"/>
            </ParameterList>
        </ParameterList>
    </ParameterList>
</ParameterList>
```

## C Software versions used

- Julia - v1.10.5
- Gridap.jl - v0.18.12
- GridapDistributed.jl - v0.4.7 with PR 174
- GridapGmsh.jl - v0.7.2
- GridapSolvers.jl - v0.5.0
- PartitionedArrays.jl - v0.3.4
- LinearAlgebra.jl

- SparseArrays.jl - v1.10.0
- BlockArrays.jl - v1.4.0
- Plots.jl - v1.40.13
- Trilinos - 16.1.0
- SuiteSparse - 7.8.2