Analysis and Massively Parallel Implementation of the 2-Lagrange Multiplier Methods and Optimized Schwarz Methods

Anastasios Karangelis

Submitted for the degree of
Doctor of Philosophy

Heriot-Watt University

Department of Mathematics,
School of Mathematical and Computer Sciences.

April 5, 2016

Abstract

A partial differential equation (PDE) is an equation that involves partial derivatives of an unknown function $u : \Omega \to \mathbb{R}$. The domain Ω denotes an open subset of R^d, $d = 2, 3$. Partial differential equations (PDEs) are used in order to model problems in all areas of science, including physics, engineering, finance, etc...

It is often impossible to solve a PDE exactly using analytical methods and thus most PDEs are solved numerically. In order to solve a PDE numerically, one introduces a triangulation of the domain with finitely many vertices. The solution of the PDE is approximated by a piecewise polynomial function on this grid; this is the Finite Element Method (FEM).

For linear PDEs, the FEM leads to a linear system of equations of the form $Au = f$, where A is a very large $n \times n$ matrix that is very sparse. Solving $Au = f$ using Gaussian elimination results in significant "fill-in", where many zero entries of A become nonzero as the Gaussian elimination algorithm progresses. For very large problems, Gaussian elimination will exhaust the main memory of any computer. As a result, it is necessary to use an iterative algorithm to solve the problem $Au = f$.

Classical iterative methods such as Jacobi and Gauss-Seidel require more and more iterations as the size n of A increases and thus these iterations are not useful for large values of n. Domain decomposition is a more sophisticated iterative scheme based on partitioning the domain Ω into many subdomains $\{\Omega_k\}$. Since the Green's functions are nonlocal, it is impossible to solve local problems on the subdomains Ω_k and thus obtain a global solution unless we iterate by exchanging information between the local problems.

In the method of H. Schwarz, the information exchanged across the "artificial interfaces" is Dirichlet data. J. L. Lions improved the Schwarz method by exchanging Robin boundary data. Gander and his collaborators found that one could obtain accelerated convergence by tuning the Robin parameter; this is the optimized Schwarz method (OSM). The design and analysis of optimized Schwarz methods for general domains and subdomains has proven to be a challenge. Loisel found that the OSM is dual to the 2-Lagrange multiplier (2LM) method, which is an iteration on the Robin traces; the 2-Lagrange multiplier method is amenable to analysis.

OSM with a coarse grid correction has previously been considered for cylindrical domains using Fourier analysis. This approach has some serious limitations, since only very regular domains and subdomains and only the Laplacian can be considered. These limitations are not purely theoretical; the implementation of an OSM for a domain decomposition with cross points (points that are shared between three or more subdomains) has remained challenging. Our main result is the design and implementation of a 2-Lagrange multiplier method, which is dual to the OSM, including a coarse-grid correction, which handles cross points. This is highly valuable

not only from the point of view of analysis, but also because it describes in detail how the cross points are handled by the implementation. Our analysis shows that the condition number of the linear problem scales like $O((H/h)^{1/2})$, where H is the subdomain diameter and h is the grid parameter.

We have implemented our algorithms in C using the PETSc library. Numerical experiments performed on the HECToR supercomputer confirm the good scaling properties of our algorithms.

For Alexandros, Angeliki, Christos.

The most formidable weapon
against errors of every kind is
reason. I have never used any other,
and I trust I never shall.

Thomas Paine, January 27 1974.

Acknowledgements

I would like to gratefully and sincerely thank my supervisor Dr. Sébastien Loisel, who supported me and also gave me complete freedom in my research. I feel very privileged to have received the opportunity to study towards a Ph.D. These last four years, I had the chance to be exposed to many different areas of Applied Mathematics and to meet interesting and enlightened people. I want to thank the Centre of Numerical Analysis and Intelligent Software, NAIS, which provided me with all the necessary training and facilities in order to complete my research and experiments. NAIS was funded by EPSRC grant EP/G036136/1.

In the Greek mythology, Sisyphus was punished to roll a huge stone up a hill, only to see it roll down again, a procedure that would repeat forever. This is how the procedure of studying towards a Ph.D feels sometimes. At these moments you need people to support and encourage you. I would like to specially thank, Nicolas Vardakis, Ruairi Donnelly, Christina Latroni, and Evgeny Vylegzhanin, for being there when I needed them. I would also like to thank my officemates who were always kind and helpful.

I would like to thank Prof. Gabriel Lord, whose passion and enthusiasm on the fields of Applied Mathematics and Numerical Analysis influenced me greatly. Finally, I should also thank the internal examiner of my Ph.D yearly reports Dr. Lyonell Boulton, who always provided me with useful and insightful feedback.

Contents

1 Introduction **10**

1.1 Introduction 10

1.1.1 Summary of main results 15

1.1.2 Overview of the thesis 17

2 Sobolev Spaces and the Variational Formulation of Elliptic PDEs **18**

2.1 Sobolev Spaces 18

2.2 Variational Formulation of Elliptic Boundary Value Problems 23

3 Finite Element Method **29**

3.1 Galerkin Method 29

3.2 Finite Element Method 31

3.2.1 Assembly of the Galerkin Systems 37

4 Krylov Subspace Methods **46**

4.1 Iterative methods and Projection methods 46

4.1.1 Projection Methods 50

4.2 The Generalized Minimal residual method 51

4.3 Convergence of GMRES 55

4.3.1 Convergence of GMRES, connection with potential theory . . 58

4.3.2 Convergence for Non HPD Matrices 62

5 2-Lagrange Multiplier methods-Optimized Schwarz methods **70**

5.1 The 2-Lagrange Multiplier methods 70

5.1.1 Obtaining the S2LM system from the global system (5.2) . . . 72

5.2 Connections of 2-Lagrange Multiplier methods and the Optimized Schwarz method 80

5.3 Algebraic form of OSM 86

5.4 Equivalence between 2LM and OSM. 88

5.5 Weak scalability of the 2-Level 2-Lagrange multiplier methods 92

5.5.1 Schur complement properties 92

5.5.2 Methods that scale weakly 97

5.5.3 The condition number of A_{2LS2LM} . . . 106
5.5.4 The condition number of A_{2L2LM} . . . 112
5.6 Motivation for the non symmetric System . . . 113
5.7 Numerical experiments . . . 115
5.7.1 Algebraic case . . . 115
5.7.2 Elliptic case . . . 116
5.7.3 Performance with GMRES and GMRES(10) . . . 117

6 Massively parallel Implementations and Experiments 120
6.1 First Implementation and Experiments . . . 120
6.2 Second Implementation: A Massively parallel implementation "black-box" solver . . . 124
6.2.1 Mesh Generation and assembly of local Neumann problems . . 126
6.2.2 The 2LS2LM and 2L2LM "black-box" solver . . . 126
6.2.3 Large scale experiments on HECToR supercomputer . . . 128
6.2.4 More details on the parallel implementation . . . 129
6.2.5 Comments on Scalability for the Massively Parallel Experiments 134

7 Conclusion and Future work 137

Appendices 139

A Notes on Massively Parallel Implementation 140
A.1 `MATIS` "black box" solver . . . 141
A.2 The deal.II Implementation . . . 147
A.3 The femH parallel C++ FEM library . . . 151

B Proof of Theorem 5.5.22 155

List of Figures

1.1 Images of HECToR supercomputer and its successor ARCHER based at EPCC. . . . 12
1.2 Projected performance development of top 500 Supercomputers, [http://top500.org/statistics/perfdevel/, data: June 2015]. . . . 12

2.1 Function (2.1) in one and two dimentions . . . 19

3.1 Triangulation of a polygonal mesh Ω, partitioned in 8 non-overlapping subdomains by METIS. [43] . . . 32
3.2 Square 2D mesh. . . . 33
3.3 Histograph of Mesh Quality for mesh of Fig. 3.1, produced by using Algorithm 1. . . . 34
3.4 Hat functions in 1-dimension . . . 35
3.5 Piecewise linear Hat function 2D . . . 36

4.1 Rectangular domain with a circular hole used for the experiments in Figure 4.2. . . . 55
4.2 Solving the Poisson problem in a rectangular domain with a circular hole, by using the PETSc library. . . . 56
4.3 A compact set $S \subset \mathbb{C}$ that approximates the spectrum of $\sigma(A)$, $A \in \mathbb{R}^{100\times100}$. . . . 58
4.4 GMRES for the Possion Problem on uniform square. . . . 64
4.5 Convergence of GMRES for indefinite matrices . . . 64
4.6 Convergence of GMRES for indefinite matrices . . . 65
4.7 GMRES Disk eigenvalues . . . 67
4.8 GMRES Ellipse Eigenvalues . . . 67
4.9 GMRES Ellipse Eigenvalues . . . 68

5.1 Domain Ω divided in three non-overlaping subdomains . . . 71
5.2 Divive Ω in two overlaping subdomains . . . 81
5.3 Floating subdomains . . . 94
5.4 Loglog plot of the condition number $\kappa_0(S)$ . . . 96

5.5 Loglog plot of condition numbers of Q, $a_{opt} = \sqrt{h/H}$ 104
5.6 Loglog plot of eigenvalues of Q, $Q - K$ 104
5.7 Functions $x(\epsilon)$, $y(\epsilon)$ of (5.132), $\epsilon \in (0, 0.5]$. 109
5.8 Functions ϕ_+, ϕ_-, defined in (5.135) 110
5.9 Right-hand-sides of (5.133) (lightly shaded area) and (5.138) (dark areas, including the dark curve $\beta(\epsilon)$). 111
5.10 Spectrum and pseudospectrum of A_{2L2LM}. 114
5.11 Condition numbers of A_{2LS2LM} for random choices of Q and K for various values (m, n, k) (dots) compared to (5.123) (solid curve). Top-left: $n = 4, k = 3, m = 2$; top-right: $n = 8, k = 4, m = 4$; bottom-left: $n = 15, k = 8, m = 7$; bottom-right: $n = 30, k = 18, m = 15$. 115
5.12 Convergence of the relative residual norm in the GMRES (top) and GMRES(10) (bottom) iterations (to a relative tolerance of 10^{-6}) with grid parameters $h = \frac{1}{4}H = \frac{1}{16}$ (circles), $h = \frac{1}{4}H = \frac{1}{32}$ (stars) and $h = \frac{1}{4}H = \frac{1}{64}$ (triangles). The solid lines correspond to A_{2LS2LM}, while the dashed lines correspond to A_{2L2LM}. In the top figure, the dotted line is (5.151) and the dot-dashed line is (5.152). 118

6.1 Scaling of S2LM. 125
6.2 Scaling of 2LM. 125
6.3 Processor that has been assigned the "gray" subdomain Ω_9, refines all the neighbouring subdomains Ω_j with $j < 9$. 127
6.4 Wrench-shaped domain Ω. 131
6.5 2-Level 2-Lagrange Multiplier solver diagram. 131
6.6 `Assemble_Neumann` function diagram. 133
6.7 Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, two subdomains 135
6.8 Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, four subdomains 135
6.9 Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, eight subdomains 136

List of Tables

5.1 Condition numbers for the 2-Level **symmetric** 2-Lagrange multiplier matrix A_{2LS2LM} for the model problem (5.2). 116
5.2 Condition numbers for the 2-Level **nonsymmetric** 2-Lagrange multiplier matrix A_{2L2LM} for the model problem (5.2). 116

6.1 Iteration counts for S2LM. 124
6.2 Iteration counts for 2LM. 124
6.3 Iteration counts for 2LS2LM. 130
6.4 Walltime for for 2LS2LM. 130
6.5 Iteration counts for 2L2LM. 130
6.6 Walltime for 2L2LM. 130

Chapter 1

Introduction

Numerical Analysis lies in the meeting point of pure mathematics, computer science and application areas. It often attracts some degree of hostility from all three. [8]

.

1.1 Introduction

A **partial differential equation** (PDE) is an equation containing an unknown multivariate function and its partial derivatives. PDEs can be used to model a wide range of phenomena such as sound, heat, elasticity, fluid flow, and more. For example, Laplace's equation for heat in a metal plate is

$$\Delta\tilde{u} = -\tilde{u}_{xx} - \tilde{u}_{yy} = \tilde{f} \text{ in } \Omega, \tag{1.1}$$

where the **domain** $\Omega \subset \mathbb{R}^2$ describes the shape of the metal plate. The **forcing** function $\tilde{f}$, defined on Ω, describes the amount of heat per unit time being pumped into the plane at each point of Ω. A PDE such as Laplace's equation must be supplemented by **boundary conditions** in order to obtain a unique solution; e.g. **Dirichlet** conditions $\tilde{u} = 0$ on $\partial\Omega$.

In [28], Joseph Fourier provided the first solution method for Laplace's equation. In modern terminology, his method is as follows. For the plate $\Omega = (0, \pi) \times (0, \pi)$, begin with the ansatz that $\tilde{u}(x, y) = \sum_{j,k=1}^{\infty} \hat{u}(j, k) \sin(jx) \sin(ky)$. Writing a corresponding Fourier series for $\tilde{f}(x, y)$ and differentiating term-by-term leads to $\hat{u}(j, k) = \hat{f}(j, k)/(j^2 + k^2)$.

Fourier's method works because the eigensolutions $\Delta\tilde{u} = \lambda\tilde{u}$ of the Laplacian on the square (with homogeneous Dirichlet boundary conditions) are of the form

$\sin(jx)\sin(ky)$. To apply Fourier's approach to a general domain Ω would require knowing the eigenvectors of the Laplacian on Ω; an impossible task except for the simplest domain shapes (e.g. rectangles and discs).

An equation that describes the bending of a plate is the biharmonic equation

$$\tilde{u}_{xxxx} + 2\tilde{u}_{xxyy} + \tilde{u}_{yyyy} = \tilde{f}.$$

In [19], Chladni described intriguing patterns formed when a sand-covered metal plate is caused to vibrate by drawing a violin bow over it. This equation is more challenging to analyze by Fourier's technique and the problem remained open until [56], where Ritz designed the first modern discretization of a PDE. In Ritz's method, the solution $\tilde{u}(x, y)$ is expanded according to some family $\{\phi_i(x, y)\}$ of basis functions. The basis functions do not have to be eigensolutions of the PDE operator but instead are selected on the basis of computational efficiency. For example, in the **Finite Element Method** (FEM), the domain Ω is triangulated and the basis functions are piecewise polynomial on this triangulation.

When we discretize (1.1) using the FEM, we obtain a linear problem

$$Au = f, \tag{1.2}$$

where $A \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix; $\tilde{u} \in \mathbb{R}^n$ is an unknown vector, and $\tilde{f} \in \mathbb{R}^n$ is a given data vector. The "continuous" solution $\tilde{u}(x, y)$ is then approximately $\tilde{u}(x, y) \approx \sum_k u_k \phi_k(x, y)$.

Any linear solver can be used to solve (1.2). However, since the number n of basis functions is large, it is desirable to use a linear solver that scales well for large linear problems. In principle, Gaussian elimination can solve (1.2) in approximately $\frac{2}{3}n^3$ floating point instructions (FLOPS). For the Laplacian, we can improve the estimate somewhat by taking into account the sparsity structure of A, which shows that we will need $O(n^2)$ FLOPS in 2d or $O(n^{2.33\cdots})$ FLOPS in 3d. Nevertheless, the computational cost grows much faster than the problem size.

Another difficulty with Gaussian elimination is that it is an inherently sequential algorithm. Seemingly presciently, G.E. Moore [51] stated that the number of transistors on a CPU chip would double approximately every two years. CPU power consumption grows proportionally to CPU frequency; this increasing amount of power dissipates as heat. Waste heat must be removed from the CPU using a "heat sink". The maximal rate at which we can remove heat from the CPU becomes a limit on CPU frequency, known as the "power wall". Since CPU can no longer increase the clock frequencies, manufacturers have used the increasing number of transistors to provide multiple CPU cores, which must then compute in parallel; see [36] and [52].

Parallelism in supercomputers was introduced in the 1960's to tackle real-world

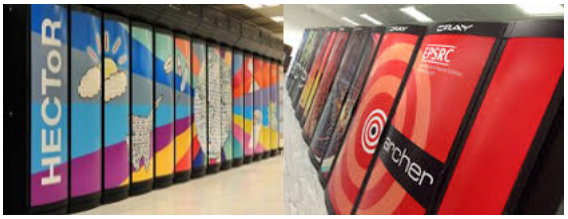


Figure 1.1: Images of HECToR supercomputer and its successor ARCHER based at EPCC.

Figure 1.2: Projected performance development of top 500 Supercomputers, [http://top500.org/statistics/perfdevel/, data: June 2015].

problems in fields such as climate modeling, data analysis, nanosciences, molecular biology and many others. Modern supercomputers feature tens of thousands of processors and with the ability to provide thousands of Gigaflops of performance [52]. The British academic national supercomputer HECToR (High End Computing Terascale Resources) and its successor ARCHER (Academic Research Computing High End Resource), based at the Edinburgh Parallel Computing Centre (EPCC), c.f. Figure 1.1, can provide 800 Teraflops and 1.56 Petaflops of theoretical peak performance respectively. The exascale era is expected to begin in 2018; see Figure 1.2.

On such massively parallel architectures, sequential solvers such as Gaussian elimination can no longer be used and one must instead resort to **iterative solvers**. In an iterative solver, one obtains successive approximations $u^{(1)}, u^{(2)}, \ldots$ to the equation (1.2), such that $\lim_k u^{(k)} = u$. Three requirements guide the design of an iterative solver:

1. Computing $u^{(k+1)}$ from $u^{(k)}$ must be efficient.

2. Each iteration must be parallelelizable.

3. The iterates must approach u to a prescribed tolerance in a small number of iterations. If the number of iterations does not depend on the problem size, we say that the iteration is **scalable**.

Iterates of classical iterations such as Jacobi and Gauss-Seidel can be efficiently computed (point 1). The Gauss-Seidel algorithm is not so easy to parallelize, but the Jacobi algorithm is highly parallelizable (point 2). However, the Jacobi algorithm applied to the 2d Laplacian will typically require $O(n^2)$ iterations to produce an approximate solution.

In **domain decomposition**, the domain Ω is partitioned into subdomains $\Omega = \cup_k \Omega_k$; the subdomains may be overlapping or disjoint. On each subdomain, one solves the PDE and one obtains boundary conditions from neighboring subdomains. For example, Schwarz's overlapping method [62] reads

$$-\Delta \tilde{u}_{next} = \tilde{f} \text{ in } \Omega_j \text{ and } \tilde{u}_{next} = \tilde{u}_{prev} \text{ on } \bar{\Omega} \setminus \Omega_j,$$

where j cycles through the various subdomains. This iteration is written in "Gauss-Seidel" style, but one can obtain a parallel version by "coloring" the subdomains in such a way that overlapping subdomains have distinct colors and solving all subdomains of a given color in parallel in a single iteration.

In [47], P.L. Lions proposed a nonoverlapping variant of Schwarz's method where one exchanges Robin instead of Dirichlet data across subdomain interfaces. In other words, the iteration scheme now looks like

$$-\Delta \tilde{u}_{next} = \tilde{f} \text{ in } \Omega_j \text{ and } a\tilde{u}_{next} + D_\nu \tilde{u}_{next} = \tilde{\lambda} \text{ on } \partial\Omega_j,$$

where D_ν denotes the directional derivative in the direction ν of the outwards-pointing unit normal to $\partial\Omega_j$, and $a > 0$ is a **Robin tuning parameter**, which can be chosen freely. The Robin data $\tilde{\lambda}$ must somehow be read off the neighboring subdomains. Points on the **artificial interface** $\Gamma = \Omega \cap \cup_j \partial\Omega_j$ are either **regular interface points** that are adjacent to exactly two subdomains, or **cross points** that are adjacent to three or more subdomains. For any regular interface points $x \in \Gamma$, we may use $\tilde{\lambda}(x) = a\tilde{u}_{prev,neighbor}(x) + D_\nu \tilde{u}_{prev,neighbor}(x)$. However, the situation is more delicate when cross points are present since there are multiple choices of previous iterates at cross points. An added twist of Lions's method is that the iterates of adjacent subdomains typically do not meet continuously across the subdomain boundaries.

Lions proved that his method converges when there are no cross points, and no overlap. In [44] and [49], the convergence of Lions's method for certain overlapping decompositions was proved.

The choice a of the Robin parameter is very important. In his original paper, Lions says "Next, it is worth discussing the effective choice of [the Robin parameter]: let us first indicate that this is by large an open problem." This open problem was widely studied by Fourier analysis, [24], [29], [67]. Such methods became known as **optimized Schwarz methods** (OSM). Although the Fourier approach reveals the correct Robin parameter, it has three important limitations. First, the Fourier approach is only applicable to simple domains and subdomains, such as rectangles. Second, the Fourier approach gives no information about cross points (but see [31]). Third, the Fourier approach does not apply to general elliptic problems.

In [48], Loisel addressed these three points. In order to explain Loisel's idea, we first introduce some notation.

Because the iterates of the OSM do not meet continuously across Γ, in practical implementations, the discrete iterates $u^{(k)}$ are stored as block vectors

$$u^{(k)} = \begin{bmatrix} u_1^{(k)} \\ \vdots \\ u_m^{(k)} \end{bmatrix},$$

where m is the number of subdomains. Each $u_j^{(k)}$ is a solution on subdomain Ω_j at iteration k. We further arrange each $u_j^{(k)}$ into blocks as follows

$$u_j^{(k)} = \begin{bmatrix} u_{Ij}^{(k)} \\ u_{\Gamma j}^{(k)} \end{bmatrix},$$

where the subscript I indicates degrees of freedom in the interior of Ω_j and the subscript Γ indicates degrees of freedom on the artificial interface Γ. The vector $u_{\Gamma j}^{(k)}$ is the trace of $u_j^{(k)}$ along $\partial\Omega_j \cap \Gamma$. We can put together all the traces, as follows:

$$u_G^{(k)} = \begin{bmatrix} u_{\Gamma 1}^{(k)} \\ \vdots \\ u_{\Gamma m}^{(k)} \end{bmatrix},$$

We can think of $u_G^{(k)}$ as a **multi-valued trace**; it is a "function" with multiple values along Γ (one value per adjacent subdomain at any given point on Γ). Note that we use the subscript G in order to keep the notion distinct from a single-valued trace, denoted by the subscript Γ.

Loisel formulated the **symmetric 2-Lagrange multiplier** linear problem:

$$A_{S2LM}\lambda = h_s \tag{1.3}$$

and the **non-symmetric 2-Lagrange multiplier** linear problem:

$$A_{N2LM}\lambda = h_n. \tag{1.4}$$

The solution vector $\lambda \in \mathbb{R}^m$ has the block structure

$$\lambda = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix},$$

where m is the number of subdomains. Each λ_k represents (discrete) Robin data on the part of the artificial interface Γ corresponding to subdomain Ω_k. The "primal" solution u_k can be recovered by solving a discrete Robin problem on Ω_k with the discrete Robin data λ_k. Systems (1.2) and (1.3) are equivalent in the following sense: if one solves (1.3) and then one recovers u_k by solving suitable Robin subproblems, then the u_k meet continuously across Γ and can be glued together to obtain the solution of (1.2). Furthermore, if one applies a suitable Richardson iteration to (1.3), the resulting iteration is equivalent to the OSM, at least when there are no cross points. This is similar to the relationship between BDDC and FETI-DB [13].

This approach has several advantages. First, the analysis and implementation of 2LM (including condition numbers) applies to general domains and subdomains. Second, the analysis and implementation of 2LM is applicable even when there are cross points. Third, the analysis and implementation of 2LM applies for general elliptic problems.

Although Loisel's general framework is excellent, it also reveals an expected problem with 2LM and OSM. Although the OSM is very efficient when there are few subdomains, the condition number of 2LM increases unboundedly when the number of subdomains increases. This is a well-known phenomenon that affects all "1-level" domain decomposition methods. The solution has long been known to introduce a "coarse space", [23]. A first attempt at introducing a coarse grid into an optimized Schwarz method is described in [24]. However, this paper is based on Fourier analysis and hence suffers from the usual flaws (it does not apply to general domains and subdomains, general elliptic equations, or cross points.)

1.1.1 Summary of main results

In this thesis, we develop a new general theory for 2-level 2-Lagrange multiplier methods, for general domain, subdomains and elliptic equations, including cross points. Using our new theory, we provide sharp condition number estimates that show that the method is scalable. We also describe the first massively parallel implementation

of a 2-level 2-Lagrange multiplier method.

Our main theoretical result is to show that the condition number of the 2LM system is $O(\sqrt{H/h})$. As a result, if one enforces that the ratio H/h remains bounded as $h \to \infty$, one obtains a scalable algorithm. Furthermore, our condition number is much smaller than the condition number $O(H/h)$ of a minimally overlapping 2-level Schwarz method.

PETSc [6] is a standard library that implements many algorithms for solving linear problems on supercomputers. For example, PETSc implements the additive Schwarz preconditioner in a "black-box" fashion, as follows. Let A be a given stiffness matrix, and let $R_1, \ldots, R_m$ be "restriction" matrices corresponding to each subdomain (each row of R_j is a row of the identity matrix, see [69]). The additive Schwarz preconditioner is then

$$P_{AS}^{-1} = \sum_j R_j^T A_j^{-1} R_j \text{ where } A_j = R_j A R_j^T. \tag{1.5}$$

We see that, in order to use an additive Schwarz preconditioner, all the user has to do is assemble A and provide the matrices $\{R_j\}$; the rest can be done "automatically" by PETSc.

More sophisticated domain decomposition methods are more difficult to implement in a "black box" manner. For example, FETI methods require the "partial assembly"

$$A = \sum_i R_i^T A_{Ni} R_i, \tag{1.6}$$

where A_{Ni} are stiffness matrices corresponding to the subdomain bilinear forms $\int_{\Omega_i} \nabla u \cdot \nabla v$. These bilinear forms can be interpreted as Neumann problems on Ω_i.

It would seem at first blush that implementing 2LM would require for the user to provide stiffness matrices for Robin problems on the subdomains. One of our innovations is to leverage the partial assembly formula (1.6); our library then automatically generates the Robin subproblems.

The choice of the Robin parameter can significantly impact the performance of the 2-Lagrange multiplier and optimized Schwarz methods. Traditionally, the Robin parameter is chosen by Fourier analysis; this is difficult to automate. We are able to prove that the optimized Robin parameter is the geometric average of the extremal eigenvalues of the subdomain Dirichlet-to-Neumann (D2N) maps. This can be computed efficiently at run time; we do not need to diagonalize fully the D2N and it suffices to only compute the extremal eigenvalues using, e.g., the power and inverse power methods.

Generating the coarse space is challenging for a wide variety of domain decomposition methods. For example, in Schwarz method, a coarse mesh must be provided,

which means that the user must provide information beyond just the R_i matrices and the stiffness matrix A. By contrast, our coarse space is generated fully automatically without any extra data from the user. Our coarse space consists of the indicating function of the floating subdomains. Note that our coarse space is discontinuous.

Our library is applicable to any elliptic problem for general domains and subdomains. We demonstrate the efficacy of our algorithms on sample problems on general domains and subdomains, using the 2d Laplacian for simplicity.

1.1.2 Overview of the thesis

This thesis is organized as follows. In Chapter 2, we provide an overview of the necessary functional analytical tools that we need for our analysis. In Chapter 3, we describe the finite element method. In Chapter 4, we describe Krylov space methods, which are used to accelerate the convergence of the 2-Lagrange multiplier methods. According to [21], Krylov subspace methods,were placed third in the top ten algorithms of in the 20th century [1]. In Chapter 5, we describe and analyze 2-Lagrange Multiplier methods and the related optimized Schwarz methods. In Chapter 6, we describe our massively parallel implementations of the 2LM and give experiment on Hector. Chapter 7 includes our conclusion and suggests possible avenues for future work. Further details of the implementation are provided in Appendix A. Some details of the proof of the condition number estimate are in Appendix B.

[1]The full list of the top ten algorithms in the 20th century according to [21]. [21], *a*) Metropolis Algorithm for Monte Carlo *b*) Simplex Method for Linear Programming *c*) Krylov Subspace Iteration Methods *d*) The Decompositional Approach to Matrix Computations. *e*) The Fortran Optimizing Compiler *f*) QR Algorithm for Computing Eigenvalue *g*) Quicksort Algorithm for Sorting *h*) Fast Fourier Transform *i*) Integer Relation Detection *j*) Fast Multipole Method .

Chapter 2

Sobolev Spaces and the Variational Formulation of Elliptic PDEs

The Sobolev spaces are the cornerstone of modern theory of partial differential equations. They were named after the significant Russian mathematician S.L. Sobolev. Sobolev spaces are vector spaces of functions, with "weak" derivatives that satisfy certain integrability conditions, [1].

Moreover as it is pointed out in [14], Sobolev spaces are one of the significant cases where we can observe the strong interconnection between different fields of mathematics. In this particular case we observe the connection between functional analysis and PDEs, since abstract results from functional analysis can be applied in order to solve PDEs problems.

In this Chapter we introduce the necessary background on Sobolev Spaces, in order establish the necessary theoretical background in order to describe the weak formulation of PDEs and the Finite Element Method. In this context we define Lebesgue spaces, weak derivatives, the notion of a variational formulation of elliptic partial differential equations and some important existence and uniqueness theorems. We follow the treatment on these topics from books, [14], [12].

2.1 Sobolev Spaces

Let u be a Lebesgue measurable function, we define the Lebesgue space

$$L_p(\Omega) = \{u : ||u||_p < \infty\},$$

where $||u||_p = (\int_\Omega |u|^p d\mu)^{1/p}$, for $1 \leq p < \infty$ and

$$||u||_\infty = \text{esssup}\{|u(x)|,\, x \in \Omega\},$$

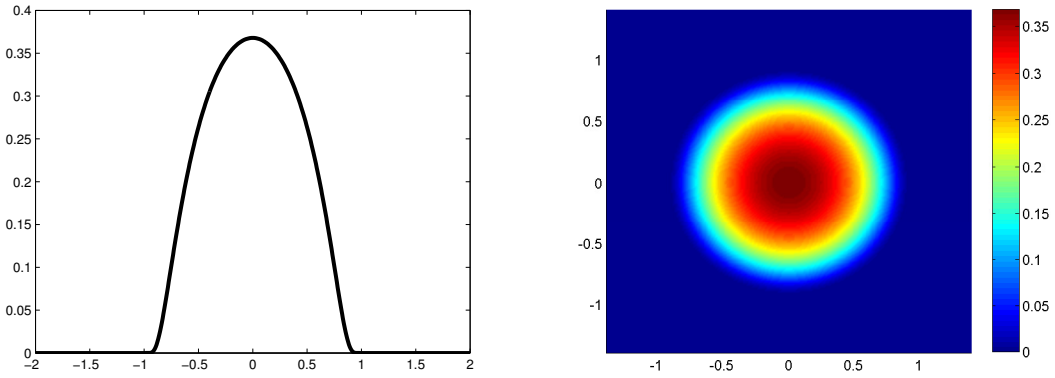


Figure 2.1: Function (2.1) in one and two dimentions

for $p = \infty$. For $p = 2$, the space $L_2(\Omega)$ is a Hilbert space equipped with the inner product

$$(u, v) = \int_\Omega uv d\mu.$$

Definition 2.1.1. *We define the support of a real valued function u on a subdomain $\Omega \subset \mathbb{R}^d$, as $supp(u) = \overline{\{x \in \Omega \ \ s.t\ u(x) \neq 0\}}$.*

Definition 2.1.2. *Let Ω be an open domain in Ω. Then $\mathcal{D}(\Omega) = C_0^\infty(\Omega)$, where $C_0^\infty(\Omega)$ is the space of C^∞ functions with compact support in Ω.*

Let $d \in \mathbb{N}$, we define the multi-index $a = (a_1, a_2, \ldots, a_n)$ and $|a| = a_1 + a_2 + \ldots + a_n$, then for $u \in \mathcal{D}(\Omega)$,

$$D^a u = \left(\frac{\partial}{\partial x_1}\right)^{a_1} \ldots \left(\frac{\partial}{\partial x_n}\right)^{a_n} u.$$

A standard example of a function that belongs to $\mathcal{D}(\mathbb{R}^n)$ is the function, see Figure (2.1),

$$u(x) = \begin{cases} e^{-\frac{1}{1-||x||^2}} & \text{if } ||x|| < 1 \\ 0 & \text{if } \|x\| \geq 1 \end{cases} \tag{2.1}$$

Definition 2.1.3. *We define the dual space of $\mathcal{D}(\Omega)$, $\mathcal{D}'(\Omega)$, as the space of distributions and let*

$$L^1_{loc}(\Omega) = \{u \in L^1(K) \ for\ all\ compact\ K\ \ \subset \Omega\}.$$

Then for a locally integrable $u \in L^1_{loc}$ we define the linear functional $T_u : C_0^\infty(\mathbb{R}) \to \mathbb{R}$,

$$T_u(\phi) = \int_\Omega u\phi d\mu. \tag{2.2}$$

*The space L^1_{loc}, the space of locally integrable functions, is also referred as the space of **regular distributions** since every element of $u \in L^1_{loc}(\Omega)$ corresponds to $T_u \in D'(\Omega)$ thought the one-to-one mapping (2.2).*

Corollary 2.1.4. *Let Ω be a bounded subset of $\mathbb{R}^d$ and let $L^p(\Omega) \subset L^1_{loc}(\Omega)$, $1 \leq p < \infty$. From Hölder's inequality and the definition of L^1_{loc} we get that for all compact $K \subset \Omega$,*

$$\|\mathbf{1}_K u\|_{L^1(\Omega)} \leq |K|^{(1-1/p)} \|u\|_p \leq \infty.$$

Hence the functions in L^p are regular distributions.

Definition 2.1.5. *[12] A function $u \in L^1_{loc}(\Omega)$ has a **weak derivative** $D^a u$, in the distribution sense, provided there exists a function $v \in L^1_{loc}(\Omega)$ such that,*

$$\int u D^a \phi d\mu = (-1)^{|a|} \int_\Omega v \phi d\mu \ \ for\ all\ \phi \in \mathcal{D}(\Omega).$$

If such v exists, we define $D^a u = v$.

Definition 2.1.6. *Let $\Omega \subset \mathbb{R}^d$ and $1 \leq p < \infty$ for any integer $k \geq 1$, $|a| \leq k$, we define the Sobolev Spaces,*

$$W^k_p(\Omega) = \{u \in L^p(\Omega) \text{ such that } \|u\|_{W^k_p(\Omega)} < \infty\}$$

where the norm

$$\|u\|_{W^k_p(\Omega)} = \Big(\sum_{|a| \leq k} \|D^a u\|^p_{L^p(\Omega)} \Big)^{1/p}$$

and the seminorm

$$|u|_{W^k_p(\Omega)} = \Big(\sum_{|a| = k} \|D^a u\|^p_{L^p(\Omega)} \Big)^{1/p}.$$

We set $H^k = W^k_2$, the space H^k is a Hilbert space, equipped with the inner product

$$(u, v)_{H^k} = \sum_{|a| \leq k} (D^a u, D^a v)_{L^2} \tag{2.3}$$

For the special case $k = 1$, the inner product has the form,

$$(u, v)_{H^1} = (u, v)_{L^2} + \sum_{i=1}^{n} (\frac{\partial u}{\partial x_i}, \frac{\partial v}{\partial x_i})_{L^2},$$

where $\frac{\partial u}{\partial x_i}$ has to still be understood in the weak sense and the associated norm

$$\|u\|_{H^1} = \Big(\|u\|^2_{L^2} + \sum_{i=1}^{n} \|\frac{\partial u}{\partial x_i}\|^2_{L^2} \Big)^{1/2}.$$

From [69] we get that for Sobolev spaces of fractional order, we can define an equivalent intrinsic norm defined in terms of integrals over a domain Ω as follows.

Definition 2.1.7. *([69]) Let $\sigma \in (0,1)$, then the norm*

$$\|u\|_{H^\sigma(\Omega)} = (\|u\|^2_{L^2(\Omega)} + |u|^2_{H^\sigma(\Omega)})^{1/2},$$

with the seminorm

$$|u|^2_{H^\sigma(\Omega)} = \int_\Omega \int_\Omega \frac{|u(x)-u(y)|^2}{|x-y|^{2\sigma+n}} dxdy,$$

provides an equivalent norm, equivalent to the one defined by the inner product (2.3)*, in H^σ. Moreover, let $s > 0$, with $[s]$ the integer part of s and $\sigma = s - [s]$. Then an equivalent norm for H^s is given by,*

$$(\|u\|_{H^{[s]}(\Omega)} + |u|^2_{H^s(\Omega)})^{1/2},$$

with the seminorm,

$$|u|^2_{H^s(\Omega)} = \sum_{|a|=[s]} \int_\Omega \int_\Omega \frac{|D^a u(x) - D^a u(y)|^2}{|x-y|^{2\sigma+n}} dxdy.$$

Finally we define H^s_0 as the closure of $\mathcal{D}(\Omega)$ in $H^s(\Omega)$ and the space H^{-s}, the space of bounded linear functional on H^s_0 as the dual of H^s_0, equipped with the norm,

$$||u||_{H^{-s}(\Omega)} = \sup_{u \in H^s_0} \frac{\langle u, v\rangle}{||u||_{H^s(\Omega)}}$$

Theorem 2.1.8. *[5],[14] Let $u \in H^k(\Omega)$, $k > 1/2$, then there exists the trace $u|_{\partial\Omega}$ of the function u on Ω with $u \in H^{(k-1/2)}(\partial\Omega) \subset L^2(\partial\Omega)$,*

$$\|u\|_{H^{(k-1/2)}(\partial\Omega)} \leq C\|u\|_{H^k(\Omega)}$$

where C is independent of u.

Remark 2.1.9. *The standard proof of Theorem* 2.1.8 *is given through Interpolation spaces [20], [50]. For our proof we use the Fourier series expansion of functions $u \in H^k(\mathbb{R}^d)$. We define the Fourier transforms for $u \in L^1(\mathbb{R}^d)$ as,*

$$(\mathcal{F}u)(\omega) = \int_{\mathbb{R}^d} u(x) e^{-2\pi i \omega^T x} dx.$$

1. *For any $u \in L^1(\mathbb{R}^d) \cap L^2(\mathbb{R}^d)$ one has that $||u||^2_L = ||\mathcal{F}u||_{L^2}$ (Parseval's identity). One may thus define $\mathcal{F}$ on L^2 by density.*

2. *If $\hat{u} = \mathcal{F}u \in L^1(\mathbb{R}^d)$ then for any $\omega \in \mathbb{R}$, the inverse Fourier transform is*

$$(\mathcal{F}^{-1}\hat{u}) = \int_{\mathbb{R}^d} \hat{u}(\omega) e^{2\pi i \omega^T x} d\omega.$$

3. *If $u \in H^1(\mathbb{R}^d)$ then,*

$$\mathcal{F}(\frac{\partial u}{\partial x_j}) = -2\pi i \omega_j \mathcal{F}u.$$

As a result the H^1 norm becomes,

$$||u||^2_{H^1}(\mathbb{R}^d) = \int_{\mathbb{R}^d} (1 + 4\pi^2 \omega^T \omega)|\hat{u}(\omega)|^2 d\omega.$$

4. *Generally speaking, we define the $H^s(\mathbb{R}^d)$ norm by,*

$$||u||^2_{H^s(\mathbb{R}^d)} = \int_{\mathbb{R}^k} (1 + 4\pi^2 \omega^T \omega)^s |\hat{u}(\omega)|^2 d\omega.$$

5. *The Fourier transform of a compactly supported smooth function is rapidly decreasing.*

6. *The Hölder inequality is $\int uv \leq ||u||_{L^p}||v||_{L^q}$, where $p, q \in [1, \infty]$ and $\frac{1}{p} + \frac{1}{q} = 1$.*

We prove Theorem 2.1.8 for the case when $k = 1$.

Theorem 2.1.10. *For a smooth and compactly supported $u \in H^1(\mathbb{R}^d)$, we define the trace γu by,*

$$(\gamma u)(x) = u(x, 0) \text{ for, } x \in \mathbb{R}^{d-1}.$$

We show that $||\gamma u||_{H^{1/2}(\mathbb{R}^{d-1})} \leq c_1 ||u||_{H^1(\mathbb{R}^d)}$. We thus define γ on all of $H^1(\mathbb{R}^d)$ by density and continuity. This γ is surjective onto $H^{1/2}(\mathbb{R}^d)$ with a continuous left inverse γ^{-1} and

$$||\gamma^{-1} u||_{H^1(\mathbb{R}^d)} \leq c_2 ||u||_{H^{1/2}(\mathbb{R}^{d-1})}.$$

Proof. Let $z = (x, y) \in \mathbb{R}^{d-1} \times \mathbb{R}$. and $\omega = (\alpha, \beta) \in \mathbb{R}^{d-1} \times \mathbb{R}$. From the naive Fourier inversion Formula for function $f(y)$, we find that $f(0) = \int_{\mathbb{R}} \hat{f}(\beta) d\beta$ provided eg. $f(y)$ is smooth and compactly supported. Thus setting $v(x) = u(x, 0)$, we find that

$$\hat{v}(\alpha) = \int_{\mathbb{R}} \hat{u}(\alpha, \beta) d\beta.$$

We now want to estimate $\hat{u}(a)$ using Hölder inequality:

$$|\hat{v}(\alpha)| = |\int_{\mathbb{R}} \hat{u}(\alpha, \beta) d\beta|$$

$$= |\int_{\mathbb{R}} (1 + 4\pi^2 ||a||^2 + 4\pi\beta^2)^{-1/2} (1 + 4\pi^2 ||a||^2 + 4\pi^2 b^2)^{1/2} \hat{u}(\alpha, \beta) d\beta|$$

$$\leq ||(1 + 4\pi^2 ||a||^2 + 4\pi^2 \beta^2)^{-1/2}||_{L^2(\beta)} ||(1 + 4\pi^2 ||a||^2 + 4\pi^2 \beta^2)^{1/2} \hat{u}(\alpha, \beta)||_{L^2(\beta)}$$

$$\leq \frac{1}{2(1 + 4\pi^2 ||a||^2)^{1/4}} |(1 + 4\pi^2 ||a||^2 + 4\pi^2 \beta^2)^{1/2} \hat{u}(\alpha, \beta)||_{L^2(\beta)}.$$

Thus, computing the $H^{1/2}$ norm of $v(x)$ gives,

$$||v||^2_{H^{1/2}(\mathbb{R}^{d-1})} = \int_{\mathbb{R}^{d-1}} (1+4\pi^2\alpha^2)^{1/2}|\hat{v}(\alpha)|^2 d\alpha$$

$$\leq \int_{\mathbb{R}^{d-1}} (1+4\pi^2||\alpha||^2)^{1/2}\Big(\frac{1}{2(1+4\pi^2||a||^2)^{1/4}}||(1+4\pi^2||a||^2+4\pi^2\beta^2)^{1/2}\hat{u}(\alpha,\beta)||_{L^2(\beta)}\Big)^2 d\alpha$$

$$= \frac{1}{4}||u||^2_{H^1(\mathbb{R}^d)}$$

For the left-inverse, given $v \in H^{1/2}(\mathbb{R}^{d-1})$, let $\hat{u}(\alpha,\beta) = \hat{v}(\alpha)\frac{2\sqrt{1+4\pi^2||\alpha||^2}}{1+4\pi^2||a||^2+4\pi^2\beta^2}$. Direct calculation shows that $\hat{v}(a) = \int_{\mathbb{R}} \hat{u}(\alpha,\beta)$ and we obtain,

$$||u||^2_{H^1(\mathbb{R}^k)} = \int_{\mathbb{R}^{d-1}}\int_{\mathbb{R}} (1+4\pi^2||\alpha||^2+4\pi^2\beta^2)\Big|\hat{v}(a)\frac{2\sqrt{1+4\pi^2||a||^2}}{1+4\pi^2||a||^2+4\pi^2\beta^2}\Big| d\beta d\alpha$$

$$= \int_{\mathbb{R}^{d-1}}\int_{\mathbb{R}} (1+||\alpha||^2+\beta^2)\Big|\hat{v}(\alpha)\frac{2\sqrt{1+4\pi^2||\alpha||^2}}{1+4\pi^2||a||^2+4\pi^2\beta^2}\Big|^2 d\beta d\alpha$$

$$= \int_{\mathbb{R}^{d-1}}\int_{\mathbb{R}} |\hat{v}(a)|^2 \int_{\mathbb{R}} \frac{4+16\pi^2||\alpha||^2}{1+4\pi^2||a||^2+4\pi^2\beta^2} d\beta d\alpha$$

$$= \int_{\mathbb{R}^{d-1}}\int_{\mathbb{R}} |\hat{v}(\alpha)|^2 2\sqrt{1+4\pi^2||\alpha||^2} d\alpha = 2||v||^2_{H^{1/2}(\mathbb{R}^{d-1})}.$$

□

2.2 Variational Formulation of Elliptic Boundary Value Problems

In the previous section we have presented some basic results on Sobolev spaces and the concept of "weak" derivatives. In this section we will present the weak formulation for elliptic boundary problems and the results that guarantee the existence and uniqueness of the solution for elliptic PDEs.

Definition 2.2.1. *A linear functional ℓ on a linear space $\mathcal{V}$ is a mapping $\ell : \mathcal{V} \to \mathbb{R}$ such that for all $u, v \in \mathcal{V}$*

1. $\ell(u+v) = \ell(u) + \ell(v)$
2. $\ell(\lambda v) = \lambda\ell(v)$, *for all* $\lambda \in \mathbb{R}$.

A linear functional is continuous if $|\ell(v)| \leq C||v||_V$ for all $v \in \mathcal{V}$.

Definition 2.2.2. *A bilinear form, $a(.,.)$ on a linear space $\mathcal{V}$ is a mapping $a : \mathcal{V}\times\mathcal{V} \to \mathbb{R}$ such that for all $u, v \in \mathcal{V}$*

1. $a(u+v,w) = a(u,w) + a(v,w)$ *for all* $w \in \mathcal{V}$

2. $a(u,v+w) = a(u,v) + a(v,w)$ *for all* $w \in \mathcal{V}$

3. $a(\lambda u, v) = a(u, \lambda v) = \lambda a(u,v)$ *for all* $\lambda \in \mathbb{R}$.

Moreover a bilinear form is called symmetric if $a(u,v) = a(v,u)$. *If additionally* $a(u,u) > 0$ *and* $a(u,u) = 0$ *iff* $u = 0$, *then* $a(u,v)$ *defines an inner product.*

Definition 2.2.3. *A bilinear form* $a(.,.)$ *on a Hilbert space* $\mathcal{H}$ *is said to be bounded if there exists* $C < \infty$ *such that*

$$|a(u,v)| \leq C\|u\|_{\mathcal{H}}\|v\|_{\mathcal{H}}, \quad for\ all\ \ u,v \in \mathcal{H};$$

and coercive on $V \subset \mathcal{H}$ *if* $\exists\ c > 0$ *such that*

$$a(v,v) \geq c\|v\|^2_{\mathcal{H}}\ for\ all\ \ v \in V.$$

Proposition 2.2.4. *Let* $\mathcal{H}$ *be a Hilbert space, and suppose* $a(.,.)$ *is a symmetric bilinear form that is continuous on* $\mathcal{H}$ *and coercive on a subspace* V *of* $\mathcal{H}$. *Then* $(V, a(.,.))$ *is a Hilbert space.*

Proof. First we will prove that $a(.,.)$ defines an inner product on the space V. Since $a(u,v)$ is a symmetric bilinear form it satisfies the inner product condition of symmetry and linearity. Moreover since $a(.,.)$ is coercive it follows that $a(u,u) \geq 0$ and that $a(u,u) = 0$ iff $u = 0$. So it is an inner product.

Moreover, we need to prove that the space V with the associated energy norm $\|u\|_E = \sqrt{a(u,u)}$ is complete. Let $\epsilon > 0$ and u_n be a Cauchy sequence in $(V, \|.\|_E)$, then since u_n is a Cauchy sequence and the bilinear form is coercive, then from Definition 2.2.3 we get that $\epsilon > \|u_m - u_n\|_E \geq \sqrt{c}\|u_m - u_n\|_{\mathcal{H}}$. Hence $\{u_n\}$ is also a Cauchy sequence in $(\mathcal{H}, \|.\|_{\mathcal{H}})$. Since $\mathcal{H}$ is a complete Banach space and V is a closed subspace of $\mathcal{H}$ then there exists $u \in \mathcal{H}$ such that $u_n \to u$ in the $\|.\|_{\mathcal{H}}$ norm and $u \in V$. Since $a(.,.)$ is bounded we have that $\|u - u_n\|_E \leq C^{1/2}||u - u_n||_{\mathcal{H}} \to 0$. Hence $(V, \|.\|_E)$ is complete. □

Theorem 2.2.5. *(Riesz representation theorem) Let* $\mathcal{H}$ *be a Hilbert space and* $f : \mathcal{H} \to \mathbb{R}$, $f \in \mathcal{H}^*$, *where the* $\mathcal{H}^*$ *is the dual space of* $\mathcal{H}$, *there exists a unique* $u \in \mathcal{H}$ *such that* $f = f_u$, *where* $f_u(v) = (v,u)$.

Proof. Let $M = \{v \in \mathcal{H}\ :\ f(v) = 0\}$, then M is a closed subset of $\mathcal{H}$. We can assume that $M \neq \mathcal{H}$ since in this case we would get that $f \equiv 0$ and $f = f_0$. Hence if we get $M \neq \mathcal{H}$ then there exists an element $w \neq 0$ orthogonal to M. (Such a w exist, since for Hilbert spaces it holds that if M is closed subspace of $\mathcal{H}$, then $\mathcal{H} = M \oplus M^T$). In [14]

w is defined explicitly with the additional property that $||w|| = 1$. Let $w_0 \in \mathcal{H}$ such that $w_0 \notin M$. Let P_M be the projection from $\mathcal{H}$ to M. Then we define $w_1 = P_M w_0$ and we get

$$w = \frac{w_0 - w_1}{|w_0 - w_1|}.$$

Then for arbitrary $v \in \mathcal{H}$, we set $z = v - \frac{f(v)}{f(w)}w$. Since $w \in M^T$ we get that $f(w) \neq 0$. Moreover, we have that $f(z) = f(v) - \frac{f(v)}{f(w)}f(w) = 0$, hence $z \in M$. Finally,

$$(w, z) = (w, v - \frac{f(v)}{f(w)}) = 0,$$

$$(w, v) = \frac{f(v)}{f(w)},$$

$$f(v) = f(w)(w, v) = (v, f(w)w) = f_u,$$

with $u = f(w)w$. □

Proposition 2.2.6. *Assume that $\mathcal{H}$ is a Hilbert space, $a(.,.)$ is a coercive and bounded symmetric bilinear form and V is a closed subspace of $\mathcal{H}$, then we can write the weak* ***variation formulation*** *of an elliptic boundary value problem as: Let $f \in V^*$, find $u \in V$ such that*

$$a(v, u) = f(v) \ for \ all \ v \in V. \tag{2.4}$$

Moreover there exists a unique solution $u \in V$, of (2.4).

Proof. Since $(V, a(.,.))$ is a Hilbert space, we can apply the Riesz representation theorem. □

A very important theorem that is widely used to prove the existence and uniqueness of solution for the variational formulation of PDEs, is the LaxMilgram theorem.

Theorem 2.2.7. *(Lax-Milgram) Assume that $a(.,.)$ is a continuous coercive bilinear form on $\mathcal{H}$. Then for any $f \in \mathcal{H}^*$, there exists a unique element $u \in \mathcal{H}$ such that*

$$a(v, u) = f(v) \ for \ all \ v \in V. \tag{2.5}$$

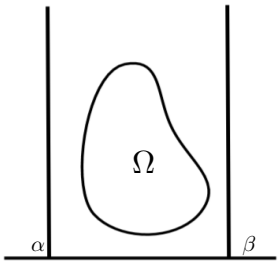
Moreover, if $a(.,.)$ is symmetric,

$$\frac{1}{2}a(u, u) - f(u) = \min_{v \in \mathcal{H}}\{a(v, v) - f(v)\} \tag{2.6}$$

Proof. Page 140, Corollary 5.8, [14] □

Proposition 2.2.8. *(Poincaré's inequality) Assume that $\Omega \subset \mathbb{R}^d$ is open and bounded in some direction. Then there exists a constant C, depending on Ω, such that*

$$||u||_{L^2(\Omega)} \leq C\|\nabla u\|_{L^2(\Omega)}, \ \ for \ all \ u \in H_0^1(\Omega)$$



Proof. We assume that $d = 2$ but the proof can be easily generalised for $d > 2$. Let $u(x, y) \in C_0^\infty(\Omega)$ and Ω be bounded in some direction with width $d = \alpha - \beta$. From the fundamental theorem of calculus we have that,

$$u(x,y) = \int_\alpha^x u_x(\xi, y)d\xi + \overbrace{u(\alpha, y)}^{=0}$$

and

$$\int_\alpha^\beta u(x,y)^2 dx = \int_\alpha^\beta [\int_\alpha^x u_x(\xi, y)d\xi]^2 dx.$$

From Jensen's inequality, [39], we get

$$\int_\alpha^\beta u(x,y)^2 dx \leq \int_\alpha^\beta (x-\alpha) \int_\alpha^x u_x^2(\xi, y)d\xi dx$$

$$\leq \int_\alpha^\beta (\beta-\alpha) \int_\alpha^\beta u_x^2(\xi, y)d\xi dx$$

$$= (\beta-\alpha)^2 \int_\alpha^\beta u_x^2(\xi, y)d\xi.$$

Hence,

$$\int_\Omega u^2 = \int_{-\infty}^\infty \int_\alpha^\beta u^2 dxdy$$

$$\leq (\beta-\alpha)^2 \int_{-\infty}^\infty \int_\alpha^\beta u_x^2(\xi, y)d\xi dy$$

$$\leq (\beta-\alpha)^2 \int_{-\infty}^\infty \int_\alpha^\beta [u_x^2 + u_y^2]d\xi dy = (\beta-\alpha)^2 |u|^2_{H^1(\Omega)}.$$

for every $u \in C_0^\infty(\Omega)$. Since $\overline{C^\infty}(\Omega) = H_0^1(\Omega)$ the proof is complete. □

Proposition 2.2.9. *Assume $\Omega \subset \mathbb{R}^d$ is a bounded domain with a Lipschitz continuous boundary Γ, then the Green's formula for arbitrary $u \in C^2(\overline{\Omega})$, $v \in C^1(\overline{\Omega})$ reads:*

$$-\Delta u v dx = -\int_\Gamma \frac{\partial u}{\partial \nu} v ds + \int_\Omega \nabla u \nabla v dx, \tag{2.7}$$

where $\frac{\partial u}{\partial \nu}$ *is the directional derivative of* u *in the direction of the outward pointing normal* ν*.*

Remark 2.2.10. *In order to present the 2-Lagrange multiplier methods in Chapter 5, we will use as a model problem the Poisson equation with Dirichlet boundary conditions. Nevertheless, our methods can be applied in a wide range of elliptic PDEs for which their corresponding bilinear forms are self-adjoint and coercive, like the reaction-diffusion equation.*

We consider the second order elliptic partial differential equation,

$$-\Delta u = f \text{ in } \Omega \tag{2.8}$$

$$u = 0 \text{ on } \partial\Omega$$

where Δ denotes the Laplace operator $\Delta = \sum_{k=1}^{n} \frac{\partial^2}{\partial x_i^2}$, with $f \in C^1(\Omega)$ and Ω has a Lipschitz boundary with positive measure.

Given $f \in C^1(\Omega)$, the strong solution of (2.8) is a function $u \in C^2(\overline{\Omega})$ that satisfies (2.8) and the boundary conditions. Multiplying both sides of (2.8) by a smooth test function $v \in C_0^\infty(\Omega)$ and by using the Green's formula (2.7) we get that

$$-\int_\Omega \Delta u v dx = \int_\Omega \nabla u \nabla v dx = \int_\Omega f v dx, \tag{2.9}$$

since $u = 0$ on the boundary $\partial\Omega$, $\int_{\partial\Omega} \frac{\partial u}{\partial \nu} v ds = 0$.

Since $\overline{C_0^\infty(\Omega)} = H_0^1(\Omega)$ we can choose choose our test functions such that $v \in H_0^1(\Omega)$. Moreover if we assume that $f \in (H_0^1)^*$, the weak formulation of the boundary problem (2.8) becomes:

Find $u \in H_0^1(\Omega)$ such that

$$a(u, v) = (f, v) \text{ for all } v \in H_0^1, \tag{2.10}$$

where $a(u, v) = \int_\Omega \nabla u \, \nabla v dx$ and $(f, v) = \int_\Omega f v dx$.

We can easily prove that problem (2.10) has a unique solution $u \in H_0^1(\Omega)$ by using Proposition 2.2.8. The bilinear form $a(u, u)$ in (2.10) is coercive, since from the Poincaré's inequality

$$a(u, u) = (\nabla u, \nabla u) = |u|_{H_0^1}^2 \geq c_1 \|u\|_{H_0^1}^2$$

and bounded since from the Cauchy-Schwarz inequality we get

$$a(u, v) = (\nabla u, \nabla v) \leq \|u\|_{H_0^1} \|v\|_{H_0^1}.$$

Moreover $u \in H_0^1 = W_{0,2}^1$, hence from Proposition 2.2.8 and the Cauchy-Schwarz

inequality we get

$$(f, v) \leq \|f\|_{L^2}\|v\|_{L_2} \leq C\|f\|_{L^2}\|\nabla v\|_{L_2} \leq C'\|v\|_{H_0^1}$$

Since problem (2.10) satisfies all conditions of Theorem 2.2.7, we conclude that problem (2.10) has a unique solution $u \in H_0^1(\Omega)$.

Chapter 3

Finite Element Method

Solving problems arising from various scientific fields that can be described by partial differential equations, can prove a very difficult task. Hence, we have to use efficient numerical methods in order to find a numerical approximation of their solution.

Such numerical methods are the Galerkin methods and especially the Finite Element Method. These methods are of significant importance for the field of numerical analysis, since they give us the flexibility to discretise the problems over very complex geometries in 2 and 3 dimensions and they have been extensively studied and numerically tested for more than four decades.

A typical way to solve numerically a PDE problem, is first to discretise the problem and then to find an approximate discrete solution of the problem. The solution lies in a finite dimensional subspace of our original infinite dimensional space. The Galerkin method provides a general framework for the finite dimensional approximation of partial differential equations.

After the discretisation of a PDE, in general we get a large discrete system with millions or billions of degrees of freedom. As science evolves there is an increasing demand for accurate approximations of solutions of complicated problems and these problems become harder and even more complex. Hence there is natural need of numerical methods that exploit the increased computational capabilities of the new high performance systems and the concept of parallelism.

The Galerkin type methods and more precisely the Finite Element Methods are one of the most widespread and well understood methods for solving the numerical solution of PDE problems, [4], [61], [12], [11].

3.1 Galerkin Method

In this section we will use as a model problem, the weak formulation of the Poisson problem (2.10). Instead of solving the infinite dimensional problem we want to find the projection of the solution on a finite dimensional subspace V_h of the infinite

dimensional space V, $V_h \subset V$. The finite dimensional weak formulation reads: find $u_h \in V_h$ such that

$$a(u_h, v) = (f, v) \text{ for all } v \in V_h. \tag{3.1}$$

For the problem (2.10) we have $V \subseteq H_0^1(\Omega)$. When the finite element space V_h is a subset of the infinite dimensional space V, $V_h \subset V$, then we have the case of conforming elements. Nevertheless, for certain problems we can allow the "variational crime" where $V_h \nsubseteq V$, the case of non-conforming-elements. From Theorem 2.2.7 (Lax-Milgram theorem), (3.1) has a unique finite dimensional solution.

We assume the finite dimensional set of functions $\{\phi_i\}_{i=1}^n$ with $n \in \mathbb{N}$, forms the finite dimensional base of the space $V_h \subset H_0^1$. The finite dimensional solution can be written as a linear combination of the basis functions ϕ_i, $u_h = \sum_{i=1}^n u_i \phi_i$ and consequently Problem (2.10) can be written in a matrix form as

$$Au_h = f_h,$$

where A is the $n \times n$ stiffness matrix, the $n \times 1$ vector f_h is the projection of f on V_h and the $n \times 1$ vector u_h is the approximate solution. Here we can note that in the case of second order elliptic PDEs the matrix A is a large sparse matrix, symmetric and positive definite. Standard direct solvers, like Gaussian elimination, are not the most efficient way to solve this type of problems as we have discussed in Section 1.1. Instead iterative methods, sequential or parallel, are more appropriate for these large scale problems.

Proposition 3.1.1. *(Céa's inequality) Assume that V is a Hilbert Space and $V_h \subseteq V$ be a subspace of V. Moreover, let $a(.,.)$ be a bilinear form continuous and coercive on V and $f \in V'$. Then if $u \in V$ is a solution for the variational problem* (2.4) *and $u_h \in V_h$ is the Galerkin approximation of the solution* (3.1) *we have the following error estimate.*

$$\|u - u_h\|_V \leq \frac{c_1}{c_2} \min_{v \in V_h} \|u - v\|_V, \tag{3.2}$$

where c_1, c_2 are positive real constants defined in the proof.

Proof. Let $v \in V_h \subset V$, subtracting $a(u_h, v) = f(v)$ from $a(u, v) = f(v)$ we get the orthogonality condition

$$a(u - u_h, v) = 0, \text{ for all } v \in V_h$$

Moreover since $a(u, v)$ is coercive and bounded bilinear form on V we get that

$$c_2 \|u - u_h\|_V^2 \leq a(u - u_h, u - u_h) = a(u - u_h, u - v) + a(u - u_h, v - u_h)$$

$$= a(u - u_h, u - v) \leq c_1 \|u - u_h\|_V \|u - v\|_V.$$

Dividing by $\|u - u_h\|$ we get that

$$\|u - u_h\|_V \leq \frac{c_1}{c_2} \|u - v\|_V \text{ for all } v \in V_h$$

which is equivalently written as

$$\|u - u_h\|_V \leq \frac{c_1}{c_2} \min_{v \in V_h} \|u - v\|_V.$$

□

Hence a general error estimate between the exact and the approximate solution is given by (3.2).

3.2 Finite Element Method

The Finite Element Method can be seen as a special case of the Galerkin method, when the finite dimensional space V_h is spanned by piecewise polynomials, associated with the partitioning of the domain. Let us define $\Pi_h u$ as the projection of u on the finite element space V_h. The projection of the solution from an infinite dimensional space to a finite dimensional space lead to some errors. The error estimates for such methods are of the type

$$\|u - \Pi_h u\|_V \leq C h^\alpha,$$

where $0 < h < 1$ and α is a positive real scalar.

In order to achieve smaller error between the exact solution of the weak problem and the approximate solution, we can either refine the mesh or increase the polynomial degree, or both, which are commonly referred as the h, p and h-p version respectively.

The construction of the finite element space consists of two main steps. We start from a polygonal domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, with boundary $\partial\Omega$. We divide Ω into elements and get our **triangulation** T_h. Then we use this triangulation in order to define the finite element space V_h.

In order to understand the analysis and the implementation of our methods it is useful to describe in detail this procedure and to present some important analytical results related with the finite element spaces.

The triangulation T_h of a polygonal domain $\Omega \subset \mathbb{R}^d$ consists of shape elements K_i, which can be triangles or rectangles in 2 dimensions and tetrahedral or hexahedral in 3 dimensions, also referred to as cells. We use the term triangulation and the notation T_h when Ω is discretized by any type of shape elements.

A formal definition of a triangulation is:

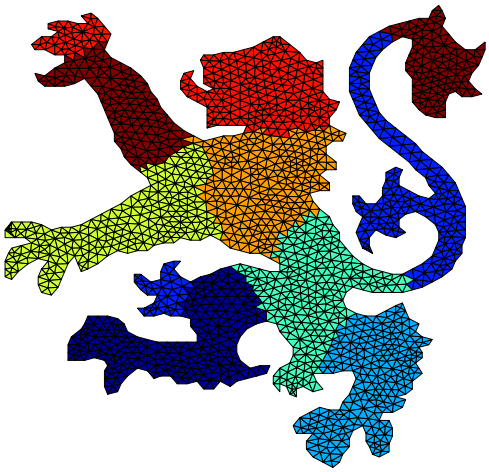


Figure 3.1: Triangulation of a polygonal mesh Ω, partitioned in 8 non-overlapping subdomains by METIS. [43]

Definition 3.2.1. *A triangulation $T_h(\Omega)$ of a polygonal domain $\Omega \subset \mathbb{R}^d$ is the partition of Ω into elements K_i such that,*

1. $\cup_{i=1}^{d} \overline{K_i} = \overline{\Omega}$
2. $K_i \cap K_j = \emptyset$ *for* $i \neq j$
3. $0 < diam(K_i) < h$.
4. $\overline{K_i} \cap \overline{K_j}$ *is either empty, a vertex, or an edge.*

Remark 3.2.2. *Efficient data structures are required in order to store and access the information related to the triangulation T_h. The main information we want to store is the matrix P that holds the coordinates of the nodes and the connectivity matrix of the underlying graph, which is denoted by T. If we assume that a triangulation T_h consists of m triangles and n nodes, the connectivity matrix T is of size $3 \times m$ and matrix P of size $2 \times n$. In the general case of d-dimensions, T is of size $(d+1) \times m$ and P of size $d \times m$. For example in the case of a 2-dimensional square domain $[0,1]^2$ we get the following matrices P and T, see Figure 3.2,*

$$P = \begin{bmatrix} 0 & 1.0 & 1.0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0.5 \\ 1.0 & 1.0 & 0 & 0 & 1.0 & 0.5 & 0 & 0.5 & 0.5 \end{bmatrix}, \quad T = \begin{bmatrix} 4 & 2 & 2 & 4 & 1 & 3 & 5 & 7 \\ 9 & 9 & 5 & 7 & 8 & 6 & 8 & 6 \\ 8 & 6 & 9 & 9 & 5 & 7 & 9 & 9 \end{bmatrix}$$

.

Another interesting concept is that of "the measure of the quality" of each element K_i. We are interested on the measure of quality for both theoretical and practical reasons. "Bad" elements, e.g. elements with very acute angles, will lead to bad approximations. Hence we introduce the definitions of the shape-regular and quasi-uniform triangulations.

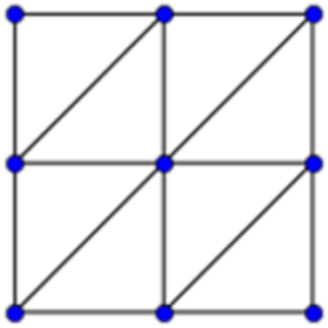


Figure 3.2: Square 2D mesh.

Definition 3.2.3. *Let d_{K_i} be the diameter of the largest inscribed circle of the shape element K_i, $h_{K_i} = diam(K_i)$ and h as defined in Definition* 3.2.1. *Then for a triangulation T_h we define the aspect ratio $C_{K_i} = h_{K_i}/d_{K_i}$. The family $\{T_h\}_{h>0}$ is called* ***shape regular*** *if there exists a constant C_0 such that*

$$C_{K_i} \leq C_0 \quad \forall K_i \in T_h.$$

Moreover we say that T_h is ***quasi-uniform*** *if there exists a constant C_1 independent of h such that*

$$h_{K_i} \geq C_1 h \quad \forall K_i \in T_h.$$

The above definitions are useful in order to establish theoretical results for our methods. It is possible to follow other estimates to measure the quality of the elements, e.g. we can use estimates like the ones that appear in [54], [27]. For example we can use the estimate that calculates the ratio between the radius of largest inscribed and the smallest circumscribed circle, see Algorithm 1. In Algorithm 1 we implement a C++ function that takes as an input the triangulation T_h and returns as an output a vector with the quality of each simplicial element. In Figure 3.3 we see a histogram of the quality of the mesh depicted in Figure 3.1.

Definition 3.2.4. *For completeness, we state a formal definition of a finite element space by P.G. Ciarlet. (Ciarlet 1978, [55], [12]).*

1. *Let $K \subseteq \mathbb{R}^d$ be a bounded closed set with nonempty interior and piecewise smooth boundary, the element domain.*

2. *$\mathcal{P}$ be a finite-dimensional space of functions on K, the space of shape functions.*

3. *$\mathcal{N} = \{N_1, N_2, ..., N_n\}$ be the nodal basis for $\mathcal{P}^*$, where N_i $i = 1, \ldots n$ is the set of nodal variables.*

Then the triplet $(K, \mathcal{P}, \mathcal{N})$ is called a finite element space.

Algorithm 1 Mesh quality, [54],[27]

```
std::vector<double> mesh_quality(const std::vector<int> &T, const std::vector<double> &P){
        int size=T.size()/3;
        std::vector<double> m_quality(size,0.0);
        double v11,v12,v21,v22,v31,v32,temp1,temp2,temp3,Rmax,Rmin;

        for(int i=0;i<size;i++){
                v11=(P[2*T[3*i+1]]-P[2*T[3*i]]); v12=(P[2*T[3*i+1]+1]-P[2*T[3*i]+1]);
                v21=(P[2*T[3*i+2]]-P[2*T[3*i]]); v22=(P[2*T[3*i+2]+1]-P[2*T[3*i]+1]);
                v31=(P[2*T[3*i+2]]-P[2*T[3*i+1]]); v32=(P[2*T[3*i+2]+1]-P[2*T[3*i+1]+1]);

                temp1=std::sqrt(v11*v11+v12*v12); temp2=std::sqrt(v21*v21+v22*v22);
                temp3=std::sqrt(v31*v31+v32*v32);

                Rmin=0.5*std::sqrt((temp2+temp3-temp1)*(temp3+temp1-temp2)*\
                                (temp1+temp2-temp3)/(temp1+temp2+temp3));
                Rmax=(temp1*temp2*temp3)/std::sqrt((temp1+temp2+temp3)*\
                                (temp2+temp3-temp1)*(temp3+temp1-temp2)*(temp1+temp2-temp3));
                m_quality[i]=2*Rmin/Rmax;
        }
        return m_quality;
}
```

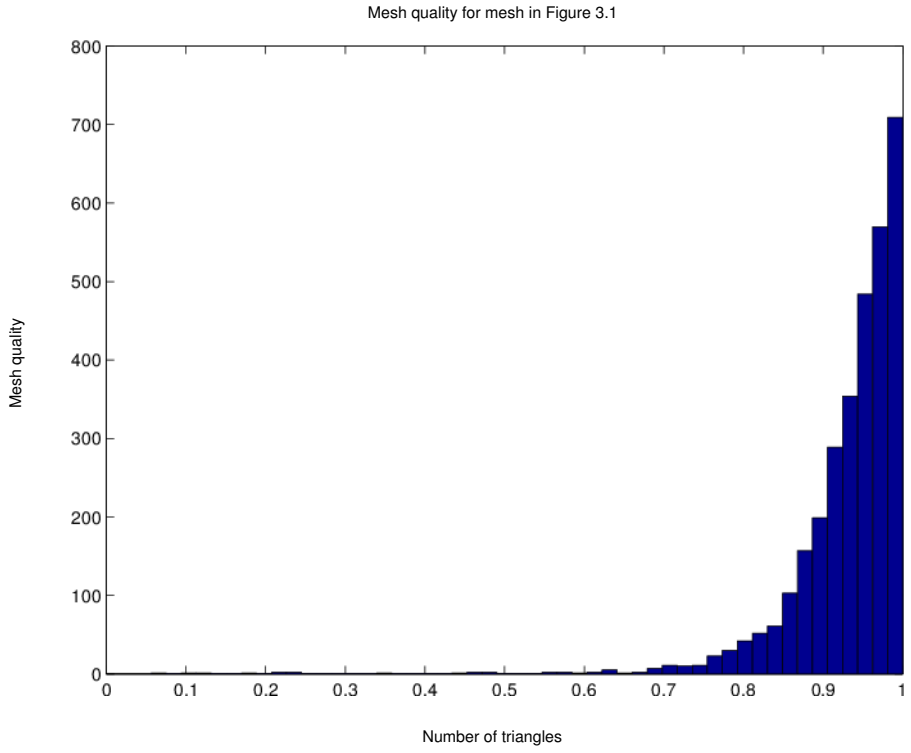


Figure 3.3: Histograph of Mesh Quality for mesh of Fig. 3.1, produced by using Algorithm 1.

An extensive treatment on the Finite Element Method can be found in [12], [18].

Definition 3.2.5. *[12]. Let $(K, \mathcal{P}, \mathcal{N})$ be a finite element space. The basis $\{\phi_1, \phi_2, \ldots, \phi_n\}$ of $\mathcal{P}$ dual to $\mathcal{N}$ is called the nodal space, $\mathcal{N}_i(\phi_j) = \delta_{ij}$.*

Example 1. *Let $K_I = [x_0, x_1]$ be a one dimensional interval and the space of polynomials*

$$P_k(K) := \{v : v = \sum_{i=0}^{k} a_i x^i, \ x \in K\}.$$

For the case that $k = 1$ we get the space of linear polynomials. The linear polynomials $v \in P_1(I)$, see Figure 3.4, can be uniquely determined by the values of of v on x_0, x_1. If we set $N_0(v) = v_0 = v(x_0)$, $N_1(v) = v_1 = v(x_1)$ then we can write

$$v = v_0\phi_0(x) + v_1\phi_1(x)$$

where $\{\phi_0, \phi_1\}$ is the nodal basis for $P_1(K_I)$, with $\phi_i(x_j) = \delta_{ij}$ and

$$\phi_0(x) = \frac{x_1 - x}{x_1 - x_0}, \ \phi_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

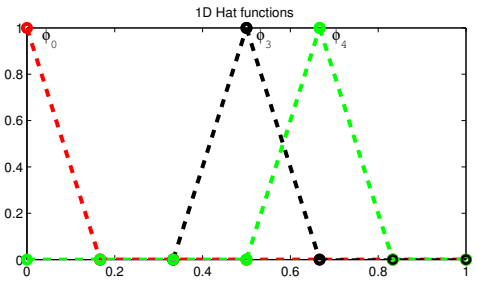


Figure 3.4: Hat functions in 1-dimension

The most commonly used types of Finite Elements are:

1. Triangular Finite Elements, defined over triangles or tetrahedra in 2 and 3 dimensions correspondingly and where the space of shape functions consist of polynomials of degree $\leq k$, $\mathbb{P}_k$, see Figure 3.5.

2. Rectangular Elements, defined over rectangles or parallelepipeds in 2 and 3 dimensions correspondingly and where the space of shape function consist of polynomials,

$$\mathbb{Q}_k = \{\sum_{i=0}^{k} a_i p_i q_i, \ p_i, q_i \in \mathbb{P}_k\}.$$

The $\mathbb{Q}_k$ elements are used for example for parallel implementations associated with the *deal.II* finite element library [7].

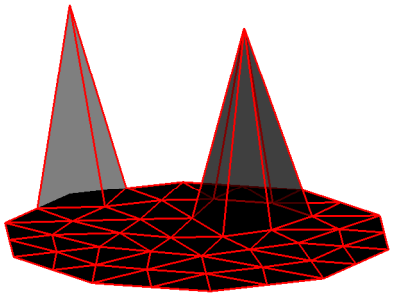


Figure 3.5: Piecewise linear Hat function 2D

Remark 3.2.6. *The shape elements K_i of triangulation of T_h can be seen as an affine transformation from a reference element $\hat{K}$ to K_i.*

$$F_{K_i} : \hat{K} \to K_i \quad F_{K_i}(x) = A_{K_i} x_i + b_{K_i}. \tag{3.3}$$

The affine mappings are also commonly used in many other areas like in the field of computer vision and for the navigation of mobile robots in 2 or 3 dimensions, [74]. For example, for the triangular $\mathbb{P}_k$ finite elements we can take as a reference element

$$\hat{K} = \{(\xi, \eta) \in \mathbb{R}^2, 0 < \xi, \eta < 1, \xi + \eta < 1\} \tag{3.4}$$

and for the rectangular Q_k finite elements we can take as a reference element

$$\hat{K} = \{(\xi, \eta) \in \mathbb{R}^2, -1 < \xi, \eta < 1\}. \tag{3.5}$$

Now we can define the finite element space V_h on which we will approximate our solutions. For the one dimensional case, let $I = [a, b]$ that is partitioned in to n elements where $K_i = [x_{i-1}, x_i]$, with $x_1 < x_2 < \ldots < x_n$. We can define the space of the piecewise continuous polynomials of order k,

$$V_h^k = \{u \in C^0(I) \text{ such that } u|_{K_i} \in \mathbb{P}_k(K_i)\}$$

and

$$V_0^k = V_h^k \cap H_0^1 = span\{\phi_i\}_{i=1}^n.$$

For $k = 1$ we have the space of piecewise linear functions,

$$\phi_i(x) = \begin{cases} (x - x_{i-1})/h_i & \text{if } x_{i-1} \le x \le x_i \\ (x_{i+1} - x)/h_{i+1} & \text{if } x_i \le x \le x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

with $u = \sum_{i=0}^{n} a_i \phi_i$ where $a_i = u(x_i)$. For problems like (2.8) $\Omega \subset \mathbb{R}^d$, where d=2,3.

1. For triangular $\mathbb{P}_k$ finite elements we define,

$$V_0^k = V_h^k \cap H_0^1, \quad where, \; V_h^k = \{u \in C^0(\Omega) \; such \; that \; u|_{K_i} \in \mathbb{P}_k(K_i)\}$$

2. For rectangular $\mathbb{Q}_k$ finite elements we define,

$$V_0^k = V_h^k \cap H_0^1, \quad where, \; V_h^k = \{u \in C^0(\Omega) \; such \; that \; u|_{K_i} \in \mathbb{Q}_k(K_i)\}$$

Remark 3.2.7. *In the finite element context, the weak formulation of problem* (2.10) *reads: find* $u_h \in V_0^k$ *such that*

$$a(u_h, v_h) = (f, v_h) \;\; for \; all \; v_h \in V_h^k. \tag{3.7}$$

Let $\{\phi_i\}_{i=1}^{m_i}$ *be a basis of* V_0^k*, then* $u_h = \sum_{i=1}^{m_i} x_i \phi_i$ *and we we can write our problem in matrix form as*

$$Ax = b, \tag{3.8}$$

where $A_{ij} = a(\phi_i, \phi_j)$, $b_i = (f, \phi)_{L_2}$ *and* $x = [x_1, \dots x_{m_i}]^T$.

3.2.1 Assembly of the Galerkin Systems

For the case of the Poisson problem with Dirichlet or Robin boundary conditions we are interested mainly in the assembly of the mass matrix M and the stiffness matrix A. The mass matrix M is constructed by the L_2 inner product between each pair of the basis functions. On the other hand the stiffness matrix A is constructed by the L_2 inner product between the gradient of each pair of the basis functions.

Hence each element of the mass matrix is given by

$$M_{ij} = \int_\Omega \phi_i \phi_j dx,$$

and for the stiffness matrix A,

$$A_{ij} = \int_\Omega \nabla\phi_i \nabla\phi_j dx.$$

We calculate the mass and stiffness matrix element-wise by the following formulas,

$$\int_\Omega \phi_i \phi_j dx = \sum_{K \in T_h} \int_K \phi_i \phi_j dy dx,$$

$$\int_\Omega \nabla\phi_i \nabla\phi_j dy dx = \sum_{K \in T_h} \int_K \nabla\phi_i \nabla\phi_j dy dx.$$

As an example we use the case of $\mathbb{P}_1$ elements in 2 dimensions. For the reference element $\hat{K}$ in (3.4), the corresponding basis functions are, $\phi_1^*(\hat{x},\hat{y}) = 1-\hat{x}-\hat{y}$, $\phi_2^*(\hat{x},\hat{y}) = \hat{x}$ and $\phi_3^*(\hat{x},\hat{y}) = \hat{y}$.

If we assume that we have the triangle K_i with vertices $(x_1,y_1),(x_2,y_2),(x_3,y_3)$, then we define the affine transformation $F : \hat{K} \to K_i$ as,

$$F(\hat{x}) = \begin{bmatrix} x_2-x_1 & x_3-x_1 \\ y_2-y_1 & y_3-y_1 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.9}$$

Moreover,

$$F^{-1}(x) = \frac{1}{2|K_i|} \begin{bmatrix} y_3-y_1 & -x_3+x_1 \\ -y_2+y_1 & x_2-x_1 \end{bmatrix} \begin{bmatrix} x-x_1 \\ y-y_1 \end{bmatrix}$$

where

$$2|K_i| = det \begin{pmatrix} x_2-x_1 & x_3-x_1 \\ y_2-y_1 & y_3-y_1 \end{pmatrix}$$

We also define the gradients as, $\nabla = [\partial_x, \partial_y]^T$, $\hat{\nabla} = [\partial_{\hat{x}}, \partial_{\hat{y}}]^T$. Hence we can write ϕ_i in the following expression,

$$\phi_i = \phi_i^* \circ F^{-1}$$

Then by change of variables we get that,

$$\int_{K_i} \phi_j \phi_j dy dx = 2 \int_{\hat{K}} \phi_i^* \phi_j^* |K_i| d\hat{y} d\hat{x} \tag{3.10}$$

For the $\mathbb{P}_1$ it's easy to calculate the integrals in (3.10) explicitly, for higher order elements we can use the Gaussian quadrature over triangles. Here we calculate the integrals,

$$\int_0^1 \int_0^{1-x} (1-\hat{x}-\hat{y})^2 d\hat{y} d\hat{x} = \int_0^1 \int_0^{1-x} \hat{x}^2 d\hat{y} d\hat{x} = \int_0^1 \int_0^{1-x} \hat{y}^2 d\hat{y} d\hat{x} = \frac{1}{12}$$

$$\int_0^1 \int_0^{1-x} (1-\hat{x}-\hat{y})\hat{x} d\hat{y} d\hat{x} = \int_0^1 \int_0^{1-x} (1-\hat{x}-\hat{y})\hat{y} d\hat{y} d\hat{x} = \int_0^1 \int_0^{1-x} \hat{x}\hat{y} d\hat{y} d\hat{x} = \frac{1}{24}$$

Hence,

$$M_{K_i} = \frac{2|K_i|}{24} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Then, we can assemble M as,

$$M = \sum_{K_i \in T_h} R_i^T M_{K_i} R_i \tag{3.11}$$

where R_i^T, R_i are the binary, prolongation and restriction operators from the local to

global numbering and vice versa. It is easy to prove that M is symmetric and positive definite.

Remark 3.2.8. *The matrix M is not in general diagonal but this is a property that can achieved by what is called "lumping" of matrix M. The "lummped" mass matrix $\tilde{M}$ can be seen as the result of applying the trapezoidal rule for the numerical integration of $(\phi_i, \phi_j)_{L_2(K_i)} = \int_{K_i} \phi_i \phi_j dx$. Another way to obtain $\tilde{M}$ is by using a higher order quadrature and then by summing the rows of the matrix,*

$$\tilde{M}_{ii} = \sum_{j=1}^{d} M_{ij}$$

and

$$\tilde{M}_{ij} = 0 \;\; for \;\; j \neq i.$$

Definition 3.2.9. *Two symmetric positive definite bilinear forms are called spectrally equivalent on the space V_h^k, if there exists $C_1 > C_0 > 0$ s.t.*

$$C_0 b(u_h, u_h) \leq a(u_h, u_h) \leq C_1 b(u_h, u_h) \; \forall u_h \in V_h. \tag{3.12}$$

Definition 3.2.10. *Let the matrix $E \in \mathbb{R}^{n \times n}$ and I be the identity matrix. Then we define*

$$c_1 \leq E \leq c_2$$

to be equivalent to

$$c_1 v^T I v \leq v^T E v \leq c_2 v^T I v \; for \; all \; v \in R^n.$$

Proposition 3.2.11. *Assume that T_h is quasi-uniform triangulation. The mass matrix M is spectrally equivalent with the identity.*

Proof. The condition number for positive definite matrices is defined as $\kappa(M) = \|M\|\|M^{-1}\| = \frac{\lambda_{max}}{\lambda_{min}}$ and

$$\lambda_{min}(M) = \min \frac{u^T M u}{u^T u} \;\; , \; \lambda_{max}(M) = \max \frac{u^T M u}{u^T u}.$$

From (3.11) we get that M can be written as

$$M = \sum_{K_i \in T_h} R_i^T M_{K_i} R_i.$$

From the quasi-uniformality of the mesh and the fact the each M_{K_i} is proportional to to the area of each element K_i we get that,

$$ch^d \leq M_{K_i} \leq Ch^d$$

which leads to,

$$ch^d \sum_{K_i} u_i^T u_i \leq M \leq Ch^d \sum_{K_i} u_i^T u_i \;\; d = 2, 3.$$

Moreover, we can set $u^T D u = \sum_{K_i} u_i^T u_i$, where $D = diag(d_1, d_2, \ldots, d_n)$ and $d_i \in \mathbb{Z}$ is the multiplicity of each node. Since we have chosen the mesh to be quasi-uniform the multiplicity of each node is bounded. Hence,

$$1 \leq d_i \leq d_{max}.$$

Finally if follows that

$$ch^d u^T u \leq u^T M u \leq Ch^d d_{max} u^T u.$$

□

We continue with the assembly of the stiffness matrix A. In (3.9) we set

$$D_{K_i} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}.$$

Then the inverse of D_{K_i} can be written as

$$D_{K_i}^{-1} = \frac{1}{\det(D_{K_i})} \begin{bmatrix} y_3 - y_1 & -x_3 + x_1 \\ -y_2 + y_1 & x_2 - x_1 \end{bmatrix}.$$

For the case of the stiffness matrix A we have that

$$A_{ij} = \int_{K_i} \nabla\phi_i \nabla\phi_j = |\det(D_{K_i})| \int_{\hat{K}} \hat{\nabla}(\phi_i \circ F)\hat{\nabla}(\phi_j \circ F) =$$

$$|\det(D_{K_i})| \int_{\hat{K}} ((D_{K_i}^{-1})^T \hat{\nabla}\phi_i^*)^T ((D_{K_i}^{-1})^T \hat{\nabla}\phi_j^*).$$

If we calculate the gradient for the basis functions and the corresponding integrals for the $\mathbb{P}_1$ case elements we get that the local stiffness matrix for element K_i is,

$$A_{Ki} = \frac{b_1}{c} \begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{b_3}{c} \begin{bmatrix} 0.5 & 0 & -0.5 \\ 0 & 0 & 0 \\ -0.5 & 0 & 0.5 \end{bmatrix} + \frac{b_2}{c} \begin{bmatrix} 1.0 & -0.5 & -0.5 \\ -0.5 & 0 & 0.5 \\ -0.5 & 0.5 & 0 \end{bmatrix}$$

where, $a_{11} = y_3 - y_1$, $a_{12} = x_1 - x_3$, $a_{21} = y_1 - y_2$, $a_{22} = x_2 - x_1$, $c = det(D^{-1}D^{-^T}) = ((a_{22}a_{11}) - (a_{12}a_{21}))$, $b_1 = a_{11}^2 + a_{12}^2$, $b_2 = a_{11}a_{21} + a_{12}a_{22}$, $b_3 = a_{22}^2 + a_{21}^2$. In Algorithm 3 we present an implementation of the global matrix A, in C++.

Proposition 3.2.12. *[69] Let T_h be a shape regular triangulation, the conforming finite element space $V_h^k(\Omega)$ with $k \geq 1$ and $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, then we have the following well known error estimates. For the energy semi-norm,*

$$|u - u_h|_{H^1(\Omega)} \leq C_1 h^{s-1} |u|_{H^s(\Omega)}, \ d/2 \leq s \leq k+1 \tag{3.13}$$

and if additionally Ω is convex we get the error estimate for the L^2 norm,

$$\|u - u_h\|_{L^2(\Omega)} \leq C_2 h^s |u|_{H^s(\Omega)}. \tag{3.14}$$

From the error estimates (3.13, 3.14), we observe that we can achieve a better approximation of the continuous solution by decreasing the diameter of the elements h, (h method) increasing the order or the polynomial space p, (p method), or both at the same time (h-p methods).

Proposition 3.2.13. *[69] (Inverse Inequalities) Let T_h be a shape regular triangulation and the conforming finite element space $V_h^k(\Omega)$. Let $s \geq t \geq 0$ be two real numbers. Then, for $K \in T_h$, there exist constants C_1, C_2 depending only on s, t, k and the aspect ratio C_K of K which is defined in Definition 3.2.3, such that*

$$|u_h|_{H^s(K)} \leq C_1 h_K^{-(s-t)} |u_h|_{H^t(K)}, \ \forall u_h \in V_h,$$

and if T_h is quasi-uniform

$$|u_h|_{H^s(K)} \leq C_2 h^{-(s-t)} |u_h|_{H^t(K)}, \ \forall u_h \in V_h.$$

Comments on the Implementation

The first step in our finite element code is to provide a seed mesh $\mathcal{T}_0$, of T_0 elements and P_0 nodes, that describes the general geometry of the mesh. In order to provide this original seed mesh we use one of the following methods:

1. The API functions provided by the Triangle 2D Mesh Generator and Delaunay Triangulator [63].

2. Read the mesh from a "*.msh" file that is created by the Gmsh, which is a 2D and 3D finite element mesh generator with built-in pre and post-processing facilities, [32].

3. Read from an "*.m" file that contain in the first line the number of elements and number of nodes followed by the elements and the nodes of the initial mesh.

Then we uniformly refine the mesh, in order to reach the desired level of refinement. In Algorithm 2 we present an implementation of the uniform refinement of the mesh

in C++. In line 12 we preallocate a vector E of size m, where the size m is equal to 3 times the number of triangles of the input mesh. Each element of the vector E, holds an std class member of type **pair** that is a tuple that contains a 2 dimensional vector and an integer. In this vector of **pairs** we store the edges of the triangle elements of the input mesh with an associated integer for each edge.

In line 27 we sort the edges in lexicographical order by using the custom sort function `SortPair` in line 3. We order the edges but the associated integer shows the initial position of each edge in the unsorted E. The array *en* holds the numbering of the edges. Some edges appear twice in the *en* list so we need assure that duplicate nodes are assigned to the same number. Vector x holds a map from the original position of the edges, in the E vector, to the numbering after the sorting.

Finally we create the coordinates of the new nodes from line 37 to 52 and then we use the map x in order to create the triangular element of the mesh. As an output we get the `public` variables of `vector` $<$ `double` $>$, P and T, which hold the elements and the nodes that fully characterise the uniformly refined mesh.

Moreover, we include an implementation of the Global Assembly of the parallel 2D stiffness matrix, Algorithm 3, which is part of the `Galerkin_Assemblance` class. As an input we give the elements T and the nodes P in a vector form and the array $*epart$ that holds the partition number that defines which processor each element belong to. In order to partition the mesh we have created `Partition` class that uses the METIS APIs, [42]. We can partition the mesh in contiguous or non-contiguous subdomains and we start the numbering from zero,

```
int options[METIS_NOPTIONS];
METIS_SetDefaultOptions(options);
options[METIS_OPTION_CONTIG]=1;
options[METIS_OPTION_NUMBERING]=0;
METIS_PartMeshNodal(&nt,&nv,eptr,eind,NULL,NULL,\
&n_partitions,NULL,options,&edgecut,epart,npart);
```

This `Partition` class plays a central role in the implementation, since after the decomposition of the domain it handles all the parallel and sequential parts of the algorithm related to the mesh manipulation.

Since we have decomposed the domain Ω, we check if (`epar[i]` = `rank`), in order to force each processor to add the correct local contributions to the Global Stiffness matrix. The variable "rank" defines the associated number of each process in the global `MPI` communicator `MPI_COMM_WORLD`. Here, we can note that even if the degrees of freedom that correspond to the partition of the parallel Matrix does not exactly coincides with the one that is provided by METIS, PETSC in step,

```
MatAssemblyBegin(AII,MAT_FINAL_ASSEMBLY);
```

```
MatAssemblyEnd(AII,MAT_FINAL_ASSEMBLY);
```

handles any necessary communication between the processors.

Algorithm 2 Uniform Refinement function.

```
/* Uniform refinment function, part of the Refinement class */

bool SortPair(const pair<vector<int>, int> &i, const pair<vector<int>, int>  &j ) { return i.first < j.first; }


int Refinement::uniform(){

  int m,n;
  m=T.size();
  n=P.size();
  std::vector<int> myVec(2,0);
  vector<pair<vector<int>, int> > E (m, std::make_pair(vector<int>(2), 0));
  pair<vector<int>,int> tmpPair;

  int a,b;

  for (int i = 0; i < m/3; ++i) {
    for (int j = 0; j < 3; ++j) {
      a = T[3 * i + j];
      b = T[3 * i + ((j + 1) % 3)];
      myVec[0] = std::min(a,b);
      myVec[1] = std::max(a,b);
      tmpPair=make_pair(myVec,3*i+j);
      E[3*i+j]=tmpPair;
    }
  }
  std::sort(E.begin(),E.end(),SortPair);
  int *en = new int [m]; //number the initial edges
  en[0] = 0;
  int *x = new int [m];
  x[E[0].second]=en[0];
  for (int i = 1; i < m; i++) {
    en[i] = en[i - 1] + ((E[i-1].first <E[i].first) || (E[i-1].first >E[i].first));
    x[E[i].second]=en[i];
  }

  int c;
  a = E[0].first[0];
  b = E[0].first[1];
  c = en[0];
  P.push_back( 0.5 * (P[2 * a] + P[2 * b]));
  P.push_back( 0.5 * (P[2 * a + 1] + P[2 * b + 1]));

  for (int i = 1; i < m; i++) { //Creating the part of matrix P with the new vertices.
    a = E[i].first[0];
    b = E[i].first[1];
    c = en[i];
    if(c!=en[i-1]){
      P.push_back( 0.5 * (P[2 * a] + P[2 * b]));
      P.push_back( 0.5 * (P[2 * a + 1] + P[2 * b + 1]));
    }
  }

  int n_temp;
  vector<int> T0(T);
  T.clear();
  int n0=n/2,m0=m/3;
  for (int i = 0; i < m0; i++) {
    a = x[3 * i];
    b = x[3 * i + 1];
    c = x[3 * i + 2];


    T.push_back(T0[3 * i]);
    T.push_back( a + n0);
    T.push_back( c + n0);


    T.push_back(T0[3 * i + 1]);
    T.push_back(b + n0);
    T.push_back(a + n0);


    T.push_back( T0[3 * i + 2]);
    T.push_back(c + n0);
    T.push_back(b + n0);

    T.push_back(a + n0);
    T.push_back( b + n0);
    T.push_back( c + n0);

  }
  T0.clear();
  delete [] en;
  delete [] x;

  return 0;
}
```

Algorithm 3 Assembly of Global stiffness matrix.

```
int Galerkin_Assemblance::Siffness_Matrix2DP1Parallel(std::vector<int> &T,std::vector<double> &P,int *epart){

  int size_mat;
  PetscInt Istart,Iend;
  PetscScalar ftemp;
  int size_boundary;
  size_boundary=0;
  int temp;
  temp=P.size()/2;
  int m=T.size()/3;
  std::vector<int> renumber(temp,-1);
  int rank;
  MPI_Comm_rank(PETSC_COMM_WORLD,&rank);
  int count=0;
  for(int i=0;i<temp;i++){
    if(boundary[i]==0){
      renumber[i]=count;
      count=count+1;
    }else {size_boundary=size_boundary+1;}
  }

  PetscInt local_size;
  size_mat=temp-size_boundary;
  PetscScalar Val;
  VecCreateMPI(PETSC_COMM_WORLD,PETSC_DECIDE ,size_mat,&RHSI);
  VecSetOption(RHSI, VEC_IGNORE_NEGATIVE_INDICES,PETSC_TRUE);
  VecZeroEntries(RHSI);
  VecGetLocalSize(RHSI,&local_size);

  MatCreate(PETSC_COMM_WORLD,&AII);
  MatSetType(AII,MATAIJ);
  MatSetSizes(AII,local_size,local_size,size_mat,size_mat);

  PetscInt *o_nnz=new PetscInt[size_mat];
  PetscInt *d_nnz=new PetscInt[size_mat];
  for(int i=0;i<size_mat;i++) d_nnz[i]=static_cast<PetscInt>(ceil(nnz[i]));
  for(int i=0;i<size_mat;i++) o_nnz[i]=static_cast<PetscInt>(ceil(nnz[i]));
  MatSeqAIJSetPreallocation(AII,NULL,d_nnz);
  MatMPIAIJSetPreallocation(AII,NULL,d_nnz,NULL,o_nnz);
  MatSetFromOptions(AII);
  MatGetOwnershipRange(AII,&Istart,&Iend);

  for(int i=0;i<m;i++){
    if(epart[i]==rank){
      LocalP1MASS(3*i,T,P);
      for(int j=0;j<9;j++){
        Val=Aloc[j];
        int ind1=renumber[Localrows[j]];
        int ind2=renumber[Localcolumn[j]];
        MatSetValues(AII,1,&ind1,1,&ind2,&Val,ADD_VALUES);
      }
      for(int j=0;j<3;j++){
        int ind2=renumber[T[3*i+j]];
        ftemp=RHSloc[j];
        VecSetValues(RHSI,1,&ind2,&ftemp,ADD_VALUES);
      }
    }
  }
  MatAssemblyBegin(AII,MAT_FINAL_ASSEMBLY);
  MatAssemblyEnd(AII,MAT_FINAL_ASSEMBLY);

  VecAssemblyBegin(RHSI);
  VecAssemblyEnd(RHSI);
  /*if(rank==0) VecView(RHSI,PETSC_VIEWER_STDOUT_SELF);
    MatView(AII,PETSC_VIEWER_ASCII_MATLAB);
    PetscViewer    viewer;
    PetscViewerASCIIOpen(PETSC_COMM_WORLD, "Dmat.m", &viewer);
    PetscViewerSetFormat( viewer, PETSC_VIEWER_ASCII_MATLAB);
    MatView(AII,viewer);

    VecView(RHSI,PETSC_VIEWER_STDOUT_WORLD);*/

  return 0;

}
```

Chapter 4

Krylov Subspace Methods

4.1 Iterative methods and Projection methods

The ability to solve efficiently and accurately large sparse systems is crucial in many different steps of our methods. After the discretisation of the problem by using a suitable method, like the Finite Element Method, we obtain three different types of matrices. The first type is obtained from the discretisation of the local Robin subproblems and the matrices are symmetric and positive definite. The second type are indefinite matrices related to the symmetric 2-Lagrange multiplier method (1.3). Finally, the third type are non-symmetric matrices with complex eigenvalues which have positive real part, related to the non-symmetric 2-Lagrange multiplier method (1.4).

In order to solve efficiently these large sparse systems, we use Krylov Subspace methods like MINRES [53] and GMRES [59]. Additionally we use suitable preconditioning techniques for each problem. Hence some background on Krylov subspace methods is necessary, in order to understand the convergence and scaling properties of our methods. In this chapter we will make a small review of Krylov Subspace methods with an emphasis on the GMRES method.

Let A be an $n \times n$ large sparse matrix with the real vector b of size n. We seek the solution x in

$$Ax = b. \tag{4.1}$$

The basic idea of iterative methods is to create an approximation x_k of the solution of the linear system. This approximation improves in each step of the algorithm. One way to derive such methods is to start by splitting matrix A in two distinct matrices M, N, where $A = M - N$. Then, the successive iterations are given by

$$Mx_{k+1} = Nx_k + b. \tag{4.2}$$

We can rewrite this as

$$x_{k+1} = x_k + M^{-1}(b - Ax_k), \tag{4.3}$$

or

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b. \tag{4.4}$$

As in [61] let,

$$T = M^{-1}N = I - M^{-1}A \;\; and \;\; c = M^{-1}b.$$

Then the iteration $x_{k+1} = Tx_k + c$ can be seen as an iteration method to solving the system

$$(I - T)x = c. \tag{4.5}$$

After substituting T and c in (4.5) the system can be rewritten as,

$$M^{-1}Ax = M^{-1}b. \tag{4.6}$$

The system (4.6) has the same solution as (4.1) and is called the **preconditioned** system.

The most obvious splitting, is probably the one for which

$$M = I \;\text{ and }\; N = I - A. \tag{4.7}$$

In (4.2) we get

$$x_{k+1} = (I - A)x_k + b = x_k + b - Ax_k =$$

$$x_k + (b - Ax_k) = x_k + r_k \tag{4.8}$$

where $r_k = b - Ax_k$ is called the residual at step k. From [72] we get that if we multiply (4.8) by $-A$ and add b we have,

$$b - Ax_{k+1} = b - Ax_k - Ar_k$$

or equivalently,

$$r_{k+1} = (I - A)r_k = (I - A)^{k+1}r_0 = P_{k+1}(A)r_0. \tag{4.9}$$

We observe that P_{k+1} is a polynomial of degree $k + 1$, with $P_{k+1}(0) = 1$.

If we assume that A has n eigenvectors v_j and the corresponding eigenvalues are λ_j, then we can write r_0 as a linear combination of the eigenvectors of A,

$$r_0 = \sum_{j=1}^{n} a_j v_j. \tag{4.10}$$

Moreover, $P_k(A)v_i = (I - A)^k v_i = (I - A)^{n-1}(v_i - \lambda_i v_i) = \ldots = (1 - \lambda_i)^k v_i$. Hence we can rewrite (4.9) as

$$r_k = P_k(A)r_0 = \sum_{j=1}^{n} a_j P_k(\lambda_j)v_j. \tag{4.11}$$

Now, from (4.11), it is clear that the convergence of the iteration algorithms to the solution of (4.1), depends on how well the roots of the polynomial P_k approximate the spectrum of A.

We assume for simplicity that $x_0 = 0$, then from (4.8) we get that

$$x_1 = r_0$$

$$x_2 = x_1 + r_1 = r_0 + r_1$$

and if we continue recursively we get that,

$$x_{j+1} = r_0 + r_1 + \ldots + r_j$$

$$= \sum_{k=0}^{j} (I - A)^k r_0 \in span\{r_0, Ar_0, \ldots, A^j r_0\} = K^{j+1}$$

Definition 4.1.1. *[72] The n-dimensional space spanned by $b, Ab, A^2b, \ldots, A^{n-1}b$, is called the n-dimensional Krylov subspace, generated by A and b and we denote it by $\mathcal{K}^n(A; b) \subset \mathbb{R}^n$, $K^n(A; b) = span\{b, Ab, A^2b, \ldots, A^{n-1}b\}$.*

From definition (4.1.1) we get that in the *kth* iteration of a Krylov method, the approximate solution will have the form

$$x_k = S_k(A)r_0,$$

where S_k is and arbitrary polynomial of degree k. Then we can write r_{k+1} as,

$$r_{k+1} = b - Ax_{k+1} = (I - AS_{k+1}(A))r_0 = J_{k+1}(A)r_0, \tag{4.12}$$

with $J_{k+1}(0) = 1$.

In the next simple 2×2 example we observe that the solution of the system $Ax = b$ can be written as the linear combination of b and Ab. These two vectors span $\mathcal{K}_2(A, b)$.

Example 2. *Consider the Linear system Ax=b. Where,*

$$A = \begin{bmatrix} 0 & 3 \\ -2 & 5 \end{bmatrix} \quad and \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

We find that the characteristic polynomial of A is,

$$p(\lambda) = \lambda^2 - 5\lambda + 6$$

From the Cayley-Hamilton theorem, [15], we know that $p(A) = 0$. *That is,*

$$A^2 - 5A + 6I = 0$$

and hence

$$A^{-1} = \frac{5I}{6} - \frac{A}{6}$$

Finally,

$$x = A^{-1}b = \Big(\frac{5I}{6} - \frac{A}{6}\Big)b = \frac{5b}{6} - \frac{Ab}{6}.$$

Hence $x \in \mathcal{K}^2(A; b)$.

A drawback that we face when we use $\mathcal{K}^n(A; r_0) = \{r_0, Ar_0, A^2r_0, \ldots, A^nr_0\}$ without any modification, is that in many cases A^kr_0 tends to the major dominant eigenvector of A. This means that the elements of the basis tend to be parallel, something that in practice leads to bad approximations and unstable methods.

For example, in the case when A is diagonalisable with eigenvalues

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$$

we have that $\frac{|\lambda_n|}{|\lambda_1|} < 1$. If we use (4.10), we get that

$$A^kr_0 = A^k\sum_{j=1}^{n} a_jv_j = \sum_{j=1}^{n} a_jA^kv_j = \sum_{j=1}^{n} a_j\lambda_j^kv_j = a_1\lambda_1^kv_1 + \lambda_1^k(\sum_{j=2}^{n} a_j(\frac{\lambda_j}{\lambda_1})^kv_j).$$

For $k \to \infty$, A^kr_0 tends to the dominant eigenvector. This "bad" behaviour of the basis leads to poor approximations of the original solution of the system.

One way to solve this problem, is to construct a stable orthonormal basis of $\mathcal{K}^n$. This can be done for example by using Arnoldi's orthogonlisation method [3]. Arnoldi's method [3] starts with the initial vector $q_1 = r_0/||r_0||_2$, then in each step we compute $z = Aq_j$ and make it orthonormal to $q_{j-1}, q_{j-2} \ldots, q_0$. The procedure continues in the same manner until $j = n$, except in the case when $||z||$ becomes zero, where the algorithm stops. The orthogonalisation can also be done with some other variant of the Gram-Schmidt method.

Here we will state three key propositions for the Arnoldi method, their proofs can be found in [61].

Proposition 4.1.2. *Assume that Algorithm* 4 *does not stop before the n-th step. Then*

Algorithm 4 Arnoldi's Orthogonalisation Procedure

```
Choose a vector v and set q_1 = v/||v||
for j = 1, 2, ..., n do
    Compute z_j = A q_j
    for i = 1, 2, ..., j do
        H_ij = q_i^T z_j
        z_j = z_j − H_ij q_i
    end for
    H_{j+1,j} = ||z_j||
    if H_{j+1,j} = 0 then
        quit
    end if
    q_{j+1} = z_j / H_{j+1,j}
end for
```

the vectors $v_1, v_2, v_3, \ldots, v_n$ form an orthogonal basis of the Krylov subspace,

$$K_m = span\{v_1, Av_1, \ldots, A^{n-1}v_1\}.$$

Proposition 4.1.3. *Denote by V_m the $n \times m$ matrix with column vectors $v_1, \ldots, v_m$ and by $\tilde{H}_m$, the $(m+1) \times m$ Hessenberg matrix, with nonzero entries H_{ij} as defined in Algorithm 4. The matrix H_m is obtained by deleting the last row of $\tilde{H}_m$. The following relations hold:*

$$AV_m = V_m H_m + z_m e_m^T = V_{m+1}\tilde{H}_m, \tag{4.13}$$

$$V_m^T A V_m = H_m. \tag{4.14}$$

4.1.1 Projection Methods

The basic iterative methods for solving the system (4.1), such as Richardson , Gauss-Seidel and SOR are in general "cheap" in terms of computation and memory requirements, but with the disadvantage that the have slow to converge rates [45]. Nevertheless, other faster iterative methods have been designed, based on the requirement that the residual $r = b - Ax$ is orthogonal to an appropriate subspace. The Krylov subspace methods and especially the GMRES method, which is of great importance, can be categorized and analysed as projection methods.

Definition 4.1.4. *[61] Let the spaces $\mathfrak{K}$,$\mathfrak{L} \subset \mathbb{R}^n$. A projection technique onto the subspace $\mathfrak{K}$ and orthogonal to $\mathfrak{L}$ is a process which finds an approximate solution $\hat{x}$ to* (4.1) *by imposing the conditions that $\hat{x}$ belongs to $\mathfrak{K}$ and that the new residual vector should be orthogonal to $\mathfrak{L}$.*

$$b - A\hat{x} \perp \mathfrak{L} \tag{4.15}$$

The subspace $\mathfrak{K}$ is called the trial or search space and the subspace $\mathfrak{L}$ is called a

test or restriction space. Moreover a projection method is called orthogonal, when $\mathfrak{L}$ and $\mathfrak{K}$ are identical and oblique if $\mathfrak{K}$ and $\mathfrak{L}$ are different.

Remark 4.1.5. *We can write the definition in a matrix form as: find* $\hat{x} = x_0 + Vy$ *with the restriction* $W^T AVy = W^T b$. *The matrices* V, W *hold the basis for* $\mathfrak{K}$ *and* $\mathfrak{L}$, *respectively. If* $W^T AV$ *is invertible, the we can write* $\hat{x}$ *as*

$$\hat{x} = x_0 + V(W^T AV)^{-1} W^T b \tag{4.16}$$

4.2 The Generalized Minimal residual method

We use (4.1.5) to derive the GMRES oblique projection method where $\mathfrak{K} = \mathcal{K}_m$ and $\mathfrak{L} = A\mathcal{K}_m$. We assume that $\hat{x} \in \mathcal{K}_m$ and also without loss of generality that $x_0 = 0$. Then, V_m holds the basis of $\mathfrak{K}$ and $\hat{x}$ can be written as $\hat{x} = V_m y$. In order for the orthogonality condition to hold we want

$$(AV_m)^T AV_m y = (AV_m)^T b,$$

which is a least-squares problem equivalent to minimising

$$||b - AV_m y||_2.$$

From Proposition 4.1.3 we get that the orthogonal basis for the Krylov subspace K_m, with basis vectors $v_1, \ldots, v_m$ leads to the following expression,

$$AV_m = V_{m+1}\tilde{H}_m.$$

Let $\mu = ||b - Ax_0||_2$, we can derive that

$$||b - A\hat{x}||_2 = ||b - AV_m y||_2 \tag{4.17}$$

$$= ||\mu v_1 - V_{m+1}\tilde{H}_m y||_2 = ||\mu V_{m+1} e_1 - V_{m+1}\tilde{H}_m y||_2 \tag{4.18}$$

and since V_{m+1} is unitary we get,

$$||b - A\hat{x}||_2 = ||\mu e_1 - \tilde{H}_m y||_2. \tag{4.19}$$

The norm (4.19) can be minimised by solving the least square problem for the matrix $\tilde{H}_m y$ and with right hand side $\mu {e_1}^1$.

The GMRES approximation [61] is the unique vector $\hat{x} \in \mathcal{K}$ that minimises (4.19)

[1]The minimiser is inexpensive to compute due to the fact that requires the solution of an $(m + 1) \times m$ least square problem.

This approximation can obtained by $\hat{x} = V_m y_m$ if $x_0 = 0$ or $\hat{x} = x_0 + V_m y_m$ if $x_0 \neq 0$. The methods can be written in the following form,

$$x_m = x_0 + V_m y_m, \text{where}$$

$$y_m = \underset{y}{\operatorname{argmin}} ||\mu e_1 - \tilde{H}_m y||_2.$$

Algorithm 5 GMRES [59]

1: Start: $r_0 = b - Ax_0$ and $\mu = ||r_0||$.
2: **for** $m = 1, 2, \ldots, m_{max}$ **do**
3: Compute step m of Arnoldi algorithm 4 for A and $v = r_0/||r_0||$.
4: Incrementally compute the QR factorisation of the $(m+1) \times m$ Hessenberg matrix $\tilde{H}_m = \{H_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$ by given rotations.
5: Test if $||r_m|| = ||\mu e_1 - \tilde{H}_m y|| = \mu |e_{m+1}^T Q_{m+1} e_1| < tol$, where tol is the tolerance parameter provided by the user. If the desirable tolerance is achieved compute y_m and return $x_m = x_0 + V_m y_m$.
6: **end for**

There are a lot of different variations of GMRES algorithm, [59], which include restarting or different methods in implementing the orthogonalisation step. Two examples are the truncated GMRES and the Flexible GMRES (FGMRES) [58].

When we increase the size of m with full orthogonalisation of K_m, the memory requirements increase, making the method impractical. Thus it is common to use the GMRES variant with restart, which means that we do not perform a full orthogonalisation, but we restart the orthogonalization procedure after every l steps of the algorithm. Here l is defined by the user or is set by default in the solver. For example PETSc `KSPGMRES` solver has a default restart value equal to 30.

Algorithm 6 GMRES(k), with restart. [59]

1: Start: $r_0 = b - Ax_0$ and $\mu = ||r_0||$.
2: **for** $m = 1, 2, \ldots, k$ **do**
3: Compute step m of Arnoldi algorithm 4 for A and $v = r_0/||r_0||$.
4: Incrementally compute the QR factorisation of the $(m+1) \times m$ Hessenberg matrix $\tilde{H}_m = \{H_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$ by given rotations.
5: Test if $||r_m|| = ||\mu e_1 - \tilde{H}_m y|| = \mu |e_{m+1}^T Q_{m+1} e_1| < tol$, where tol is the tolerance parameter provided by the user. If the desirable tolerance is achieved compute y_m and return $x_m = x_0 + V_m y_m$.
6: **end for**
7: Restart: Compute $||r_k|| = ||\mu e_1 - \tilde{H}_k y|| = \mu |e_{k+1}^T Q_{k+1} e_1|$
8: **if** tolerance is satisfied **then** return $x_k = x_0 + V_k y_k$.
9: **else** $x_0 := x_k$, $v := r_k/||r_k||$, go to 2.
10: **end if**

The Krylov subspace methods and in particular, the GMRES method, have proved very successful, especially since the mid 70's, [73], when they started to be used along

with effective preconditioners which drastically reduced the number of iterations for large problems. In the case of a modified Krylov subspace method we solve a modified system.

We define the left preconditioned system,

$$M^{-1}Ax = M^{-1}b, \tag{4.20}$$

and the right preconditioned system,

$$AM^{-1}u = b, \text{ where } u = Mx. \tag{4.21}$$

In the case of Flexible GMRES, Algorithm 8, we can allow the preconditioner M to vary between each outer iteration step. For the case of a left preconditioner for GMRES, leads to the preconditioned Krylov subspace, $span\{r_0, M^{-1}Ar_0, \ldots, (M^{-1}A)^{m-1}r_0\}$.

Moreover we can split matrix A as $A = D + L + L^T$ where D is a diagonal and F is a lower diagonal matrix. Then we can define some simple and well studied preconditioners like,

1. Jacobi $M = D$,
2. Gauss-Seidel $M = D + L$,
3. SOR $M = \frac{1}{\omega}(D + \omega L)$.

In Figure 4.2 we include some parallel experiments of solving the Poisson problem on a rectangular domain with a circular hole and zero boundary conditions, see Figure 4.1. In order to solve the discrete system we use the parallel variant of GMRES, `KSPGMRES` provided by the PETSc library, with restart 500, absolute tolerance 10^{-7} and relative tolerance 10^{-6}. For the experiments we used 8 cores and we solved the system without a preconditioned M, `PCNONE`, Jacobi `PCJACOBI`, Gauss-Seidel `PCSOR` $\omega = 1$ and SOR `PCSOR` with $\omega = 1.1$. Moreover we tried the Additive Schwarz `PCASM` with two different levels of overlap and 8 subdomains. We used `PCASM` of type `PC_ASM_BASIC` which means full interpolation and restriction for the local subproblems. In order to create the overlap we used the Algorithm 7 which is implemented in C++.

Remark 4.2.1. *Assume that we have a varying right preconditioner M_j. When we apply FGMRES Algorithm* 8 *on our problem we get*

$$z_j = M_j^{-1}v_j$$

and finally we can calculate x_m as,

$$x_m = x_0 + Z_m y_m.$$

Algorithm 7 Algorithm that creates the overlap that used on experiments in figure 4.2.

```
1  //Create partitioning for 2D mesh, with minor changes should work also for the 3D case//
2  //input: Elements, size of overlap//
3  // Anastasios Karangelis October 2014
4  int Partition::create_overlap(const std::vector<int> &T0,const int level_overlap,const std::vector<int> &boundary){
5
6    int rank;
7    int size_of_local_mesh;
8    MPI_Comm_rank(PETSC_COMM_WORLD,&rank);
9    int Elements_on_local_mesh;
10   int temp;
11   std::set<it64> myset;
12   std::set<it64>::iterator it;
13   std::unordered_map<it64,it64> edge_triangle;
14   std::vector<PetscInt> myvector(3);
15   PetscInt number_of_triangles;
16   number_of_triangles=T0.size()/3;
17   std::vector<PetscInt> overlap_epart;
18
19   //We create an initial the initial set with the overlap indices and
20   //we set the initial set of edges//
21   overlap_npart.clear();
22   for(int i=0;i<number_of_triangles;i++){
23     overlap_epart.push_back(epart[i]);
24     overlap_npart.push_back(npart[i]);
25   }
26   for(int j=0;j<level_overlap;j++){
27     edge_triangle.clear();
28     myset.clear();
29     for(int i=0;i<number_of_triangles;i++){
30       myvector[0]=T0[3*i];myvector[1]=T0[3*i+1];myvector[2]=T0[3*i+2];
31       std::sort(myvector.begin(), myvector.end());
32       if(rank==overlap_epart[i]){
33         overlap_indices.insert(myvector[0]);
34         overlap_indices.insert(myvector[1]);
35         overlap_indices.insert(myvector[2]);
36         myset.insert((myvector[0] * pow(10,static_cast<int>(log10(myvector[1])+1)) + myvector[1]));
37         myset.insert((myvector[0] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]));
38         myset.insert((myvector[1] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]));
39
40       }else{
41         auto p1 = std::make_pair((myvector[0] * pow(10,static_cast<int>(log10(myvector[1])+1)) + myvector[1]),i);
42         auto p2 = std::make_pair((myvector[0] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]),i);
43         auto p3 = std::make_pair((myvector[1] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]),i);
44         edge_triangle.insert (p1);
45         edge_triangle.insert (p2);
46         edge_triangle.insert (p3);
47       }
48     }
49     for (auto& x: myset){
50       if(edge_triangle.find (x)!=edge_triangle.end()){
51         temp=edge_triangle.at(x);
52         myvector[0]=T0[3*temp];myvector[1]=T0[3*temp+1];myvector[2]=T0[3*temp+2];
53         std::sort(myvector.begin(), myvector.end());
54         myset.insert((myvector[0] * pow(10,static_cast<int>(log10(myvector[1])+1)) + myvector[1]));
55         myset.insert((myvector[0] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]));
56         myset.insert((myvector[1] * pow(10,static_cast<int>(log10(myvector[2])+1)) + myvector[2]));
57         overlap_epart[temp]=rank;
58         overlap_indices.insert(T0[3*temp]);
59         overlap_indices.insert(T0[3*temp+1]);
60         overlap_indices.insert(T0[3*temp+2]);
61       }
62     }
63   }
64
65   int count=0;
66   std::vector<int> temp_vec;
67   temp_vec.clear();
68
69   for (auto& x: boundary)
70     if(!x){
71       temp_vec.push_back(count);
72       count=count+1;
73     }else{
74       temp_vec.push_back(-1);
75     }
76   renumbered_overalp_indices.clear();
77   for (auto& x: overlap_indices)
78     if(boundary[x]==0){
79       renumbered_overalp_indices.push_back(temp_vec[x]);
80     }
81
82   return 0;
83
84 }
```

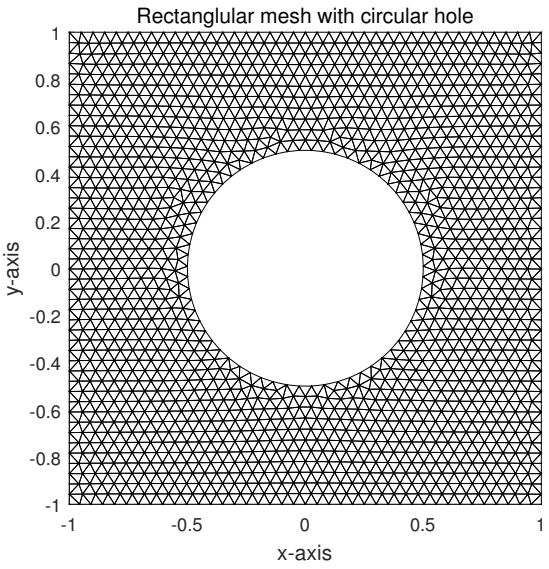


Figure 4.1: Rectangular domain with a circular hole used for the experiments in Figure 4.2.

Since the preconditioner varies we have to hold the sequence of vectors z_j in a matrix form as

$$Z_m = [z_1, z_2, \ldots, z_m].$$

Hence for the FGMRES algorithm we have that

$$AZ_m = V_{m+1}\tilde{H}_m.$$

Moreover, as we have noticed from our large scale numerical experiments in [40] on problem (2.8)*, FGMRES algorithm can lead to better numerical results in the case when we calculated the preconditioner M^{-1} by an inexact solver like CG. Additionally we have observed that the extra memory cost is manageable since we just have to reserve memory for the matrix containing the z_j vectors, Z_m.*

4.3 Convergence of GMRES

The convergence analysis in this section is based on [71], [22] and [46]. The GMRES approximation can be reduced to a polynomial similar to (4.12) with the advantage that the coefficients of the polynomial are optimal, since they are derived from the least squares problem (4.19) . Hence, we have that at step k, $x_k = P_k(A)r_0$ and the corresponding residual is $r_k = (I - AS_k(A))r_0 = J_k(A)r_0$ where $J_k(z) = 1 - zq(z)$ of degree smaller or equal to k and $J_k(0) = 1$.

Hence, the GMRES approximation problem can be seen as a problem to find a

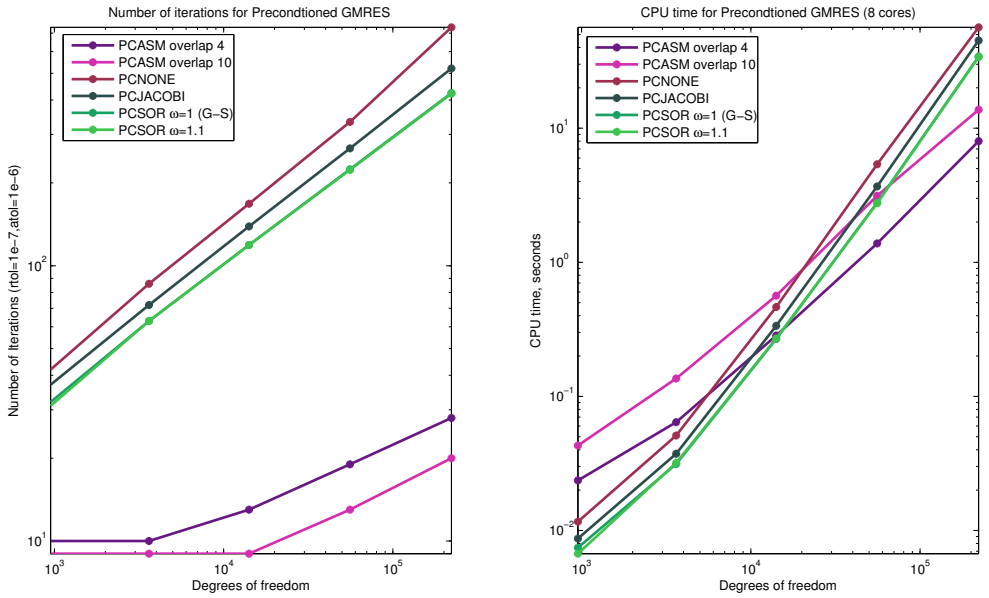


Figure 4.2: Solving the Poisson problem in a rectangular domain with a circular hole, by using the PETSc library.

polynomial

$$J_k \in P_k = \{ \text{ polynomials } p \text{ of degree } \leq k \text{ with } p(0) = 1\}.$$

such that the norm

$$||J_k(A)r_0||_2$$

is minimised. From now on in order to simplify the notation we set $||\cdot|| = ||\cdot||_2$ The crucial question that we need to ask is: How many steps are needed in order that $|r_n||/||b||$ reaches a satisfactory level of accuracy? Here we can start with some very interesting observations,

1. The value of $||r_k||$ is non increasing as k gets larger since the space $\mathcal{K}_k$ increases by one dimension in each step and hence the approximation improves gradually.

2. In at most n steps the algorithm should converge to the real solution since the Krylov space will have the same dimension as the actual system. In this sense, GMRES can be seen as an exact solver. When $\mathcal{K}_n$ has the same dimension as the system then we recover the exact solution of the problem.

From the inequality

$$||r_k|| = ||J_k(A)r_0|| \leq ||J_k(A)||||r_0||,$$

Algorithm 8 FGMRES [59]

1: Start: $r_0 = b - Ax_0$ and $\mu = ||r_0||$.
2: **for** $m = 1, 2, \ldots, m_{max}$ **do**
3: Compute step , of Arnoldi algorithm 4 for A and $v_1 = M_1 r_0/||r_0||$ with varying right preconditioner M_m and $z_m = M_m^{-1} v_m$.
4: Incrementally compute the QR factorisation of the $(m+1) \times m$ Hessenberg matrix $\tilde{H}_m = \{H_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$ by given rotations.
5: Test if $||r_m|| = ||\mu e_1 - \tilde{H}_m y|| = \mu |e_{m+1}^T Q_{m+1} e_1| < tol$, where tol is the tolerance parameter provided by the user. If the desirable tolerance is achieved compute y_m and return $x_m = x_0 + Z_m y_m$.
6: **end for**
7: Restart: Compute $||r_k|| = ||\mu e_1 - \tilde{H}_k y|| = \mu |e_{k+1}^T Q_{k+1} e_1|$
8: **if** tolerance is satisfied **then** return $x_k = x_0 + Z_k y_k$.
9: **else** $x_0 := x_k$, $v := r_k/||r_k||$, go to 2.
10: **end if**

if we divide both sides by $||r_0||$ we see that we can bound $\frac{||r_n||}{||r_0||}$ from $||J_k(A)||$,

$$\frac{||r_k||}{||r_0||} \leq \inf_{J_k \in P_k} ||J_k(A)||. \tag{4.22}$$

So the convergence rate depends on how small the value of $||J_k(A)||_2$, for polynomial $J_k(A)$ can be, for a given matrix A and integer k.

If we assume that A is non-singular and we use the diagonalisation of A, then A can be written as $A = V \Lambda V^{-1}$, where Λ is diagonal.

Theorem 4.3.1. *[71], [46], Let $\sigma(A)$ be spectrum of A, assume A is diagonalisable and define*

$$||J||_{\Lambda(A)} = \sup_{\lambda_i \in \sigma(A)} \{J(\lambda_i)\}.$$

At step k of the GMRES iteration, the residual r_k satisfies

$$\frac{||r_k||}{||r_0||} \leq \inf_{J_k \in P_k} ||J_k(A)|| \leq \kappa(V) \inf_{J_k \in P_k} ||J_k||_{\Lambda(A)}. \tag{4.23}$$

where κ is the 2-norm condition number of V

Proof.

$$||r_n|| = \inf_{J_K \in P_k} ||J_k(A) r_0|| \leq ||V||\,||V^{-1}||\,||r_0|| \inf_{J_K \in P_k} ||J_k||_{\Lambda(A)}.$$
$$\leq \kappa(V) \inf_{J_K \in P_k} ||J_k||_{\Lambda(A)} ||r_0||.$$

□

If A is a normal matrix then it can be written as $A = U \Lambda U^T$ where U is unitary matrix with columns the eigenvectors of A and where Λ is a diagonal matrix with the

eigenvalues of A on the diagonal. Hence when A is normal we have,

$$\frac{||r_k||}{||r_0||} \leq \inf_{J_k \in P_k} ||J_k||_{\Lambda(A)}. \tag{4.24}$$

4.3.1 Convergence of GMRES, connection with potential theory

In [22] the authors manage to connect the convergence of the GMRES method with potential theory. They do that by applying conformal mappings from a compact approximation of the spectrum of the operator to the exterior of the unit disk. They provide estimates for the convergence of GMRES, derived in a very elegant way. Although the reader might be more familiar with convergence proofs related to Chebyshev polynomials, we think that it could be beneficial to see the convergence thought this different perspective. Thus in this section we briefly present some results from [22].

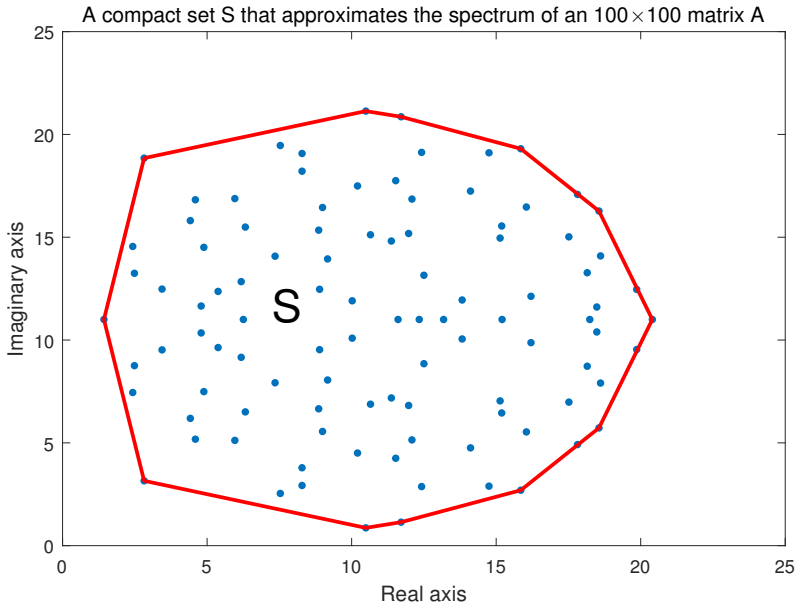


Figure 4.3: A compact set $S \subset \mathbb{C}$ that approximates the spectrum of $\sigma(A)$, $A \in \mathbb{R}^{100\times100}$.

Let $S \subset \mathbb{C}$ be a compact set that approximates the spectrum $\sigma(A)$ of A with $0 \notin S$. Then our main question is to find how small can $J(z) \in P_k$ be on S. Let us define the norm,

$$||J||_S = \sup_{z \in S} |J(z)|. \tag{4.25}$$

Then, we want to find the minimum of the quantities,

$$E_k(S) = \min_{J \in P_k} ||J||_S \quad \text{for} \quad k = 1, 2, \ldots. \tag{4.26}$$

The sequence $\{E_k(S)\}$ decreases geometrically with k at some rate

$$\rho = \lim_{k\to\infty} (E_k(S))^{1/k} < 1 \tag{4.27}$$

from [22]. The limiting value ρ, is called the estimated asymptotic convergence factor and always exists. Unless S completely surrounds the origin the value of ρ is smaller than one.

As we can see from (4.23) the convergence of the Krylov space iteration improves depending on how well the polynomials $J(z)$ approximate the spectrum of the matrix. Moreover, from [22] we get that in many cases,

$$\frac{||r_k||}{||r_0||} \approx \rho^k \tag{4.28}$$

is a reasonable approximation.

We assume that S consists of a finite collection of simply connected components with a piecewise smooth boundary. In order to make the connection with potential theory, firstly we consider the monic polynomial

$$p(z) = \prod_{k=1}^{N}(z - z_k) \tag{4.29}$$

where z_k are complex roots counted with multiplicity. Hence we get that,

$$|p(z)| = \prod_{k=1}^{N} |(z - z_k)| \tag{4.30}$$

and we want to minimise $|p(z)|/|p(0)|$, or equivalently to minimize

$$\log|p(z)| - \log|p(0)| = \sum_{k=1}^{N} \log(\Big|\frac{z - z_k}{z_k}\Big|). \tag{4.31}$$

The minimisation of (4.31) is difficult and hence we consider the case where $N \to \infty$. If we scale by dividing by N (4.31) we get,

$$\phi(z) = N^{-1} \sum_{k=1}^{N} (\log(|(z - z_k)|) + C \tag{4.32}$$

This function is harmonic in the complex plane except at points $\{z_k\}$.

We can interpret (4.32) as an electrostatic potential [22] : $\phi(z)$ is the potential corresponding to point charges at $\{z_k\}$ each of strength $-N^{-1}$. As $N \to \infty$ we can imagine that the negative unit charge is distributed in a continuous fashion in the complex plane.

Next, we define the set $S_0 = S \setminus \{isolated\ points\}$ and we want to minimise $max_{z\in S_0}\phi(z) - \phi(0)$. Due to the properties of continuous charge, we get that the minimum will be achieved when $\phi(z)$ is constant on ∂S_0. Hence we can subtract a constant C' from this potential in order that $\phi(z)|_{\partial S_0} = 0$. Then if we go back to (4.27) we get

$$(E_N(S_0))^{1/N} = \Big(\min\max_{z\in_{S_0}} \Big|\frac{p(z)}{p(0)}\Big|\Big)^{1/N}$$

$$= \Big(\min\max_{z\in_{S_0}} \Big|\frac{e^{N\phi(z)}}{e^{N\phi(0)}}\Big|\Big)^{1/N} = e^{-\phi(0)}.$$

Hence, the asymptotic convergence factor is equal to, $\rho = e^{-\phi(0)}$.

When S_0 is connected, the potential $\phi(z)$ can be seen as a level curve function of a conformal map. Let S_0 be connected compact subset of $\mathbb{C} \setminus \{0\}$ with piecewise smooth boundary. If we consider the extended complex plain $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$, then the exterior of S_0 is a simply connected set, [22].

We define

$$\Phi(z) = e^{(\phi(z)+ih(z))}$$

in the exterior of S_0, where $\phi(z)$ is the potential and $h(z)$ is the harmonic conjugate of $\phi(z)$, single-valued except for increasing by $2\pi i$ with each counterclockwise circuit around S_0.

Theorem 4.3.2. *[22] Let S_0 be connected and let $\Phi(z)$ be a conformal map of the exterior of S_0 to the exterior of the unit disk Δ with $\Phi(\infty) = \infty$. The asymptotic convergence factor of S is*

$$\rho = \frac{1}{|\Phi(0)|} \tag{4.33}$$

Again from [22], we have two significant examples of convergence factors ρ.

1. When $S = S_0$ is the disk $|z - z_0| \leq R$ for $R < |z_0|$, a conformal mapping from the exterior of S to the exterior of the disk is $\Phi(z) = \frac{(z-z_0)}{R}$. We find that for this case the asymptotic convergence is $\rho = \frac{1}{\Phi(0)} = \frac{R}{|z_0|}$.

   Let κ be the condition number that is defined by the ratio of the biggest to the smallest elements of S_0. Then we can write $z_0 = a(\kappa+1)/2$ and $R = |a|(\kappa-1)/2$, where a is a real positive constant. By substituting R and r_0 get that,

   $$\rho = \frac{\kappa - 1}{\kappa + 1}. \tag{4.34}$$

2. Let $S = S_0$ be an interval we derive estimates related with positive definite and negative definite matrices. The conformal map for this set is the inverse of a Joukowski map, which maps ellipses to circles. Let $S = [1, \kappa]$ for some

$\kappa > 1$.Then,

$$\Phi(z) = \frac{2z - \kappa - 1 + 2\sqrt{z^2 - (\kappa+1)z + \kappa)}}{\kappa - 1}.$$

For $z = 0$,

$$\Phi(0) = \frac{-\kappa - 1 + 2\sqrt{\kappa}}{k - 1} = -\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}$$

hence,

$$\rho = \frac{1}{|\Phi(0)|} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \tag{4.35}$$

As we mentioned in the introduction of this section, many readers are more familiar with convergence proofs related to Chebyshev polynomials. Hence, although we have proved the estimate (4.35) for the symmetric and positive definite A, by using potential theory, we will present a different approach based on article [33]. First we introduce the Chebyshev polynomials.

Definition 4.3.3. *The Chebyshev polynomials are,*

$$T_k(x) := \cos(k \arccos(x)) \quad x \in [-1, 1], \ k = 0, 1, 2, 3, \ldots$$

They satisfy the three terms recursion property, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ *for* $k \geq 2$.,

We define,

$$J_k(x) = \frac{T_k(\frac{2x-\lambda_{max}-\lambda_{min}}{\lambda_{max}-\lambda_{min}})}{T_k(\frac{-\lambda_{max}-\lambda_{min}}{\lambda_{max}-\lambda_{min}})} \tag{4.36}$$

where J_k is the scaled and shifted Chebyshev polynomial on the interval $[\lambda_{min}, \lambda_{max}]$. The bound for the numerator in (4.36) is 1 since

$$|T_k(x)| \leq 1. \tag{4.37}$$

This bound is attained at the endpoints of the interval. To estimate the size of the denominator we have to extended the polynomial outside the interval $[-1, 1]$, [33]. In this case we have $T_k(z) = \cosh(k \operatorname{arccosh} z)$, where $\cosh z = \frac{e^z+e^{-z}}{2}$, and if we set $z = \cosh(\log x) = \frac{1}{2}(x + x^{-1})$ then we get $T_k(z) = \frac{1}{2}(x^k + x^{-k})$.

Hence we set

$$-\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}} = \frac{1}{2}(x + x^{-1}). \tag{4.38}$$

In the case when A is Hermitian positive definite we know that the condition number can be written as $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$. Then (4.38) can we rewritten as

$$\frac{\kappa+1}{\kappa-1} = \frac{1}{2}(x + x^{-1}). \tag{4.39}$$

If we solve (4.39) for x we get that

$$x = -\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \; or \; x = -\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \tag{4.40}$$

Finally, from (4.24), (4.37) and (4.40), we get the following estimate,

$$\frac{||r_k||}{||r_0||} \leq \min_{J_k \in P_k} \max_{z \in \Lambda(A)} ||J_k||_{\Lambda(A)} \tag{4.41}$$

$$\leq 2\Big[\Big(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\Big)^k + \Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^k\Big]^{-1} \tag{4.42}$$

$$\leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^k. \tag{4.43}$$

We can observe that the estimate (4.43) is identical with estimate (4.35) derived using Theorem 4.3.2 where we used as a conformal mapping the inverse of the Joukowski map and S_0 is an interval.

4.3.2 Convergence for Non HPD Matrices

In this section we include estimates and important remarks related to the convergence properties of GMRES, for some general cases other than the case of Hermitian and positive definite matrices.

An important class of problems arise from the discretazation of partial differential equations and saddle point problems. For example, the stable mixed finite element methods for the Poison problem or the Stokes problem, [57],[64] lead to Hermitian indefinite operators.

In order to estimate the convergence rate of GMRES for a matrix A we need to find the smallest possible polynomial on the spectrum of A polynomial $J(z)$ such that $J(0) = 1$ and $J(z)$ is the smallest possible polynomial on the spectrum of A, see (4.24). As we have seen in the previous subsection for HPD matrices we have a satisfying "complete" solution, for a given condition number κ, and the convergence estimates can be given in terms of Chebyshev polynomials.

For indefinite matrices we can use a similar approach by using $J(z^2)$ and try to find the smallest possible polynomial on the square root of the spectrum of the indefinite matrix with $J(0) = 1$. For this case the estimate (4.1.3) becomes,

$$\frac{||r_k||}{||r_0||} \leq 2\Big(\frac{\kappa-1}{\kappa+1}\Big)^{k/2}. \tag{4.44}$$

This estimate is not necessarily sharp, especially when the spectrum of A is highly

non symmetric. For example, if A has a single eigenvalue at -1 and then consists only of positive spectrum, then MINRES on A will converge in terms of $\sqrt{\kappa}$, estimate (4.1.3), since the isolated eigenvalue -1 will not affect the convergence rate significantly. It is very hard to give a detailed analysis in the fully general case where eigenvalues are in (a, b) union (c, d), where $a < b < 0 < c < d$. However, the following result of Greenbaum is a significant improvement on the naive estimate above. As far as we know there is no sharper known result.

If we assume that the eigenvalues are contained in two intervals I^+, I^-, [33],

$$I^- \cup I^+ = [\lambda_{min}, \lambda_n] \cup [\lambda_{n+1}, \lambda_{max}]$$

where $\lambda_{min} \leq \lambda_n \leq 0 \leq \lambda_{n+1} \leq \lambda_{max}$. When $\lambda_n - \lambda_{min} = \lambda_{max} - \lambda_{n+1}$, then in [33] it has been shown that,

$$\min_{J_k \in P_k} \max_{z \in \Lambda(I^- \cup I^+)} ||J_k||_{\Lambda(A)} \leq 2\Big(\frac{\sqrt{|\lambda_{min}\lambda_{max}|} - \sqrt{\lambda_n \lambda_{n+1}}}{\sqrt{|\lambda_{min}\lambda_{max}|} + \sqrt{\lambda_n \lambda_{n+1}}}\Big). \tag{4.45}$$

According to [33], the kth order polynomial on $I^- \cup I^+$ with $J_k(0) = 1$, that has maximum deviation from 0 equal to 1 at $z = 0$, is given by,

$$J_k(z) = T_{[k/2]}(q(z))/T_{[k/2]}(q(0)) \tag{4.46}$$

with $q(z) = 1 + \frac{2(z-\lambda_n)(z-\lambda_{n+1})}{\lambda_{min}\lambda_{max} - \lambda_n \lambda_{n+1}}$ and where $[k/2]$ is the integer part of $k/2$. Similar to the case of the HPD operator, the numerator of (4.46) is bounded by 1, since for the Chebyshev polynomial of order $[k/2]$, $|T_{[k/2]}| \leq 1$. Moreover, following the procedure in (4.38), we set $q(0) = \frac{1}{2}(x + x^{-1})$ and hence $T_{[k/2]}(q(0)) = \frac{1}{2}(x^{[k/2]} + x^{-[k/2]})$.

When

$$q(0) = 1 + 2\frac{\lambda_{max}\lambda_{min} + \lambda_n \lambda_{n+1}}{\lambda_{max}\lambda_{min} - \lambda_n \lambda_{n+1}} = \frac{1}{2}(x + x^{-1}) = 0. \tag{4.47}$$

that leads to the quadratic equation,

$$\frac{1}{2}x^2 - q(0)x + \frac{1}{2} = 0. \tag{4.48}$$

The solutions of (4.48) are $x = \frac{\sqrt{\lambda_{max}\lambda_{min}} - \sqrt{\lambda_n \lambda_{n+1}}}{\sqrt{\lambda_{max}\lambda_{min}} + \sqrt{\lambda_n \lambda_{n+1}}}$ and $x = \frac{\sqrt{\lambda_{max}\lambda_{min}} + \sqrt{\lambda_n \lambda_{n+1}}}{\sqrt{\lambda_{max}\lambda_{min}} - \sqrt{\lambda_n \lambda_{n+1}}}$. Finally, we get the estimate for the kth step of GMRES,

$$\frac{||r_k||}{||r_0||} \leq 2\Big(\frac{\sqrt{\lambda_{max}\lambda_{min}} - \sqrt{\lambda_n \lambda_{n+1}}}{\sqrt{\lambda_{max}\lambda_{min}} + \sqrt{\lambda_n \lambda_{n+1}}}\Big)^{[k/2]} \tag{4.49}$$

In the case when the intervals are symmetric around the origin, for example when

$-\lambda_{min} = \lambda_{max}$ and $-\lambda_n = \lambda_{n+1}$ then we get,

$$\frac{||r_k||}{||r_0||} \leq \left(\frac{\kappa(A)-1}{\kappa(A)+1}\right)^{[k/2]}. \tag{4.50}$$

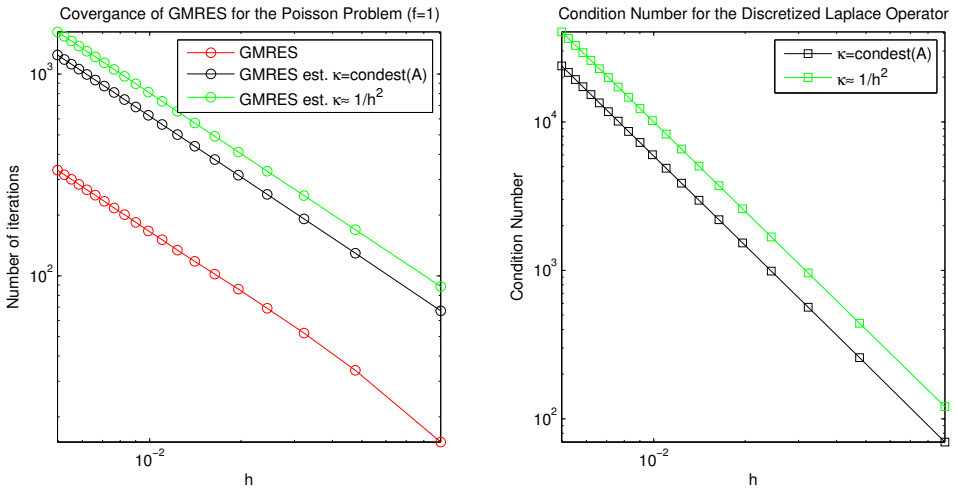


Figure 4.4: GMRES for the Possion Problem on uniform square.

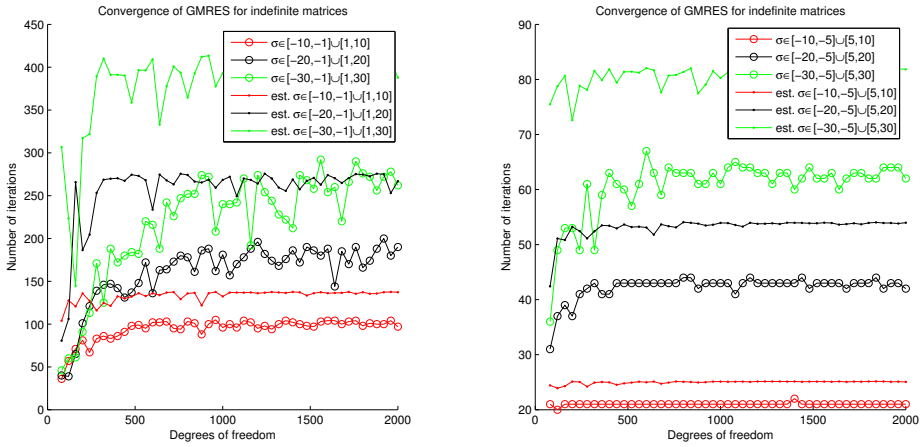


Figure 4.5: Convergence of GMRES for indefinite matrices

As it is mentioned in [33], [46], we observe that GMRES for the indefinite operator with condition number $\kappa(A)$ needs much more steps to reach the prescribed tolerance compared to a Hermitian positive definite operator with condition number $\kappa(A_{HPD}) = \kappa(A)^2$. Something that shows that Hermitian symmetric indefinite problems are a very challenging type of problems and an active field of research. Finally if A is a general normal matrix, meaning that $\kappa(V) = 1$, then again the behaviour depends mainly on the spectrum of A but on the contrary, when A is far from normal then convergence estimates based on Theorem 4.3.1 are most likely to fail to predict the behaviour of the GMRES method. For more details see [33], [46].

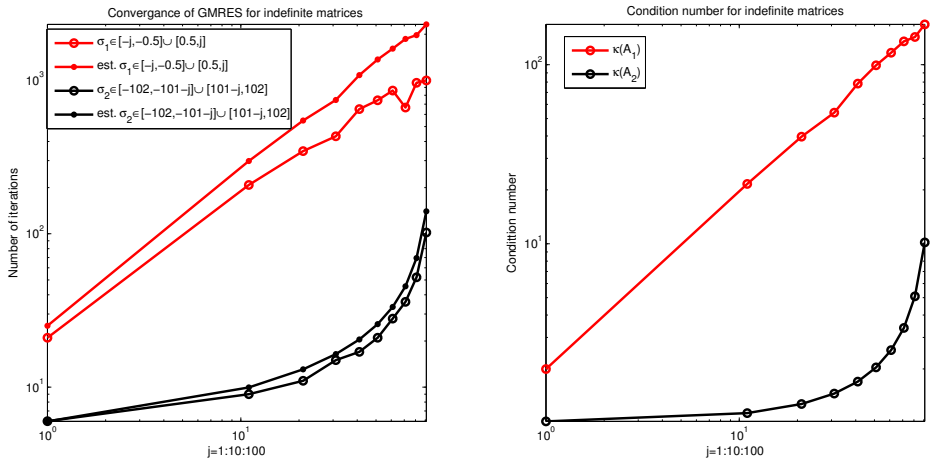


Figure 4.6: Convergence of GMRES for indefinite matrices

Remark 4.3.4. *Let* $\delta_k = \min_{J_k \in P_k} \max_{z \in \Lambda(A)} ||J_k||_{\Lambda(A)}$, *in Section 3 of [57] we can find some interesting observations about the convergence rate for Hermitian indefinite matrices. The authors note that for the general case of two intervals* I^+, I^- *with* $I = I^- \cup I^+$, *although it is hard to derive sharp estimates, they were able to present some interesting properties. They suggest that with proper scaling of the polynomials* P_k, *we obtain that* $\delta_k(aI) = \delta(I)$ *for any* $a \neq 0$ *and also that if* $\tilde{I} \subset I$ *them* $\delta_k(\tilde{I}) \leq \delta_k(I)$. *Hence the estimate* (4.24) *indicates that when we shrink the intervals* I^+, I^-, *GMRES converges faster.*

Let S be a compact subset of the complex plane that approximates the spectrum $\Lambda(A)$, $S \subset \mathbb{C}$. Then for the case of non-Normal matrices we can try to estimate the worst case convergence for GMRES by,

$$\frac{||r_k||}{||r_0||} \leq \min_{J \in P_k} ||J(A)|| \leq c \min_{J \in P_k} \max_{\lambda \in S} |J(\lambda)|. \tag{4.51}$$

We can choose S to be the ϵ-pseudospectrum of A, [46],[70]. Then for $\epsilon > 0$ the ϵ-pseudospectrum of A is defined by,

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : \ ||(zI - A)^{-1}|| \geq \epsilon^{-1}\}, \tag{4.52}$$

or alternately,

$$\Lambda_{epsilon}(A) = \{z \in \mathbb{C} : \ z \ is \ an, \ eignenvlaues \ of \ A+E \ for \ some \ E \ with \ ||E|| \leq \epsilon\}. \tag{4.53}$$

where the convention is used that $||(zI - A)^{-1}|| = \infty$ if z is an eigenvalue of A.

Then we define the boundary $\partial\Lambda_\epsilon(A)$ of $\Lambda_\epsilon(A)$, on which the resolvent norm is constant, $||(zI - A)^{-1}|| = \epsilon^{-1}$. The polynomial $J(A)$ can be written as a Cauchy

integral,

$$J(A) = \frac{1}{2\pi i} \int_{\partial\Lambda_\epsilon(A)} J(\lambda)(\lambda I - A)^{-1} d\lambda. \tag{4.54}$$

Finally we derive the following bound,

$$\min_{J \in P_k} ||J(A)|| \leq \frac{L(\partial\Lambda_\epsilon(A))}{2\pi}) \max_{\lambda \in \partial\Lambda_\epsilon(A)} ||J(\lambda)(\lambda I - A)^{-1}|| \tag{4.55}$$

$$\leq \frac{L(\partial\Lambda_\epsilon(A))}{2\pi\epsilon} \min_{J \in P_k} \max_{\lambda \in \partial\Lambda_\epsilon(A)} |J(\lambda)|. \tag{4.56}$$

Another approach to obtain residual bounds according to [33],[46],[10], is based on the field of values of matrix A.

$$\mathcal{F}(A) = \{v^T A v \ : \ ||v|| = 1, \ v \in \mathbb{C}^n\}. \tag{4.57}$$

An alternative equivalent definition is

$$\mathcal{F}(A) = \{\frac{v^T A v}{v^T v} \ : \ v \in \mathbb{C}^n, \ v \neq 0\}, \tag{4.58}$$

where $\mathcal{F}(A)$ is a convex set that contains the convex hull of the eigenvalues of A. When A is normal, then $\mathcal{F}(A)$ is exactly the convex hull of the eigenvalues. We define

$$\nu(A) = \max\{|z| \ : \ z \in \mathcal{F}(\mathcal{A})\}$$

as the numerical radius of $\mathcal{F}(A)$.

From [25] by using the field of values of A the authors derive the following estimate. Let $D = \{z \in \mathbb{C}, |z - c| \leq r\}$ be the disk with centre c and radius r. If the field of values $\mathcal{F}(A)$ is contained in this disk, then we get

$$\frac{||r_k||}{||r_0||} \leq 2\Big(\frac{r}{|c|}\Big)^k. \tag{4.59}$$

If A is a normal matrix and its spectrum is contained in an ellipse, we have the following result due to [60]. In the case that we have an "almost real spectra", such that the eigenvalues are contained in an ellipse with centre c, foci $c + \epsilon$ and major semi-axis $a > 0$, then from Theorem 4.4 in [60] we get

$$\frac{||r_k||}{||r_0||} \leq \frac{T_k(a/\epsilon)}{|T_k(c/\epsilon)|},$$

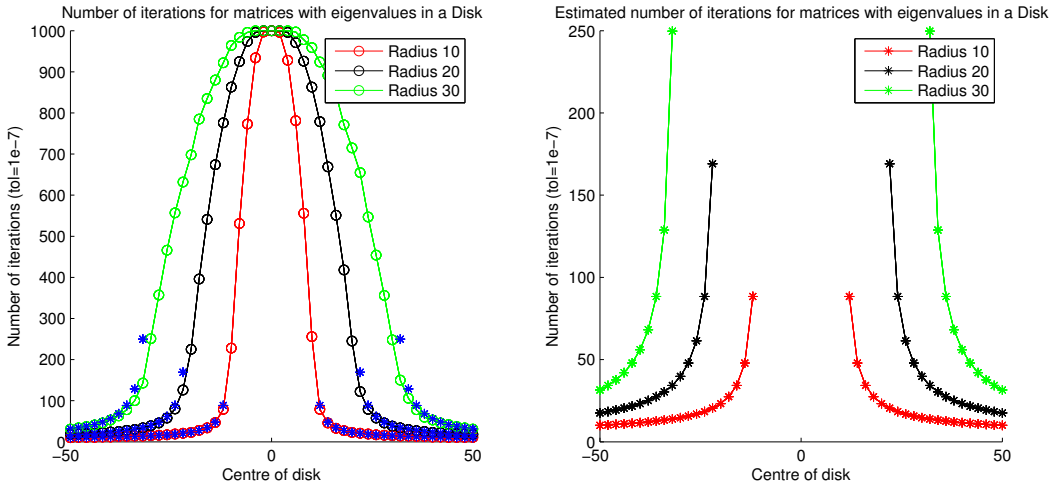


Figure 4.7: GMRES Disk eigenvalues

which is asymptotically equivalent to,

$$\frac{||r_k||}{||r_0||} \leq \left[\frac{a+\sqrt{a^2-\epsilon^2}}{|c|+\sqrt{c^2-\epsilon^2}}\right]^k. \tag{4.60}$$

Since for normal matrices the operator norm equals the spectral radius from 4.57 we get that that $\sigma(A) = \mathcal{F}(A)$.

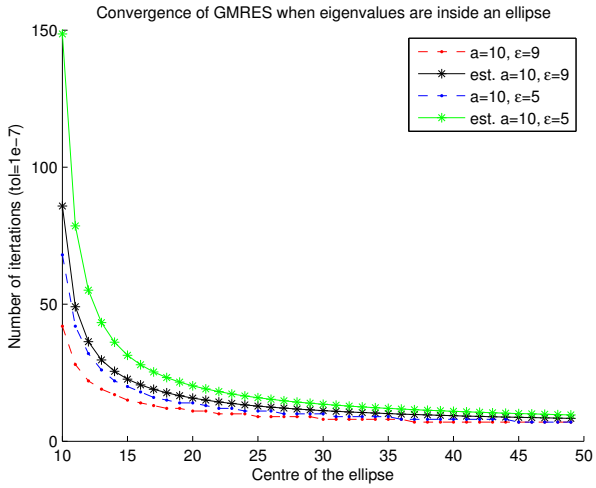


Figure 4.8: GMRES Ellipse Eigenvalues

Remark 4.3.5. *An interesting observation also made in [60], is that for the case of an degenerate ellipse which lies on the real axis, we get exactly the same estimate with the one that we have derived by Chebyshev polynomial for the case of symmetric positive definite matrices. Indeed, let the spectrum of A be $[\lambda_{min}, \lambda_{max}]$, in the case of the degenerate ellipse, we have that $\epsilon = a = \frac{\lambda_{max}-\lambda_{min}}{2}$, $c = \frac{\lambda_{max}+\lambda_{min}}{2}$. If we substitute c, a, ϵ with the ones above and divide the numerator and the denominator of the*

estimate (4.60) *by* λ_{min} *we get,*

$$\begin{aligned}\rho &= \frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}+2\sqrt{\lambda_{max}\lambda_{min}}} \\ &= \frac{\frac{\lambda_{max}-\lambda_{min}}{\lambda_{min}}}{\frac{\lambda_{max}+\lambda_{min}+2\sqrt{\lambda_{max}\lambda_{min}}}{\lambda_{min}}} \\ &= \frac{\kappa-1}{\kappa+1+2\sqrt{\kappa}} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\end{aligned} \tag{4.61}$$

where $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$*, which agrees with the estimate* (4.35)*.*

Moreover, in the case when the major and the minor semi-axis of the ellipse are equal, we get that the convergence asymptotic estimate is $\rho = (\frac{a}{c})$*, which agrees with the estimate* (4.59)*. In general, we observe that we start from good convergence properties for the positive definite case and the convergence rate becomes worse as the field of values moves away from the degenerate ellipse case and closer to a disk formation.*

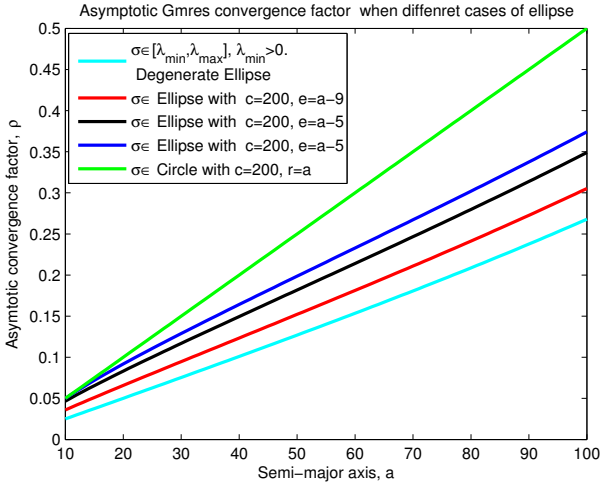


Figure 4.9: GMRES Ellipse Eigenvalues

Remark 4.3.6. *We finish this chapter with a remark that can be found in Section* 3 *of article [9]. This remark can also be useful in order to attempt to explain the better performance of the non-symmetric over the symmetric 2-Lagrange multiplier method, since we use a similar transformation that reflects the eigenvalues to positive part of the axis.*

The authors, in [9], consider a large class of saddle point problems. After the discretization of this problems by a suitable method, like the finite element method, we obtain Hermitian indefinite matrices. Assume that we have a matrix A such that,

$$A = \begin{bmatrix} H & B^T \\ B & -C \end{bmatrix}$$

where $H \in \mathbb{R}^{n\times n}$ is symmetric positive definite, $B \in \mathbb{R}^{n\times m}$ has full rank and $C \in \mathbb{R}^{m\times m}$ is symmetric positive semidefinite. The matrix A is a Hermitian indefinite matrix. The authors provide a plethora of examples where there has been observed that when we apply a simple linear transformation J on A, then we get complex eigenvalues of JA that lie on the right part of the complex plane, (they have real positive part). Let $A^* = JA$ where

$$J = \begin{bmatrix} I_n & 0 \\ 0 & -I_m \end{bmatrix}.$$

In many cases it has been observed that when the eigenvalues have positive real part, then the convergence of GMRES is faster, [9]. This might be related with the fact that, as we have seen previously, when the spectrum is clustered in two distinct sets, I^- with negative real part and I^+ with positive real part, it is harder to be approximated by polynomial P_k, at least compared with the case that the eigenvalues have positive real part and are clustered away from the origin.

Chapter 5

2-Lagrange Multiplier methods-Optimized Schwarz methods

5.1 The 2-Lagrange Multiplier methods

The 2-Lagrange multiplier methods belong to the class of non-overlapping domain decomposition methods for solving numerically large scale elliptic problems. The symmetric 2-Lagrange multiplier method is a linear system of the form

$$(Q - K)\lambda = h_s, \tag{5.1}$$

where Q is symmetric and positive definite, K is an orthogonal projection, λ is the unknown and $h_s = -Qg$ is the data. Matrix Q is block diagonal and hence the calculation of the matrix-vector product $Q\lambda$ can be calculated efficiently in parallel. As regards the matrix K, this matrix is not block diagonal but is extremely sparse, hence it can be assembled in a parallel space matrix format, e.g in a compressed row format (CRS) which is the default sparse matrix representation in PETSc [6].

These "one-level" methods were introduced and extensively analysed in [48], based on the related methods introduced in [26]. One of the main goals of domain decomposition methods is to solve efficiently problems such as (5.1) in parallel. To achieve that, we use Krylov subspace solvers such as MINRES [53] and GMRES [59]. It has been shown in [48] that the condition number increases unboundedly when the number of subdomains p increases.

Methods like (5.1) are often called one-level methods. These one-level methods are effective for small problems but they fail for large scale problems, [65]. In order to achieve scalable methods we introduce a "coarse grid" preconditioner P, which leads to two-level algorithms with very good parallel scaling properties.

It is known, [30], that the 2-Lagrange multiplier methods are very closely related with the non-overlapping Optimized Schwarz methods. These methods have been used in order to solve a large spectrum of problems like the Helmholtz problem and convection-diffusion related problems, [24], [29], [48]. We define the initial domain Ω which is decomposed in p subdomains $\Omega = \bigcup_{k=1}^{p} \Omega_k$ and the artificial interface $\Gamma = \bigcup_{k=1}^{p} \partial\Omega_k \backslash \partial\Omega$. For each vertex $x_j \in \Gamma$, we let m_j be the number of subdomains adjacent to x_j. If $m_j = 2$ then x_j is a regular interface vertex, otherwise if $m_j > 2$ then x_j is a cross point. For example in Figure 5.1 vertex x_5 is a cross point. In the case that there are no cross points and we have a non-overlapping partition of the initial domain Ω, we essentially have partitioned the domain into strips [69]. Moreover, in the case that our domain is decomposed into strips, there are two Lagrange multipliers per interface point, something that explains the nomenclature.

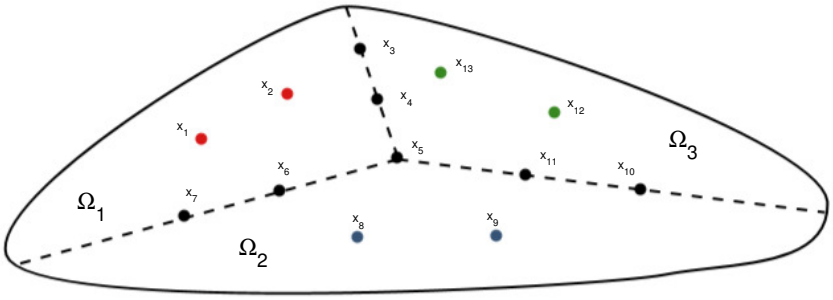


Figure 5.1: Domain Ω divided in three non-overlaping subdomains

In the non-overlapping domain decomposition methods we subdivide the domain in a way that the data are shared in a balanced way among the processors. Then, instead of solving the large elliptic problem on the whole subdomain, we are able to solve the much smaller local sub-problems in parallel. This might not be so obvious in the 2-Lagrange multiplier methods, since in equation (5.1), we need to solve a linear system defined on the whole artificial interface. Nevertheless, due to the fact that Q is block diagonal, in order to calculate the matrix-vector product $Q\lambda$, we can solve local Robin subproblems corresponding to each subdomain independently in parallel.

In this Section we see that the symmetric and non-symmetric 2-level 2-Lagrange muliplier methods scale weakly, which means that the condition number remains bounded as we increase the number of processors and the size of the problem. We also emphasise on the connection between these methods and the OSM method. Finally we present a set of numerical experiments that confirm the theoretical results. The massively parallel implementation and the large scale experiments performed on

HECToR supercomputer are presented in Chapter 6.

The 2-Lagrange multiplier methods apply to general self-adjoint and coercive elliptic partial differential equations. In order to present our methods we will consider the model problem

$$-\Delta\tilde{u} = \tilde{f} \text{ in } \Omega \subset \mathbb{R}^d, d = 2, 3 \text{ and } \tilde{u} = 0 \text{ on } \partial\Omega. \tag{5.2}$$

In order to solve problem (5.2) numerically, we discretise by using a suitable finite element method, discussed in Chapter 3, or a finite difference method [68]. After the discretisation we get the global discrete system,

$$Au = f, \tag{5.3}$$

where A is a large symmetric and positive definite sparse matrix, f is the load vector and u is the desired discrete solution of our problem. We use the notation $\tilde{u} = \tilde{u}(x)$ for the solution $\tilde{u} \in H_0^1(\Omega)$ and u for the corresponding finite element coefficient vector.

As regards the restrictions on the type of meshes and domain decompositions that can be used with our methods we need the following. From the point of view of ensuring that the linear algebra works we expect quasi uniform, conforming meshes, this is so that we can easily assemble the K matrix which is a continuous and coercive operator. As regards the domain decomposition is simply a partition of the triangulation T_h into non overlapping subdomains formed by the union of then finite elements $t_i \in T_h$.

5.1.1 Obtaining the S2LM system from the global system (5.2)

We partition domain Ω into p non-overlapping subdmains $\Omega_1, \ldots, \Omega_p$. From now on we assign each subdomain to a separate processor and we assume that the number of processors coincides with the number of subdomains. Next we define the artificial interface

$$\Gamma = \bigcup_{k=1}^{p} \partial\Omega_k \setminus \partial\Omega$$

where the set $\partial\Omega$ is called the natural boundary of Ω.

The main idea of the 2-Lagrange Multiplier methods is to replace the original system (5.3) with the 2-Lagrange multiplier systems of smaller dimension. These systems are: the indefinite system that corresponds to the Symmetric 2-Lagrange multiplier method,

$$A_{S2LM}\lambda = h_s; \tag{5.4}$$

and the non-symmetric system that corresponds to the non-Symmetric 2-Lagrange

multiplier method,

$$A_{N2LM}\lambda = h_n, \tag{5.5}$$

where the λ is the Lagrange multipliers vector. Since systems (5.4), (5.5) can still be too large to be solved by a direct solver, we are interested in implementing a parallel iterative solver that exploits the decomposition of the domain Ω.

We start the analysis of the methods, by deriving analytically the 2-Lagrange multiplier methods, by splitting the original problem (5.2) to an equivalent system of local Robin subproblems,

$$\begin{cases} -\Delta\tilde{u}_k = \tilde{f}_k & \text{in } \Omega_k, \\ \tilde{u}_k = 0 & \text{on } \partial\Omega_k \cap \partial\Omega, \\ (a + D_\nu)\tilde{u}_k = \tilde{\lambda}_k & \text{on } \partial\Omega_k \cap \Gamma; \end{cases} \tag{5.6}$$

where $a > 0$ is the Robin parameter, $k = 1, \ldots, p$, D_ν denotes the directional derivative in the direction of the exterior unit normal ν of $\partial\Omega_k$, and $\tilde{\lambda_k}$ is the Robin flux data imposed on the "artificial interface" $\partial\Omega_k \cap \Gamma$.

By multiplying subproblems (5.6) by a test function $v \in V_k$ and then using Green's formula, we obtain the weak formulation of the local Robin subproblems.

Find $\tilde{u}_k \in V_k$ such that,

$$\int_{\Omega_k} \nabla\tilde{u}_k \nabla v dx + a\int_{\partial\Omega_k\cap\Gamma} \tilde{u}_k v dx = \int_{\Omega_k} \tilde{f}_k v dx + \int_{\partial\Omega_k\cap\Gamma} \tilde{\lambda}_k v dx \tag{5.7}$$

holds for all $v \in V_k$, where $V_k = \{v \in H^1(\Omega_k) \mid v = 0 \text{ on } \partial\Omega_k \cap \partial\Omega\}$.

In principle, the term $\int_{\partial\Omega_k\cap\Gamma} \tilde{u}_k v dx$ would give rise to a mass matrix on the artificial interface $\partial\Omega_k \cap \Gamma$. However, we know from (3.2.11) that for a quasi- uniform meshes the mass matrix is spectrally equivalent to the identity matrix by some h factors, which have been absorbed into the a parameter.

If we discretise (5.6) using the finite element method, we have the following coupled systems,

$$\begin{bmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix}. \tag{5.8}$$

Here, the subscript I denotes the nodes that are in the interior of Ω_k, while the subscript Γ denotes the nodes on $\Gamma \cap \partial\Omega_k$; this notation is consistent with existing literature, see [69].

Remark 5.1.1. *For each subdomain $\Omega_1, \ldots, \Omega_p$ the corresponding restriction matrices $R_1, \ldots, R_p$ are obtained. We have seen that the discretization of the model problem (5.2) leads to a linear system of the form*

$$Au = f.$$

By using the restriction matrices we can write the global matrix A in (5.3)*, as a linear combination of the restriction and interpolation of the local Neumann matrices* A_{Nk} *for each subdomain* Ω_k*. Let,*

$$A_{Nk} = \begin{bmatrix} A_{II k} & A_{I\Gamma k} \\ A_{\Gamma I k} & A_{\Gamma\Gamma k} \end{bmatrix} \text{ and } f_k = \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} \end{bmatrix}, \tag{5.9}$$

then the global matrix A can be written as a linear combination of the restriction and interpolation of local Neumann matrices (5.9) *for each subdomain* Ω_k*,*

$$A = \sum_k R_k^T A_{Nk} R_k \tag{5.10}$$

in the same manner f now can be written as,

$$f = \sum R_k^T f_k. \tag{5.11}$$

The matrix A_{Nk} *is obtained by discretizing the bilinear form* $\int_{\Omega_k} \nabla u \cdot \nabla v$*.*

Remark 5.1.2. *The procedure in order to eliminate the interior degrees of freedom and to derive the Schur complement of* (5.9) *is the following,*

$$\begin{bmatrix} I & 0 \\ A_{\Gamma I k} A_{II k}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{II k} & A_{\Gamma I k} \\ 0 & A_{\Gamma\Gamma k} - A_{\Gamma I k} A_{II k}^{-1} A_{I\Gamma k} \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} \end{bmatrix}$$

or equivalently,

$$\begin{bmatrix} A_{II k} & A_{\Gamma I k} \\ 0 & A_{\Gamma\Gamma k} - A_{\Gamma I k} A_{II k}^{-1} A_{I\Gamma k} \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A_{\Gamma I k} A_{II k}^{-1} & I \end{bmatrix} \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} \end{bmatrix}$$

and we finally get,

$$\begin{bmatrix} A_{II k} & A_{\Gamma I k} \\ 0 & S_k \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} f_{Ik} \\ f_{\Gamma k} - A_{\Gamma I k} A_{II k}^{-1} f_{Ik} \end{bmatrix}.$$

By using the same procedure as in Remark (5.1.2) for the Schur complement, we eliminate the interior nodes of equation (5.8) to get the equivalent system

$$\overbrace{\begin{bmatrix} S_1 + aI & & \\ & \ddots & \\ & & S_p + aI \end{bmatrix}}^{S+aI} \overbrace{\begin{bmatrix} u_{\Gamma 1} \\ \vdots \\ u_{\Gamma p} \end{bmatrix}}^{u_G} = \overbrace{\begin{bmatrix} g_1 \\ \vdots \\ g_p \end{bmatrix}}^{g} + \overbrace{\begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_p \end{bmatrix}}^{\lambda}. \tag{5.12}$$

In a more compact form,

$$(S + aI)u_G = g + \lambda, \tag{5.13}$$

where $S = \text{diag}\{S_1, \ldots, S_p\}$ with symmetric and semidefinite Schur complements $S_k = A_{\Gamma\Gamma k} - A_{\Gamma I k} A_{I I k}^{-1} A_{I \Gamma k}$; the column vector $u_G = [u_{\Gamma 1}^T, \ldots, u_{\Gamma p}^T]^T$ is the multi-valued trace, with one value per interface vertex per adjacent subdomain, the "Robin data" $\lambda = [\lambda_1^T, \ldots, \lambda_p^T]^T$ and the "accumulated fluxes" are $g_k = f_{\Gamma k} - A_{\Gamma I k} A_{I I k}^{-1} f_{I k}$.

We can rewrite (5.13) as

$$a u_G = Q(g + \lambda). \tag{5.14}$$

where Q is the scaled "Robin-to-Dirichlet" map $Q = \text{diag}\{Q_1, \ldots, Q_p\}$, where $Q_k = a(S_k + aI_k)^{-1}$. We call u_G the multi-valued trace, since it can be interpreted as the trace of a finite element function which is discontinuous along Γ. Moreover, as we can also see in detail in Theorem 5.1.3, we can enforce the continuity and flux transmission conditions by,

$$K u_G = u_G \tag{5.15}$$

and

$$K(S u_G) = K g. \tag{5.16}$$

Where K is an orthogonal projection that averages the function values for each interface vertex, presented in Remark 5.1.4.

Adding equations (5.14) and (5.15) and substituting (5.16) we produce the Symmetric 2-Lagrange multiplier system,

$$\overbrace{(Q - K)}^{A_{S2LM}} \lambda = -Qg. \tag{5.17}$$

Left multiplying (5.17) by $(I - 2K)$, gives the Nonsymmetric 2-Lagrange multiplier system,

$$\overbrace{(I - 2K)(Q - K)}^{A_{N2LM}} \lambda = -(I - 2K)Qg. \tag{5.18}$$

We define that (5.17) or (5.18) are equivalent in the following sense. If we solve either equation (5.17) or (5.18), then in order to find the local solutions u_k of each subdomain, we substitute the resulting λ_k, $k = 1 \ldots p$ in (5.8) and then we get the local solution $u_1, \ldots, u_p$. For each element on the local solutions u_k we have a local to global numbering map, that associates the local indices Ω_k to the global indices of the *global* domain Ω. By gathering all the local solutions together we assemble the global vector u by using the local to global numbering. The solution u that we have derived is the original solution of the global system $Au = f$.

Theorem 5.1.3. *Let E be the orthogonal projection onto the kernel of $I - Q$. Assume that $Q - K$ is nonsingular and $\|EK\| < 1$. The problem* (5.3) *is equivalent to* (5.17).

Proof. Let us assume that we decompose our domain Ω in p non-overlaping subdomains. Then we get p local Robin subproblems with a set of Lagrange multipliers λ_i on the artificial interface of each subdomain Ω_i. In order to prove that the global problem (5.2) is equivalent to (5.17) we need to check that the values of the Lagrange multipliers λ_i $i = 1 \dots p$ give rise to their corresponding " primal" solutions u_i $i = 1 \dots p$, which are continuous and their fluxes match.

By imposing the continuity condition (5.15) we get

$$Ka(S + aI)^{-1}(\lambda + g) = a(S + aI)^{-1}(\lambda + g)$$

and since $Q = a(S + aI)^{-1}$,

$$KQ(\lambda + g) = Q(\lambda + g). \tag{5.19}$$

Alternatively we can rewrite (5.19) as,

$$(I - K)Q\lambda = (K - I)Qg. \tag{5.20}$$

Imposing only the continuity condition on the solution is not sufficient, we must also ensure that the "fluxes" match, condition (5.16). Since u is continuous on the artificial interface, there is a unique u and restriction matrices R_j such that

$$u_j = R_j u \ \ , j = 1, \dots p. \tag{5.21}$$

Hence, we can expand Au in the following way,

$$f = Au = \sum_{k=1}^{p} R_k^T A_{N_k} R_k u.$$

From (5.21) we have

$$f = \sum_{k=1}^{p} R_k^T A_{N_k} u_k$$

and from (5.8) we get

$$f = \sum_{j=1}^{p} R_j^T \begin{pmatrix} f_{Ik} \\ f_{\Gamma_k} \end{pmatrix} + \sum_{k=1}^{p} R_k^T \begin{pmatrix} 0 \\ \lambda_k - a u_{\Gamma_k} \end{pmatrix}.$$

Since from (5.1.1)

$$f = \sum_{k=1}^{p} R_k^T \begin{pmatrix} f_{Ik} \\ f_{\Gamma_k} \end{pmatrix}$$

we derive that

$$\sum_{k=1}^{p} R_k^T \begin{pmatrix} 0 \\ \lambda_k - a u_{\Gamma_k} \end{pmatrix} = \begin{pmatrix} 0 \\ \lambda - a u_{\Gamma} \end{pmatrix}.$$

Finally, if we multiply both sides by the orthogonal averaging operator K, we get that

$$K\lambda - aK\mathbf{u}_G = 0.$$

Hence using (5.14)

$$-K\lambda + KQ\lambda + KQg = 0;$$

we get that

$$K(Q - I)\lambda = -KQg. \tag{5.22}$$

Now if we add (5.20) and (5.22) we get (5.17). □

Remark 5.1.4. *In this remark we will look in some more detail the averaging orthogonal operator and the restriction matrices R_i. In Figure* 5.1 *we divide the domain in three non-overlapping subdomains and points x_i, $i = 1 \dots 13$. We have two points in the interior of each subdomain Ω_i, $i = 1, 2, 3$ and* 5 *nodes on the artificial interface Γ, denoted by the dashed lines. In this case we have six regular interface vertices, vertices shared between two subdomains, and only one cross point x_5, which is shared between more than two subdomains, in this case between three subdomains.*

As an example, we will present the restriction matrix R_2 for subdomain Ω_2,

$$R_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The matrix R_2 acts on the global vector u over the global domain Ω and restricts it to the degrees of freedom that correspond to subdomain Ω_2,

$$R_2\overline{u} = \begin{bmatrix} u_{\Gamma_1} \\ u_{\Gamma_2} \\ u_{\Gamma_3} \\ u_{I_1} \\ u_{I_2} \\ u_{\Gamma_4} \\ u_{\Gamma_5} \end{bmatrix}.$$

We want to point out, that in the case of p subdomains, for each subdonain we get a local solution vector $\overline{u}_i$, $i = 1 \dots p$. *Each* u_i *can be decomposed in an interior part of the solution* u_{I_i} *and the part that lies on the artificial interface* u_{Γ_i}. *If we concatenate all the* u_{I_i} *in a single vector, we get a global vector* u_G *that is continuous inside* Ω_i, *but has jumps on the degrees of freedom corresponding to vertices on the artificial interface.*

The operator K *is an averaging operator that is applied on the degrees of freedom that belong on the artificial interface, in order to enforce the continuity on global vector* u_G, *we refer to* u_G *as the multi-valued trace, [48].*

From a different perspective, matrix K *or the projection on the null space of* K, $I - K$, *can be seen as the graph Lapclacian on the interface* Γ. *We define* G *to be an undirected graph defined by a set* $\mathcal{N} = 1, 2, ..., N$ *of* N *nodes and the set* $E \subset \mathcal{N} \times \mathcal{N}$ *of edges. With an edge we define the link between the two nodes of the graph* $x_j \sim x_i$. *Moreover we say that* G *is connected if for every pair of nodes* x_i, x_j *there is a finite sequence of nodes such that* $x_i \sim x_{i+1} \sim \dots \sim x_j$.

Now consider the case where we have three subdomains as in Figure 5.1. *Since the nodes on the artificial interface* Γ *are shared between two or more domains, we have fifteen degrees of freedom, five for each sobdomain, and* x_5 *is the only cross point shared between all subdomains.*

Assume that vertices of the graph that correspond to each degree of freedom are y_1, y_2, y_3, y_5 *for subdomain* Ω_1 *and* y_6, y_7,y_8, y_9,y_{10} *for subdomain* Ω_2 *and* y_{11}, y_{12}, y_{13}, y_{14},y_{15} *for subdomain* Ω_3 .

We define $y_i \sim y_j$ *if they are shared between two subdomains. In the standard notation we define* L *as the adjacency matrix where* $L_{ij} = 1$, *if* $y_i \sim y_j$ *and* $L_{ij} = 0$ *else. Also let* W *be a diagonal matrix of weights, where* $w_i = W_{ii} = 1/2$ *if the node* y_i *is shared between* 2 *subdomains and* $W_{ii} = 1/3$ *if the node shared between* 3 *subdomains, meaning that it is a cross point. Then* K *can be written as* $K = WL$.

$$K = \begin{bmatrix}
1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \\
0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \\
0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 \\
1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 1/2
\end{bmatrix}.$$

We have $y = [y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}]$*, hence the matrix vector product* Ky *we gives*

$$Kx = W \begin{bmatrix}
y_1 + y_{11} \\
y_2 + y_{12} \\
y_3 + y_6 + y_{13} \\
y_4 + y_7 \\
y_5 + y_8 \\
y_6 + y_3 + y_{13} \\
y_7 + y_4 \\
y_8 + y_5 \\
y_9 + y_{14} \\
y_{10} + y_{15} \\
y_{11} + y_1 \\
y_{12} + y_2 \\
y_{13} + y_3 + y_6 \\
y_{14} + y_9 \\
y_{15} + y_{10}
\end{bmatrix}.$$

Then if we calculate the $y^T Ky$ *product and set it to zero we get*

$$y'Ky = w_1(y_1 + y_{11})^2 + w_2(y_2 + y_{12})^2 + \ldots + w_{15}(y_{15} + y_{10})^2 = 0,$$

where $w_i = 1/2$ *or* $w_i = 1/3$, *which means that* $w_i > 0$ *for* $i = 1 \dots 15$. *Hence,* $y_1 = -y_{11}$ *and* $y_2 = -y_{13}$, $y_3 = -(y_6 + y_{13})$ *and we observe that the nullspace of* K *can be defined as*

$$null(K) = \{y \in R^m \ \ st.\ y_i = 0 \ \ if y_i \not\sim y_j \ \ and \ \sum_{y_i \sim y_j} (y_i) = 0.\} \tag{5.23}$$

5.2 Connections of 2-Lagrange Multiplier methods and the Optimized Schwarz method

In this section we will discuss the connections between the Optimized Schwarz method and the 2-Lagrange multiplier methods. It is known that these methods are closely related, [30], [48], [34]. More precisely in the case when the subdomains are arranged in strips, it is known that the Richardson iteration applied to the non-symmetric Lagrange multiplier system (5.18) is equivalent to the Optimized Schwarz method [66], [48]. An extended and concrete presentation and analysis of Optimized Schwarz methods can be found in [29].

In the introduction of this chapter, we present the Classical Schwarz methods. Then we present the Optimized Schwarz methods and finally the intrinsic connection between the optimized Schwarz method and the 2-Lagrange multiplier methods.

One of the main motivations for the development of Optimized methods, was the fact that the classical Schwarz methods failed to converge for the case of non-overlapping domains, [29]. J.L. Lions pointed out this disadvantage and proposed in 1988, [47], for the first time, the optimized variant of the classical Schwarz methods.

Althought the classical Schwarz methods are considered as the origin of the domain decomposition methods, there has been a variety of new methods which are efficient and widely used, developed during recent decades. Many of them have better convergence properties than classical Schwarz methods. Nevertheless, classical Schwarz methods are well understood, with a solid theoretical framework and they have a relatively simple implementation. Moreover, the are already a variety of classical Schwarz parallel implementations, capable to solve large scale problems efficiently in parallel, for example the `PCASM` preconditioner in the PETSc library.

The convergence properties of the classical Schwarz methods have been studied extensively, for example in [65], [69]. In Figure 5.2 we can see the simple case of two overlapping subdomains, where a domain Ω is divided in two subdomains, Ω_1, Ω_2, such that $\Omega_1 \cup \Omega_2 = \Omega$. The overlap has width δ and the diameter of Ω_1, Ω_2 is equal to H. The classical Schwarz method originates from H. A. Schwarz work in his celebrated paper, [62]. There he proposes an iterative method commonly referred to as Schwarz alternating method, while he tries to construct harmonic solutions of

elliptic PDEs on irregular domains by subdividing these domains into simple regular domains like circles and squares.

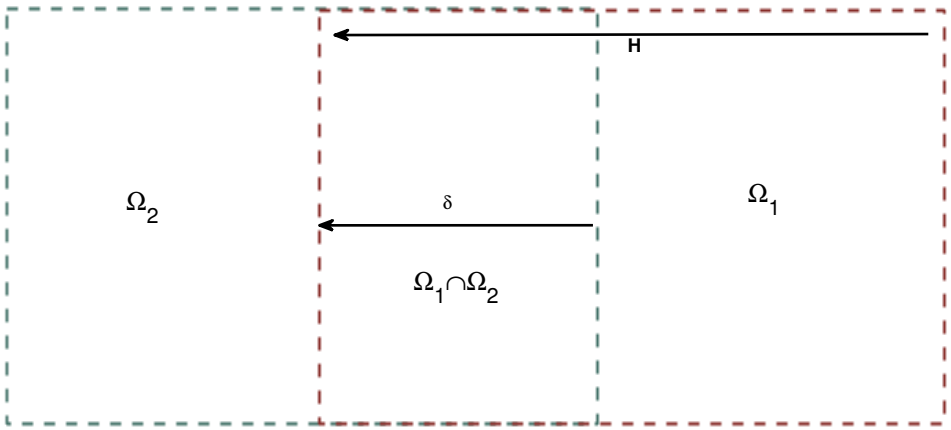


Figure 5.2: Divive Ω in two overlaping subdomains

At the continuous level, the Schwarz alternating method can be written as an iterative procedure over subdomains, e.g [69], Ω_1, Ω_2 with $\Gamma_i = \Omega \cap \partial\Omega_i$. We start from an initial guess u_2^0 and we repeat sequentially the following steps, For $n = 1, 2, 3, \ldots$
Step 1:

$$\Delta \tilde{u}_1^n = \tilde{f}_1 \ in \ \Omega_1 \tag{5.24}$$

$$\tilde{u}_1^n = \tilde{g}_1 \ \ on \ \partial\Omega_1 \setminus \Gamma_1 \tag{5.25}$$

$$\tilde{u}_1^n = \tilde{u}_2^{n-1}|_{\Gamma_1} \ on \, \Gamma_1 \tag{5.26}$$

Step 2:

$$\Delta \tilde{u}_2^n = \tilde{f}_2 \ in \ \Omega_2 \tag{5.27}$$

$$\tilde{u}_2^n = \tilde{g}_2 \ \ on \ \partial\Omega_2 \setminus \Gamma_2 \tag{5.28}$$

$$\tilde{u}_2^n = \tilde{u}_1^n|_{\Gamma_2} \ on \, \Gamma_2 \tag{5.29}$$

If we discretise equations (5.24), (5.27) by a suitable method like the finite element method we get, for $n = 1, 2, 3, \ldots$

Step 1:

$$Au_1^n = f_1 \ in \ \Omega_1 \tag{5.30}$$

$$u_1^n = g_1 \ \ on \ \partial\Omega_1 \setminus \Gamma_1 \tag{5.31}$$

$$u_1^n = u_2^{n-1} \ on \, \Gamma_1 \tag{5.32}$$

Step 2:

$$Au_2^n = f_2 \ in \ \Omega_2 \tag{5.33}$$

$$u_2^n = g_2 \ \ on \ \partial\Omega_2 \setminus \Gamma_2 \tag{5.34}$$

$$u_2^n = u_1^n \ on \, \Gamma_2. \tag{5.35}$$

Let us assume that we work on matching grids Ω_1 and Ω_2 and their corresponding restriction operators are R_1, R_2, with $R_i : \Omega_i \to \Gamma_i$, $i = 1, 2$. If we assume that we have homogeneous Dirichlet boundary conditions, we can write the systems in a matrix form as

$$A = \begin{bmatrix} A_{I\Omega_1} & A_{\Gamma_1} R_1 \\ A_{\Gamma_2} R_2 & A_{I\Omega_2} \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}.$$

Moreover we get that $A = R_1^T A_1 R_1 + R_2^T A_2 R_2$. If we apply the preconditioned Richardson iteration procedure (4.3), on systems (5.30)(5.33) we can write these systems equivalently as

$$u_1^n = u_1^{n-1} + A_{I\Omega_1}^{-1}(f_1 - (A_{I\Omega_1} u_1^{n-1} + A_{\Gamma_1} R_1 u_2^{n-1})) \tag{5.36}$$

$$u_2^n = u_2^{n-1} + A_{I\Omega_2}^{-1}(f_1 - (A_{I\Omega_2} u_2^{n-1} + A_{\Gamma_2} R_2 u_1^n)). \tag{5.37}$$

Two widely used variants of the Alternating Schwarz method (5.37), are the Multiplicative and the Additive Schwarz methods. The Multiplicative Schwarz method can be seen as an iterative solver. Moreover the Multiplicative and the Additive methods can be used as parallel preconditioners for a Krylov subspace solver like GMRES. For the analysis and the convergence properties and more information on these methods we refer to [65],[69].

The multiplicative Schwarz method can be written as

$$u^{n+1/2} = u^n + R_1^T A_1^{-1} R_1 (f - Au^n) \tag{5.38}$$

$$u^{n+1} = u^{n+1/2} + R_2^T A_2^{-1} R_2 (f - Au^{n+1/2}) \tag{5.39}$$

or as a one step procedure as

$$u^{n+1} = u^n + (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2 - R_2^T A_2^{-1} R_2 A R_1^T A_1^{-1} R_1)(f - Au^n). \tag{5.40}$$

We can define the Multiplicative Schwarz preconditioner as

$$P_m^{-1} = (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2 - R_2^T A_2^{-1} R_2 A R_1^T A_1^{-1} R_1), \tag{5.41}$$

which can be applied in a Krylov subspace method, like GMRES. The multiplicative Schwarz preconditioner is not symmetric. Hence, this gives motivation to use the Additive Schwarz preconditioner, which is obtained by removing the multiplication part in (5.41) and then we have

$$P_a^{-1} = (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2). \tag{5.42}$$

Multiplicative Schwarz and Additive Schwarz preconditioners, due to their structural properties, are related to the block Jacobi and the block Gauss-Siedel preconditioners respectively.

As regards the convergence properties it is well known that for these one-level methods, [65],[69], the number of iterations grows proportionally to $1/H$, where H is the Euclidean diameter of the subdomains. Which clearly indicates that these methods do not scale weakly (as $H \to 0$, $\frac{1}{H} \to \infty$).

Moreover, another important property of these methods is, that the convergence rate vastly improves as the overlap δ region increases in size, [65]. This behaviour is expected since if with think in terms of projection methods, this is equivalent to increasing the dimension of the search spaces, something that naturally leads to faster approximations. In order for these methods to scale weakly, we need the numbers of iterations to remain bounded as we increase the number of subdomains and the size of the problem, [65].

To make our methods scale weakly we need to add a coarse grid correction. As mentioned in [65], for the case that we have an elliptic PDE like (5.2), the solution at a point x_0 depends strongly on the value of the load function f at points close to x_0. In one level methods the information about values of f is not transferred fast enough between the subdomains. In higher level methods where we have introduced a coarse grid correction, we essentially interpolate between the coarse and the fine grid such

that the information is spread faster between the subdomains.

Optimized Schwarz method

Following the notation in [67], let $\mathcal{L}$ be an elliptic operator and B the boundary operator on Ω. We define the steady state elliptic problem

$$\mathcal{L}\tilde{u} = \tilde{f} \ \ in \ \ \Omega \tag{5.43}$$

$$B\tilde{u} = \tilde{g} \ on \ \ \partial\Omega \tag{5.44}$$

which after discretisation, by the finite element method, leads to an algebraic system of the form

$$\overline{A}u = f. \tag{5.45}$$

Then if we apply the preconditioned Richardson equation, on system (5.45) we get

$$u^{n+1} = u^n + M^{-1}(f - \overline{A}u^n)$$

where the M, for example, can be the Additive or the Multiplicative Schwarz preconditioner, (5.41), (5.42).

The Optimized Schwarz method at the continuous level for the case of two non-overlapping subdomains Ω_1, Ω_2 and artificial interface $\Gamma = (\partial\Omega_1 \cup \partial\Omega_2) \setminus \partial\Omega$, can be written as follows, for $n = 1, 2, 3 \ldots$

Step 1:

$$\mathcal{L}\tilde{u}_1^{n+1} = \tilde{f}_1 \text{ in } \Omega_1 \tag{5.46}$$

$$Bu_1^{n+1} = \tilde{g}_1 \ \text{ on } \partial\Omega \cap \partial\Omega_1 \tag{5.47}$$

$$\overline{B}_{12}\tilde{u}_1^{n+1} = \overline{B}_{12}\tilde{u}_2^n \text{ on } \Gamma \tag{5.48}$$

Step 2:

$$\mathcal{L}\tilde{u}_2^{n+1} = \tilde{f}_2 \ in \ \ \Omega_2 \tag{5.49}$$

$$B\tilde{u}_2^{n+1} = \tilde{g}_2 \ \ on \ \partial\Omega \cap \partial\Omega_2 \tag{5.50}$$

$$\overline{B}_{21}\tilde{u}_2^{n+1} = \overline{B}_{21}\tilde{u}_1^n \ on \ \Gamma \tag{5.51}$$

In the optimized Schwarz methods, as in 2-Lagrange multiplier methods, the Dirichlet boundary conditions on the interface Γ are replaced by Robin boundary conditions, $\overline{B}_{12}$, $\overline{B}_{21}$, with parameters that are tuned in a way that optimal convergence properties

are obtained, cf. [29]. We will prove the equivalence between the Optimized Schwarz methods and the 2-Lagrange Multiplier methods in the case of two non-overlapping subdomains.

In order to be able to prove the equivalence of the OSM and the 2-Lagrange multiplier methods, we introduce the Optimized Schwarz method in the 2-Lagrange multiplier framework for the Poisson problem (5.2). We have

$$-\Delta u_i^k = f \ \ in \ \ \Omega_i \tag{5.52}$$

$$(a + D_{ni})u_i^k = \lambda \ \text{ on } \ \Gamma_i \ \text{ with } \ \lambda = (a + D_{n3-i})u_{3-i}^{k-1} \tag{5.53}$$

$$u_i^k = 0 \ \ on \ \ \partial\Omega_i \cap \partial\Omega. \tag{5.54}$$

Assume $u_i^k \in H_0^1(\Omega) \cap H^1(\Omega_i)$, the weak formulation for problem (5.52) reads

$$\int_{\Omega_i} \phi f d\Omega = \int_{\Omega_i} -\phi \Delta u_i^k d\Omega_i \tag{5.55}$$

for all $\phi \in H_0^1(\Omega) \cap H^1(\Omega_i)$. From Green's formula we have that

$$\int_{\Omega_i} -\phi \Delta u_i^k d\Omega_i = \int_{\Omega_i} \nabla\phi \nabla u_i^k d\Omega_i - \int_{\partial\Omega_i} \phi D_n u_i^k d\ell.$$

Then we use that $\partial\Omega_i$ can be split into $(\partial\Omega_i \cap \partial\Omega) \cup \Gamma_i$ and from standard measure theory and (5.54), we get that

$$\int_{\partial\Omega_i} \phi D_{ni} u_i^k d\ell = \int_{\partial\Omega_i \cap \partial\Omega} \phi D_{ni} u_i^k d\ell + \int_{\Gamma_i} \phi D_{ni} u_i^k d\ell.$$

From (5.53), we get $D_{ni}u_i^k = \lambda - a u_i^k$ and hence

$$\int_{\Gamma_i} \phi D_{ni} u_i^k d\ell = -a \int_{\Gamma_i} u_i^k \phi d\ell + \int_{\Gamma_i} \lambda\phi\ell.$$

This leads to

$$\int_{\Omega_i} -\phi \Delta u_i^k d\Omega_i = \int_{\Omega_i} \nabla\phi \nabla u_i^k d\Omega_i + a \int_{\Gamma_i} u_i^k \phi d\ell - \int_{\Gamma_i} \lambda\phi\ell. \tag{5.56}$$

Finally from (5.55) and (5.56), we get the weak formulation of (5.52) as

$$\int_{\Omega_i} \phi f d\Omega + \int_{\Gamma_i} \lambda\phi d\ell = \int_{\Omega_i} \nabla\phi \nabla u_i^k d\Omega_i + a \int_{\Gamma_i} u_i^k \phi d\ell. \tag{5.57}$$

From (5.53) and since $Dn_i = -Dn_{3-i}$ we get

$$\int_{\Gamma_i} \lambda\phi d\ell = \int_{\Gamma_i} (a + Dn_{3-i}) u_{3-i}^{k-1} \phi d\ell \tag{5.58}$$

$$= a \int_{\Gamma_i} u_{3-i}^{k-1} \phi d\ell - \int_{\Gamma_i} Dn_i u_{3-i}^{k-1} \phi d\ell. \tag{5.59}$$

In order to eliminate the dual variable λ we start from the following,

$$\int_{\Omega_i^c} -\Delta u_{3-i}^{k-1} \phi d\Omega_i^c = \int_{\Omega_i^c} f\phi d\Omega_i^c,$$

$$\int_{\Omega_i^c} -\Delta u_{3-i}^{k-1} \phi d\Omega_i^c = \int_{\Omega_i^c} \nabla\phi\nabla u_{3-i}^{k-1} d\Omega_i^c - \int_{\Gamma_i} Dn_{3-i} u_{3-i}^{k-1} \phi d\ell.$$

Then from (5.58) we get that

$$\int_{\Omega_i^c} -\Delta u_{3-i}^{k-1} \phi d\Omega_i^c = \int_{\Omega_i^c} \nabla\phi\nabla u_{3-i}^{k-1} d\Omega_i^c + \int_{\Gamma_i} Dn_i u_{3-i}^{k-1} \phi d\ell,$$

$$\int_{\Omega_i^c} f\phi d\Omega_i^c = \int_{\Omega_i^c} \nabla\phi\nabla u_{3-i}^{k-1} d\Omega_i^c + \int_{\Gamma_i} Dn_i u_{3-i}^{k-1} \phi d\ell,$$

$$\int_{\Gamma_i} \lambda\phi d\ell = a \int_{\Gamma_i} \phi u_{3-i}^{k-1} d\ell + \int_{\Omega_i^c} f\phi d\Omega_i^c - \int_{\Omega_i^c} \nabla\phi\nabla u_{3-i}^{k-1} d\Omega_i^c. \tag{5.60}$$

If we substitute (5.60) back into (5.57), we get

$$\int_{\Omega_i} \phi f \partial\Omega + a \int_{\Gamma_i} \phi u_{3-i}^{k-1} d\ell - \int_{\Omega_i^c} \nabla\phi\nabla u_{3-i}^{k-1} d\Omega_i^c = \int_{\Omega_i} \nabla\phi\nabla u_i^k d\Omega_i + a \int_{\Gamma_i} u_i^k \phi d\ell. \tag{5.61}$$

5.3 Algebraic form of OSM

In this section write (5.61) in an algebraic form. Assume that

$$V_{hi} = span\{\phi_i\} \subset H_0^1(\Omega) \cap H^1(\Omega_i),$$

where

$$V_h \subset H_0^1(\Omega) \ \ and \ \ V_{hi} = V_h|_{\Omega_i}.$$

We use the finite element method in order to discretise each component of (5.61). We start from

$$\int_{\Omega_i} \nabla\phi\nabla u_i^k d\Omega_i + a \int_{\Gamma_i} u_i^k \phi d\ell$$

which becomes

$$\Big(A_{Ni} + a\begin{bmatrix}0 & 0\\ 0 & B_i\end{bmatrix}\Big)u_i^k.$$

Moreover we can rewrite $\int_{\Omega_i^c}\nabla\phi\nabla u_{3-i}^{k-1}$ as

$$-\int_{\Omega_i^c}\nabla\phi\nabla u_{3-i}^{k-1}d\Omega_i^c = -\int_{\Omega}\nabla\phi\nabla\tilde{u}_{3-i}^{k-1}d\Omega + \int_{\Omega_i}\nabla\phi\nabla\tilde{u}_{3-i}^{k-1}d\Omega_i$$

and after the discretisation becomes

$$-R_iAR_{3-i}^Tu_{3-i}^{k-1} + A_{Ni}R_iR_{3-i}^Tu_{3-i}^{k-1}.$$

Finally,

$$\int_{\Omega}\phi f d\Omega + a\int_{\Gamma_i}\phi u_{3-i}^{k-1}d\ell$$

becomes

$$R_iF + a\begin{bmatrix}0 & 0\\ 0 & B_i\end{bmatrix}R_iR_{3-i}^Tu_{3-i}^{k-1}.$$

Hence we get the OSM method in an algebraic form as

$$\Big(A_{Ni}+a\begin{bmatrix}0 & 0\\ 0 & B_i\end{bmatrix}\Big)u_i^k = R_iF+a\begin{bmatrix}0 & 0\\ 0 & B_i\end{bmatrix}R_iR_{3-i}^Tu_{3-i}^{k-1}-A_{Ni}R_iR_{3-i}^Tu_{3-i}^{k-1}+R_iAR_{3-i}^Tu_{3-i}^{k-1}. \tag{5.62}$$

From Proposition 3.2.11 we see that the mass matrix B_i is spectrally equivalent to the identity and hence we can set $B_i = I$ in order to get

$$\Big(A_{Ni}+a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}\Big)u_i^k = R_iF+a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}R_iR_{3-i}^Tu_{3-i}^{k-1}-A_{Ni}R_iR_{3-i}^Tu_{3-i}^{k-1}+R_iAR_{3-i}^Tu_{3-i}^{k-1}. \tag{5.63}$$

Remark 5.3.1. *Assume that R_1 and R_2 and the restriction operators from Ω to subdomains Ω_1 and Ω_2 respectively then*

$$A = \sum_i R_i^TA_{Ni}R_i,$$

$$A_{N1}R_1 + R_1R_2^TA_{N2}R_2 = R_1A,$$

$$A_{N1}R_1 = R_1A - R_1R_2^TA_{N2}R_2,$$

$$R_2R_1^TA_{N1}R_1 = R_2A - A_{N2}R_2.R_1R_2^TA_{N2}R_2 = R_1A - A_{N1}R_1. \tag{5.64}$$

If we apply (5.64) to (5.63) we get,

$$\Big(A_{Ni} + a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}\Big)u_i^k = R_iF + a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}R_iR_{3-i}^Tu_{3-i}^{k-1} - R_iR_{3-i}^TA_{N3-i}u_{3-i}^k. \tag{5.65}$$

5.4 Equivalence between 2LM and OSM.

Theorem 5.4.1. *The sequence of iterates u_1^k and u_2^k produced by the OSM method* (5.63) *is the same as the sequence of iterates that are produced if we apply the damped Richardson method to the non-symmetric 2-Lagrange multiplier system* (5.18)*, ($\omega = 2$).*

Proof. In the 2-Lagrange multiplier methods notation, we define the systems of equations of the Poisson problem for 2-subdomains with Robin data by

$$\Big(\overbrace{\begin{bmatrix}A_{II1} & A_{I\Gamma1}\\ A_{\Gamma I1} & A_{\Gamma\Gamma1}\end{bmatrix}}^{A_{N1}} + a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}\Big)\overbrace{\begin{bmatrix}v_{I1}^k\\ v_{\Gamma1}^k\end{bmatrix}}^{v_1^k} = \begin{bmatrix}f_{I1}\\ f_{\Gamma1}\end{bmatrix} + \begin{bmatrix}0\\ \lambda_1^k\end{bmatrix}, \tag{5.66}$$

$$\Big(\overbrace{\begin{bmatrix}A_{II2} & A_{I\Gamma2}\\ A_{\Gamma I2} & A_{\Gamma\Gamma2}\end{bmatrix}}^{A_{N2}} + a\begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}\Big)\overbrace{\begin{bmatrix}v_{I1}^{k-1}\\ v_{\Gamma2}^{k-1}\end{bmatrix}}^{v_2^{k-1}} = \begin{bmatrix}f_{I2}\\ f_{\Gamma2}\end{bmatrix} + \begin{bmatrix}0\\ \lambda_2^{k-1}\end{bmatrix}. \tag{5.67}$$

In the case of 2 non-overlapping subdomains the averaging operator K has the form

$$K = \frac{1}{2}\begin{bmatrix}I & I\\ I & I\end{bmatrix}.$$

In the 2-Lagrange multiplier methods matrix Q as defined in (5.14) is

$$Q_k = a(S_k + aI)^{-1}, \text{ for } k = 1, 2.$$

Moreover we have seen that the non-symmetric Lagrange multiplier system is given by

$$(I - 2K)(Q - K)\lambda = -(I - 2K)Qg. \tag{5.68}$$

Hence for two subdomains the right side of (5.68) becomes

$$(I - 2K)(Q - K) = \Big(\begin{bmatrix}I & 0\\ 0 & I\end{bmatrix} - \begin{bmatrix}I & I\\ I & I\end{bmatrix}\Big)\Big(\begin{bmatrix}Q_1 & 0\\ 0 & Q_2\end{bmatrix} - \frac{1}{2}\begin{bmatrix}I & I\\ I & I\end{bmatrix}\Big)$$

$$= \begin{bmatrix} \frac{1}{2}I & -(Q_2 - \frac{1}{2}I) \\ -(Q_1 - \frac{1}{2}I) & \frac{1}{2}I \end{bmatrix} = \frac{1}{2} \begin{bmatrix} I & I - 2a(S_2 + aI)^{-1} \\ I - 2a(S_1 + aI)^{-1} & I \end{bmatrix}.$$

The relaxed Richardson method for arbitrary system $Ax = b$ is

$$\lambda_{k+1} = \lambda_k + 2(b - A\lambda_k).$$

Let $b = -(I - 2K)Qg$, then for the case of system (5.68), the relaxed Richardson method takes the form,

$$\lambda_{k+1} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \lambda_k + \Big(2b - \begin{bmatrix} I & I - 2a(S_2 + aI)^{-1} \\ I - 2a(S_1 + aI)^{-1} & I \end{bmatrix} \lambda_k\Big),$$

equivalent to

$$\begin{bmatrix} \lambda_1^{k+1} \\ \lambda_2^{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 2a(S_2 + aI)^{-1} - I \\ 2a(S_1 + aI)^{-1} - I & 0 \end{bmatrix} \begin{bmatrix} \lambda_1^k \\ \lambda_2^k \end{bmatrix} + 2 \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \tag{5.69}$$

From the right side of (5.68) we get

$$b = -(I - 2K)Qg = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} g = \begin{bmatrix} 0 & Q_2 \\ Q_1 & 0 \end{bmatrix} g,$$

end so

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} Q_2 g_2 \\ Q_1 g_1 \end{bmatrix}.$$

Hence the b_1, b_2 in (5.69) can be written as,

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a(S_2 + aI)^{-1} g_2 \\ a(S_1 + aI)^{-1} g_1 \end{bmatrix}. \tag{5.70}$$

Finally, we can rewrite (5.69) as a system of equations

$$\lambda_1^{k+1} = 2a(S_2 + aI)^{-1}\lambda_2^k - \lambda_2^k + 2b_1, \tag{5.71}$$

$$\lambda_2^{k+1} = 2a(S_1 + aI)^{-1}\lambda_1^k - \lambda_1^k + 2b_2. \tag{5.72}$$

Moreover, from (5.13), we have that

$$(S + aI)u_G = g + \lambda, \tag{5.73}$$

$$u_G = (S + aI)^{-1}(g + \lambda), \tag{5.74}$$

where

$$g_k = f_{\Gamma k} - A_{\Gamma I} k A_{II k}^{-1} f_{Ik} \tag{5.75}$$

and

$$S_k = A_{\Gamma\Gamma_k} - A_{\Gamma Ik} A_{IIk}^{-1} A_{I\Gamma k}. \tag{5.76}$$

Hence, from (5.73) we get

$$u_{\Gamma_1}^{k+1} = (S_1 + aI)^{-1}(\lambda_1^{k+1} + g_1), \tag{5.77}$$

$$u_{\Gamma_1}^{k+1} = (S_1 + aI)^{-1}\lambda_1^{k+1} + (S_1 + aI)^{-1} g_1, \tag{5.78}$$

$$\lambda_1^{k+1} = (S_1 + aI) u_{\Gamma_1}^{k+1} - g_1 \tag{5.79}$$

and in the same manner

$$u_{\Gamma_2}^{k} = (S_2 + aI)^{-1}(\lambda_2^{k} + g_2), \tag{5.80}$$

$$u_{\Gamma_2}^{k} - (S_2 + aI)^{-1} g_2 = (S_2 + aI)^{-1}\lambda_2, \tag{5.81}$$

$$\lambda_2^{k} = (S_2 + aI) u_{\Gamma_2}^{k} - g_2. \tag{5.82}$$

If we substitute (5.82) into (5.71) we get

$$\lambda_1^{k+1} = 2a u_{\Gamma_2}^{k} - 2a(S_2 + aI)^{-1} g_2 - ((S_2 + aI) u_{\Gamma_2}^{k} - g_2) + 2b_1. \tag{5.83}$$

Moreover from (5.79) and (5.83) we have that

$$(S_1 + aI) u_{\Gamma_1}^{k+1} - g_1 = 2a u_{\Gamma_2}^{k} - 2a(S_2 + aI)^{-1} g_2 - (S_2 + aI) u_{\Gamma_2}^{k} + g_2 + 2b_1. \tag{5.84}$$

From (5.66) and (5.67) we have that the solutions that correspond to the interior nodes, u_{I1} and u_{I2}, satisfy the following equations:

$$A_{II1} u_{I1}^{k+1} + A_{I\Gamma 1} u_{\Gamma_1}^{k+1} = f_{I1}; \tag{5.85}$$

$$A_{II2} u_{I2}^{k} + A_{I\Gamma 2}^{k} u_{\Gamma 2}^{k} = f_{I2}. \tag{5.86}$$

Hence from (5.76) and (5.85), (5.86) we get that

$$S_1 u_{\Gamma 1}^{k+1} = A_{\Gamma\Gamma_1} u_{\Gamma_1}^{k+1} - A_{\Gamma I1} A_{II1}^{-1} f_{I1} + A_{\Gamma I1} u_{I1}^{k+1} \tag{5.87}$$

and

$$S_2 u_{\Gamma 2}^{k} = A_{\Gamma\Gamma_2} u_{\Gamma_2}^{k} - A_{\Gamma I2} A_{II2}^{-1} f_{I2} + A_{\Gamma I2} u_{I2}^{k}. \tag{5.88}$$

If we substitute (5.87), (5.88) into (5.84) we get

$$A_{\Gamma\Gamma_1}u_{\Gamma_1}^{k+1} - A_{\Gamma I1}A_{II1}^{-1}f_{I1} + A_{\Gamma I1}u_{I1}^{k+1} + au_{\Gamma_1}^{k+1} - g_1$$

$$= 2au_{\Gamma_2}^{k} - 2a(S_2+aI)^{-1}g_2 - A_{\Gamma\Gamma_2}u_{\Gamma_2}^{k} + A_{\Gamma I2}A_{II2}^{-1}f_{I2} - A_{\Gamma I2}u_{I2}^{k} - au_{\Gamma_2}^{k} + g_2 + 2b_1. \tag{5.89}$$

Then if we substitute g_1 and g_2 form (5.75) into (5.89) we derive

$$A_{\Gamma\Gamma_1}u_{\Gamma_1}^{k+1} + A_{\Gamma I1}u_{I1}^{k+1} + au_{\Gamma_1}^{k+1} =$$

$$au_{\Gamma_2}^{k} - 2a(S_2+aI)^{-1}f_{\Gamma 2} + 2a(S_2+aI)^{-1}(A_{\Gamma I2}u_{I2}^{k}$$

$$+A_{\Gamma I2}A_{II2}^{-1}A_{I\Gamma 2}^{k}u_{\Gamma 2}^{k}) - A_{\Gamma\Gamma_2}u_{\Gamma_2}^{k} - A_{\Gamma I2}u_{I2}^{k} + f_{\Gamma 1} + f_{\Gamma 2} + 2b_1. \tag{5.90}$$

Moreover, from (5.66) we have that

$$A_{II1}u_{I1}^{k+1} + A_{I\Gamma 1}u_{\Gamma_1}^{k+1} = f_{I1}. \tag{5.91}$$

Finally, from (5.90) and (5.91) we get the equivalence between the non-symmetric 2-Lagrange multiplier method and OSM,

$$\Big(\overbrace{\begin{bmatrix} A_{II1} & A_{I\Gamma 1} \\ A_{\Gamma I1} & A_{\Gamma\Gamma 1}\end{bmatrix}}^{A_{N1}} + a\begin{bmatrix} 0 & 0 \\ 0 & I\end{bmatrix}\Big)\overbrace{\begin{bmatrix} u_{I1}^{k+1} \\ v_{\Gamma 1}^{k+1}\end{bmatrix}}^{u_1^{k+1}} = a\begin{bmatrix} 0 & 0 \\ 0 & I\end{bmatrix}R_1R_2^Tu_2^k - R_1R_2^TA_{N2}u_2^k + R_iF. \tag{5.92}$$

In order to derive the right side of equation (5.92) we have used the following

Remark 5.4.2.

$$-2a(S_2+aI)^{-1}f_{\Gamma 2} - 2a(S_2+aI)^{-1}(A_{\Gamma I2}u_{I2}^{k} + A_{\Gamma I2}A_{II2}^{-1}(f_{I2} - A_{II2}u_{I2}^{k}) + f_{\Gamma 1} + f_{\Gamma 2} + 2b_1 \tag{5.93}$$

$$= -2a(S_2+aI)^{-1}f_{\Gamma 2} + 2a(S_2+aI)^{-1}A_{\Gamma I2}A_{II2}^{-1}f_{I2} + 3f_{\Gamma 1} + f_{\Gamma 2}$$

$$-2a(S_2+aI)^{-1}(f_{\Gamma 2} - A_{\Gamma I2}A_{II2}^{-1}f_{I2}) + f_{\Gamma 1} + f_{\Gamma 2} + 2b_1$$

If we use (5.70) *to substitute* b_1 *we get,*

$$= -2a(S_2+aI)^{-1}g_2 + f_{\Gamma 1} + f_{\Gamma 2} + 2a(S_2+aI)^{-1}g_2 \tag{5.94}$$

$$= f_{\Gamma 1} + f_{\Gamma 2} = f_{\Gamma}.$$

□

5.5 Weak scalability of the 2-Level 2-Lagrange multiplier methods

In this section we prove the weak scalability of the 2-Lagrange multiplier methods. An important element of the 2-Lagrange multiplier methods and in general for the most non-overlapping Domain Decomposition methods is the Schur complement. The Schur complement is considered as the discrete equivalent of the Steklov-Poincaré operator [2]. Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ and $\Omega = \cup\Omega_i$, $i = 1, \ldots, p$, we define the subdomains Ω_i as floating subdomains if $\partial\Omega \cap \partial\Omega_i = \emptyset$. An important issue that arises in non-overlapping methods is that the Schur complement for the case of floating subdomains is singular. The advantage of the 2-Lagrange multiplier methods is that the Robin-boundary conditions that we imposed on the artificial interface helps us to remove this singularity in a very simple and efficient way.

We start by some important properties of the Schur complement and then we prove that, under some assumptions, the condition number of the Schur compliment for the case of elliptic problems with coercive and self-adjoint weak formulation, is bounded by $O(\frac{H}{h})$. Then we use this result in order to prove the scalability of the methods.

The main contribution of this work, is that we provide a coarse grid correction preconditoner for the 2-Lagrange multiplier methods, which makes the methods scalable and opens the doors for the massively parallel implementation of these methods. In order to prove the scalability, we provide the condition number estimates for the symmetric and the non-symmetric 2-Lagrange Multiplier methods and we explore experimentally the convergence properties by using the GMRES solver [59].

In this chapter we provide two sets of experiments that support the theoretical findings. Large scale experiments and the implementation of the methods in C/C++ with the use of parallel libraries like `MPI` and $PETSc$ are provided in Chapter 6.

5.5.1 Schur complement properties

The left side of the discretised system (5.3) can be written as

$$A = \sum_{i=0}^{p} R_i^T A_{Ni} R_i, \tag{5.95}$$

where R_i are the restriction binary matrices which restrict a n-dimensional vector v, from the whole domain Ω, to an n_i-dimensional vector $R_i v$, containing only the vertices that belong to subdomain Ω_i. Moreover the local Neumman matrices can be

written as

$$A_{N_i} = \begin{bmatrix} A_{II i} & A_{I\Gamma i} \\ A_{\Gamma I i} & A_{\Gamma\Gamma i} \end{bmatrix}. \tag{5.96}$$

Moreover let A be the global matrix after the discretization of (5.2). Then A can be written in the form

$$A = \begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_I \\ f_\Gamma \end{bmatrix}. \tag{5.97}$$

Before we proceed to the analysis of our methods we would like to make some useful comments on the Schur complement, i.e invertability, condition number etc., in order to understand better the properties of the 2-Lagrange methods.

If we use block Gaussian Elimination in order to eliminate the interior degrees of freedom we get that $u_I = A_{II}^{-1}(f_I - A_{IB}u_\Gamma)$. Substituting back to row 2 of (5.97) we get the system on the artificial interface

$$Su_\Gamma = g_\Gamma,$$

where $S = A_{\Gamma\Gamma} - A_{\Gamma I}A_{II}^{-1}A_{I\Gamma}$ and $g_\Gamma = f_\Gamma - A_{\Gamma I}A_{II}^{-1}f_I$.

Proposition 5.5.1. *Let $u_h = (u_I, u_\Gamma) \in \mathbb{R}^n$ on Ω. It is said to be discrete harmonic if*

$$A_{II}u_I + A_{I\Gamma}u_\Gamma = 0$$

and

$$\begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_\Gamma \end{bmatrix} \tag{5.98}$$

for $\lambda_\Gamma \in \mathbb{R}^m$ where m is the number of dofs on the artificial interface Γ. Then the Schur complement energy satisfies

$$s(u,v) = u_\Gamma^T S u_\Gamma = \begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix}^T \begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix}. \tag{5.99}$$

Proof. Let u_h be discrete harmonic then $u_I = -A_{II}^{-1}A_{I\Gamma}u_\Gamma$, the right side of (5.99) becomes

$$\begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix}^T \begin{bmatrix} A_{II}u_I + A_{I\Gamma}u_\Gamma \\ -A_{\Gamma I}A_{II}^{-1}A_{I\Gamma}u_\Gamma + A_{\Gamma\Gamma}u_\Gamma \end{bmatrix} =$$

$$\begin{bmatrix} u_I \\ u_\Gamma \end{bmatrix}^T \begin{bmatrix} 0 \\ -A_{\Gamma I}A_{II}^{-1}A_{I\Gamma}u_\Gamma + A_{\Gamma\Gamma}u_\Gamma \end{bmatrix} = u_\Gamma^T S u_\Gamma.$$

□

Remark 5.5.2. *In the case of a floating subdomain (see Fig.* 5.3*) the local Neumman problems* (5.96) *are singular with*

$$null(A_{Ni}) = \{(1, 1, \ldots, 1)^T \in R^n\}$$

and that same is true for the corresponding Schur complement

$$null(S_i) = \{(1, 1, \ldots, 1)^T \in R^m\}.$$

Hence the addition of the Robin parameter $a > 0$ is essential to guarantee the invertibility of the local Schur complements S_k and the local Robin subproblems (5.8).

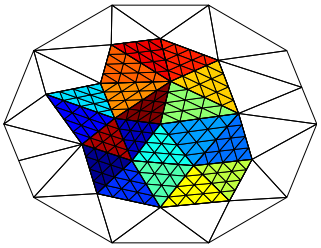


Figure 5.3: Floating subdomains

Definition 5.5.3. *Let Ω be a Lipschitz domain with non-overlapping subdomains Ω_i, $i = 1 \ldots p$. Moreover, let A be symmetric and positive definite. The resulting Schur complement is positive semi-definite. We define $s_{min} > 0$ as the smallest non-zero eigenvalue of S, and s_{max} the largest eigenvalues of S. We define the condition number $\kappa_0(S) = \frac{s_{max}}{s_{min}}$. Finally we define the optimal Robin parameter as $a_{opt} = \sqrt{s_{max} s_{min}}$.*

Lemma 5.5.4 ([69])**.** *Let u_h be discrete harmonic for each Ω_i. Then there exist constants c, C independent of the grid parameter h and the diameter of the subdomains, such that*

$$c|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \leq |u_h|^2_{H^1(\Omega_i)} \leq C|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \tag{5.100}$$

and hence

$$c|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \leq u_{\Gamma_i} S_i u_{\Gamma_i} \leq C|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i \cap \Gamma)} \tag{5.101}$$

where u_{Γ_i} is the restriction of the finite element trace of u_h on $\partial\Omega_i \cap \Gamma$.

Now we are going to estimate the condition number of the Schur complement for regular domains.

Lemma 5.5.5. *We have a regular domain decomposition when the following properties hold. Let Ω be a domain of unit diameter, T_h be a quasi uniform triangulation*

with $h > 0$ and the typical subdomain size $H > 0$. In addition the following (1)-(4) properties hold.

1. *Assume that $\Omega_1, \ldots, \Omega_p$ are polygons or polyhedra of a globally conforming mesh T_h with diameter $H_i < H$.*

2. *For $i = 1, \ldots p$, either Ω_i is a floating subdomain or the size of the intersection of $\partial\Omega_i$ with the measure of $\partial\Omega$ is comparable to $\partial\Omega_i$.*

3. *The matrix A is the finite element discretization of the bilinear form*

$$a(u, v) = \int_\Omega a(x)\nabla u(x)\nabla v(x)dx$$

and where

$$A_{ij} = \int_K a(x)\nabla\phi_i\nabla\phi_j(x)dx.$$

4. *The inverse inequality holds for all u_h in the finite element basis V_h,*

$$|u_{\Gamma_i}|_{H^{1/2}(\partial\Omega_i)} \leq \frac{c_1}{\sqrt{h}}||u_{\Gamma_i}||_{L^2(\partial\Omega_i)}.$$

For regular domains there is constant $\tilde{C}$ which depends on the shape of the domain Ω and the shape of the non-overlapping subdomains Ω_i, $i = 1 \ldots p$ but not on the grid parameters h, H, such that the following inequality holds,

$$\kappa_0(S) \leq \tilde{C}\frac{H}{h}. \tag{5.102}$$

Proof. The Sobolev space $H^1(\Omega)$ and $H^{1/2}(\Omega)$ have been defined in Chapter 2 over subdomains with diameter $H = 1$. By using a simple dilation, as in Chapter 4 [69], we can replace the domains Ω_i by the subodmains $\frac{1}{hi}\Omega_i$. Assume that $u_h = (u_I, u_\Gamma)$ defined on Ω_i, is discrete harmonic, with u_Γ the trace on $\partial\Omega_i$. Then from the assumption 4 and [69, Lemma A.17], we get that

$$|u_{\Gamma_i}|_{H^{1/2}(\partial\Omega_i)} \leq \frac{c_1}{\sqrt{h}}||u_{\Gamma_i}||_{L^2(\partial\Omega_i)}, \tag{5.103}$$

$$||u_{\Gamma_i}||_{L^2(\partial\Omega_i)} \leq c_2|u_{\Gamma_i}|_{H^{1/2}(\partial\Omega_i)}. \tag{5.104}$$

From 5.101 we get that

$$c|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i\cap\Gamma)} \leq u_{\Gamma_i}S_iu_{\Gamma_i} \leq C|u_\Gamma|^2_{H^{1/2}(\partial\Omega_i\cap\Gamma)}.$$

Hence combining (5.103), (5.104) and (5.101) we get the following,

	h				
$\kappa_0(S)$	0.062500	0.031250	0.015625	0.007813	0.003906
$H = 0.5000$	22.033364	44.942523	90.349765	180.938463	361.998007
$H = 0.2500$	18.878024	39.820895	80.985800	162.722751	325.842387
$H = 0.1250$			39.820895	80.985800	162.722751
$H = 0.0625$				39.820895	80.985800

$$\frac{c}{c_2^2}||u_{\Gamma_i}||_{L^2(\partial\Omega_i} \leq u_{\Gamma_i} S_i u_{\Gamma_i} \leq \frac{Cc_1^2}{h}||u_{\Gamma_i}||_{L^2(\partial\Omega_i}.$$

Since u_{Γ_i} is the trace of a finite element function u_h on Γ and from the properties of the mass matrix we get that that $||u_{\Gamma_i}||_{L^2(\partial\Omega_i)}$ will be equivalent up to a scaling factor to the Euclidean norm of u_h restricted to Γ_i. Therefore, the estimate (5.102) holds. □

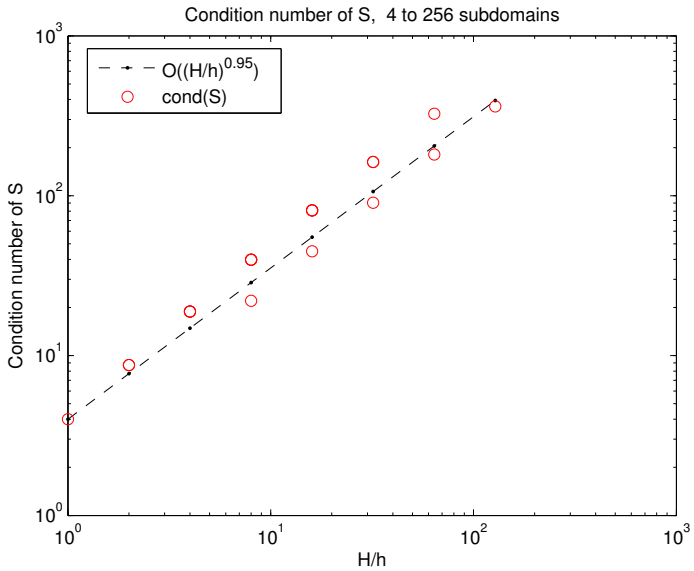


Figure 5.4: Loglog plot of the condition number $\kappa_0(S)$

Remark 5.5.6. *The spectrum of matrix Q, defined in* (5.14)*, can be written as*

$$\sigma(Q) = \frac{\alpha}{\sigma(S)+\alpha} = \left\{\frac{\alpha}{z+\alpha} \ : \ s = 0, s_{min}, \ldots, s_{max}\right\},$$

if we set $q = \frac{\alpha}{z+\alpha}$, then q attains the following values:

- *If $z = 0$, then $q = 1$.*
- *If $z > 0$, then $q \in (0, 1)$.*
- *If $z < s_{max}$, then $q \geq \frac{\alpha}{s_{max}+\alpha}$ and if we set $a = a_{opt}$, see Definition 5.5.3, $q \geq \frac{\sqrt{s_{min}s_{max}}}{s_{max}s+\sqrt{s_{min}s_{max}}}$.*

Hence, $K(Q) = \frac{s_{max}+\sqrt{s_{max}s_{min}}}{\sqrt{s_{max}s_{min}}} = \sqrt{\kappa_o(S)} + 1.$

5.5.2 Methods that scale weakly

Proposition 5.5.7 (The matrix A_{S2LM})**.** *Let* $\epsilon > 0$*. Assume that the matrix* Q *is symmetric and positive definite, with the spectrum* $\sigma(Q) \subset [\epsilon, 1-\epsilon] \cup \{1\}$*,* $0 < \epsilon < \frac{1}{2}$*. Also assume that* K *is an orthogonal projection.*

From Lemma 5.5.5 and Remark 5.5.6

$$\frac{1}{\epsilon} = O\left(\sqrt{\frac{H}{h}}\right), \tag{5.105}$$

where H is the Euclidean diameter of a typical subdomain Ω_k, and h is the diameter of the fine grid discretizing (5.2). Therefore, ϵ is a quantity that "scales" in the sense that if we increase the parallelism by shrinking h while keeping H/h bounded, the quantity ϵ remains bounded away from 0.

Definition 5.5.8 (Weak scaling)**.** *We say that a method* **scales weakly** *if the condition number depends only on* ϵ *and not on the spectral norm* $\|EK\|$ *(the spectral norm is the largest singular value).*

In domain decomposition, the definition of a method that scales weakly is one where the condition number depends on the ratio H/h, but not on h or H individually. Thus, our definition of weak scaling is justified by the case considered in [48] with ϵ given by (5.105). We now present methods that scale weakly.

Remark 5.5.9. *We now briefly discuss how the condition number of* $Q - K$ *grows unboundedly if* $\|EK\| \to 1$*.*

Recall that E *is an orthogonal projection. After a suitable orthogonal change of basis, we may assume that* $E = \begin{bmatrix} O & O \\ O & I \end{bmatrix}$*. In this basis, we have that* $Q = \begin{bmatrix} Q_1 & \\ & I \end{bmatrix}$*, since* E *is the orthogonal projection onto the kernel of* $I - Q$*. As a result, the matrix* A_{S2LM} *has the form*

$$A_{S2LM} = Q - K = \begin{bmatrix} Q_1 - K_{11} & K_{12} \\ K_{21} & I - K_{22} \end{bmatrix}.$$

Note that K *is symmetric and hence* $UK_{22}U^T$ *is diagonal for a suitable orthogonal matrix* U*. Conjugating* $Q - K$ *by the matrix* $\begin{bmatrix} I & \\ & U \end{bmatrix}$ *if necessary, we may assume that* K_{22} *is diagonal. If the spectral radius* $\rho(K_{22}) = 1$ *then one of the diagonal entries must be one. For simplicity, assume that this is the bottom-right entry of* K_{22}*.*

If an orthogonal projection (such as K) has a one on its diagonal, then the corresponding row and column must have zeros everywhere else. Thus, the entire last row and column of $Q-K$ is 0; in other words, $Q-K$ is singular.

Thus, if $\|EK\| = \sqrt{\rho((EK)(EK)^T)} = \sqrt{\rho(EKE)} = \sqrt{\rho(K_{22})}$ approaches 1 when we refine the grid, we must have that the condition number of $A_{S2LM} = Q-K$ grows unboundedly; this shows that bounding $\|EK\|$ away from 1 is necessary for weak scaling.

Motivation for the 2-Level methods

In Section 5.1.1 we have defined the Symmetric 2-Lagrange multiplier method as

$$\overbrace{(Q-K)}^{A_{S2LM}}\lambda = -Qg \tag{5.106}$$

and the non-Symmetric 2-Lagrange multiplier method as

$$\overbrace{(I-2K)(Q-K)}^{A_{N2LM}}\lambda = -(I-2K)Qg. \tag{5.107}$$

We will prove that A_{S2LM} and A_{N2LM} methods do not scale weakly something that motivated us to develop higher level methods in order to overcame the lack of scalability. These methods are the symmetric and non-Symmetric 2-level 2-Lagrange multiplier methods. We will define and analyse the 2-level 2-Lagrange multiplier methods in detail and we will prove that they scale weakly. In Chapter 6 we include various sets of experiment that support our findings.

In order to prove that the one-level methods do not scale weakly we start from finding the condition number of the Symmetric 2-Lagrange multiplier method. The spectrum of $S+aI$ is $\sigma(S+aI) = \{z+a : z \in \sigma(S)\}$, and the matrix $Q = a(s+aI)^{-1}$ in the right side of the symmetric system (5.17) has spectrum $\sigma(Q) = \left\{\frac{a}{z+a} : z \in \sigma(S)\right\}$. Since S is positive semi-definite we get that $0 < \frac{a}{z+a} \leq 1$ and hence we have that $\sigma(Q) \subset (0,1]$. Moreover since K is an orthogonal projection we have that $\sigma(K) = \{0,1\}$ and consequently that $\sigma(Q-K) \subset [-1,1]$.

If $1 \notin \sigma(Q)$ and $0 < \epsilon \leq \frac{1}{2}$, we will see that the spectral condition number $\kappa(Q-K)$ is bounded by $(1-\epsilon)/\epsilon$. On the other hand, in the case that $1 \in \sigma(Q)$ then $\kappa(Q-K)$ also depends on $\|EK\|$, where E is the orthogonal projection onto the kernel of $I-Q$, where I is the identity matrix of appropriate size. Since the one level methods depend on $\|EK\|$ we will provide an estimate for this norm based on article [48]. Let ρ be the spectral radius $\rho = \max\{|\lambda_1, \ldots, |\lambda_n|\}$, where λ_i $i = 1, \ldots n$ are the eigenvalues of

a square matrix $n \times n$ matrix, then

$$\|EK\| = \sqrt{(\rho(EK)(EK)^T)} = \sqrt{\rho(EKE)} \tag{5.108}$$

or equivalently

$$\rho(EKE) = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & K_{22} \end{bmatrix} = \rho(K_{22}).$$

Let $\Omega \subseteq \mathbb{R}^d$, $d = 2, 3$, $\Omega = \bigcup_{k=1}^{p} \Omega_k$ and the artificial interface $\Gamma = \bigcup_{k=1}^{p} \partial\Omega_k \setminus \partial\Omega$. Then K_{22} can be written as $K_{22} = JKJ^T$, where J is a matrix whose columns form an orthogonal basis for the range of E. The range of E consists of piecewise constant many-sided traces and hence matrix J can be written explicitly as,

$$J := \text{blkdiag}(\frac{1}{\sqrt{n_{\Gamma 1}}}\mathbf{1}_{n_{\Gamma 1}}, \ldots, \frac{1}{\sqrt{n_{\Gamma p}}}\mathbf{1}_{n_{\Gamma p}}), \tag{5.109}$$

where $n_{\Gamma k}$ is the number of vertices on the artificial interface $\partial\Omega_k \cap \Gamma$. Moreover

$$\rho(K_{22}) = \rho(J^T K J) = \lambda_{max}(J^T K J) = 1 - \lambda_{min}(I - J^T K J). \tag{5.110}$$

We now estimate the smallest eigenvalue of $Z = I - J^T K J$. The estimate of this eigenvalue done in [48] is correct in spirit has some minor errors. We have corrected these minor errors below; our estimate of the smallest eigenvalue of Z is very similar, but not exactly the same as [48].

Recall that the columns of J are the indicating functions of the floating subdomains, scaled so that $J^T J = I$:

$$J = \begin{bmatrix} m_{\Gamma 1}^{-1/2} f_1 & & \\ & \ddots & \\ & & m_{\Gamma n}^{-1/2} f_n \\ & O & \end{bmatrix},$$

where $f_1, \ldots, f_n$ are suitable vectors of ones and $m_{\Gamma k} = \|f_k\|_1$ is the number of vertices on $\Gamma_k = \partial\Omega_k \cap \Gamma$. The large O at the bottom of J encompasses all subdomains that are not floating.

We now estimate the smallest eigenvalue of Z. We can embed the matrix Z into a matrix $\tilde{Z}$ as follows. Let $\tilde{J}$ denote the matrix whose columns are the indicating functions of all subdomains, floating or not, normalized so that $\tilde{J}^T \tilde{J} = I$ and let $\tilde{Z} = I - \tilde{J}^T K \tilde{J}$. In what follows, we need to be careful about the physical grid, with grid points $\{x_k\}$, versus the "Tearing and Interconnecting", where grid points along subdomains are duplicated for each subdomain. To be concrete, if we denote

by u_G some multi-valued trace on Γ, we will write $[j]$ to signify the grid point $[j] = x_k$ corresponding to $[u_G]_j$.

Lemma 5.5.10. *The entries of $\tilde{Z}_{ij}$ are*

$$\tilde{Z}_{ij} = \delta_{ij} - \frac{C_{ij}}{\sqrt{m_{\Gamma i} m_{\Gamma j}}} \text{ where } C_{ij} = \sum_{x_k \in \Gamma_i \cap \Gamma_k} \frac{1}{m_{x_k}}. \tag{5.111}$$

In the expression for C_{ij}, m_{x_k} denotes the number of subdomains adjacent to the grid vertex x_k.

Proof. Note that $\tilde{J}e_j$ is simply a column of $\tilde{J}$, which is an indicating function of a subdomain, scaled by $m_{\Gamma j}^{-1/2}$:

$$(\tilde{J}e_j)_k = \begin{cases} m_{\Gamma j}^{-1/2} & \text{if } k \text{ belongs to the subdomain } \Omega_j; \\ 0 & \text{otherwise.} \end{cases}$$

The matrix K acts on $\tilde{J}e_j$ by averaging, and hence

$$(K\tilde{J}e_j)_k = \begin{cases} m_{[k]}^{-1} m_{\Gamma j}^{-1/2} & \text{if } [k] \in \Gamma_j, \\ 0 & \text{otherwise;} \end{cases}$$

where we have denoted by $m_{[k]}$ the number of subdomains adjacent to the vertex $[k]$. Note that there are precisely $m_{[k]}$ entries of the vector λ that correspond to $[k]$ so

$$e_i^T \tilde{J} K \tilde{J} e_j = [K\tilde{J}e_i]^T [K\tilde{J}e_j] = \sum_{k \text{ s.t. } [k] \in \Gamma_i \cap \Gamma_j} m_{[k]}^{-1} m_{\Gamma i}^{-1/2} m_{[k]}^{-1} m_{\Gamma j}^{-1/2} = \sum_{x_k \in \Gamma_i \cap \Gamma_k} \frac{1}{m_{x_k} \sqrt{m_{\Gamma i} m_{\Gamma j}}},$$

as required. □

Lemma 5.5.11. *Let $D = diag(\sqrt{m_{\Gamma 1}}, \ldots, \sqrt{m_{\Gamma p}})$, where p is the total number of subdomains. $D\tilde{Z}D$ is diagonally semidominant.*

Proof.

$$(D\tilde{Z}D)_{ij} = m_{\Gamma i}\delta_{ij} - C_{ij} \text{ where } C_{ij} = \sum_{x_k \in \Gamma_i \cap \Gamma_j} \frac{1}{m_{x_k}}. \tag{5.112}$$

Since m_{x_k} is the number of subdomains adjacent to x_k, we find that

$$\sum_j C_{ij} = \sum_j \sum_{x_k \in \Gamma_i \cap \Gamma_j} \frac{1}{m_{x_k}} = \sum_{k \text{ s.t. } x_k \in \Gamma_i} \overbrace{\sum_{j \text{ s.t. } x_k \in \Gamma_j}}^{m_{x_k} \text{ terms}} \frac{1}{m_{x_k}} = m_{\Gamma i}.$$

As a result,

$$(D\tilde{Z}D)_{ii} - \sum_{j\neq i} |(D\tilde{Z}D)_{ij}| = m_{\Gamma i} - \sum_j C_{ij} = 0.$$

□

We now give a "semidefinite superposition representation" of $D\tilde{Z}D$:

$$D\tilde{Z}D = X + R,$$

where X and R are diagonally semidominant (and hence semidefinite).

Definition 5.5.12 ((L_1, ℓ) path cover)**.** *Let $\Omega_1, \ldots, \Omega_p$ be the subdomains. A* **path of length** *ℓ, $\gamma = (\gamma_1, \ldots, \gamma_\ell)$ is a subset of $\{1, \ldots, p\}$ such that, for each k,*

$$C_{\gamma_k, \gamma_{k+1}} > L_1 \min_k m_{\Gamma k} > 0. \tag{5.113}$$

In particular, Ω_{γ_k} and $\Omega_{\gamma_{k+1}}$ are adjacent. We further require that Ω_{γ_1} is a subdomain that does not float.

An (L_1, ℓ) **path cover** *$\{\gamma^{(k)}\}$ is a disjoint cover of $\{1, \ldots, p\}$ by paths of lengths $\{\ell_k\}$ at most ℓ. After relabelling the subdomains if necessary, we may assume that each path is a consecutive run of integers:*

$$\gamma^{(k)} = (q_k + 1, q_k + 2, \ldots, q_k + \ell_k),$$

and the paths are ordered $\gamma^{(1)} < \gamma^{(2)} < \ldots < \gamma^{(m)}$; in other words, $q_k = \sum_{j<k} \ell_j$. We then say that the labelling of the subdomains is **standardized by** *γ.*

The idea here is for L_1 to be a "reasonable" constant that merely captures how well-connected are neighboring subdomains. For example, if each subdomain is a quadrilateral sharing about one quarter of its boundary vertices with each adjacent subdomain, then $L_1 \approx 0.25$. On the other hand, if one of the paths in the chosen path cover $\{\gamma^{(k)}\}$ goes through a very narrow interface $\partial\Omega_i \cap \partial\Omega_j$, L_1 will have to be chosen much smaller. This means that the mesh partitioning software should avoid generating nearly degenerate domain decompositions, where one subdomain is adjacent to a very large number of subdomains but only shares a few vertices with each individual adjacent subdomain.

Lemma 5.5.13. *Let $\gamma = \{\gamma^{(1)}, \ldots, \gamma^{(m)}\}$ be an (L_1, ℓ) path cover, and that the la-*

belling of the subdomains is standardized by γ. Put

$$X = \left(L_1 \min_k m_{\Gamma k}\right) \begin{bmatrix} Y(\ell_1) & & \\ & \ddots & \\ & & Y(\ell_m) \end{bmatrix} \quad \textit{where} \tag{5.114}$$

$$Y(\ell_k) = \begin{bmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{\ell_k \times \ell_k} \tag{5.115}$$

Put $R = D\tilde{Z}D - X$. Then, R is diagonally semidominant (and hence positive semidefinite).

Proof. Set $R = D\tilde{Z}D - X$ and consider first an off-diagonal entry R_{ij}. If $|i-j| > 2$ then clearly $R_{ij} = (D\tilde{Z}D)_{ij} \leq 0$ since X is tridiagonal. Because the labels are standardized by γ, we have that $\gamma_{k+1}^{(j)} = \gamma_k^{(j)} + 1$ and hence $C_{\gamma_k^{(j)},\gamma_{k+1}^{(j)}}$ is the first superdiagonal of C; thus we may apply (5.113). Combining with (5.112) we find that, for the first superdiagonal, $R_{i,i+1} < 0$. Since R is symmetric, the first subdiagonal must also be negative. Finally, note that the row sums of both $D\tilde{Z}D$ and X are zero, hence the row sums of R are also zero. Since all the off-diagonal entries of R are nonpositive, we conclude that the diagonal of R is nonnegative, and R is diagonally dominant. □

Lemma 5.5.14. *Assume that the domain decomposition admits an (L_1, ℓ) path cover γ. Then,*

$$\lambda_{\min}(Z) \geq L_1 \frac{\min_k m_{\Gamma k}}{\max_k m_{\Gamma k}} \left(2 - 2\cos\left(\frac{\pi}{2(\ell-1)}\right)\right)$$

Proof. Since Z is a submatrix of $\tilde{Z}$, the "Rayleigh quotient" is bounded by

$$\frac{u^T Z u}{u^T u} = \frac{\tilde{u}^T \tilde{Z} \tilde{u}}{\tilde{u}^T \tilde{u}} \geq \frac{\tilde{u}^T D^{-1/2} X \overbrace{D^{-1/2}\tilde{u}}^{v}}{\tilde{u}^T \tilde{u}} = \frac{v^T X v}{v^T D v},$$

where $\tilde{u}$ has zero entries for the non-floating subdomains, and coincides with u for the

floating subdomains. As a result, we arrive at an eigenvalue problem for the matrix

$$\hat{Y}(\ell) = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{(\ell-1)\times(\ell-1)}$$

This can be regarded as a finite-difference approximation of the second derivative in one dimension, with Dirichlet condition on the left and Neumann condition on the right. This is a slightly nonstandard elementary problem, but it diagonalizes with the following "sine transform" matrix:

$$S = \left[\sin\left(\tfrac{(2j-1)\pi i}{2(\ell-1)}\right)\right]_{i,j=1}^{\ell-1}$$

The eigenvector with the smallest eigenvalue is $\left[\sin\left(\tfrac{(2j-1)\pi i}{2(\ell-1)}\right)\right]_{i=1}^{\ell-1}$ and the corresponding eigenvalue is

$$\lambda = 2 - 2\,\cos\left(\frac{\pi}{2(\ell-1)}\right)$$

As a result,

$$\frac{u^T Z u}{u^T u} \geq \left(\lambda L_1 \min_k m_{\Gamma k}\right) \frac{v^T v}{v^T D v} \geq \left(\lambda L_1 \min_k m_{\Gamma k}\right) \frac{1}{(\max_k m_{\Gamma k})} \frac{v^T v}{v^T v},$$

as required. □

As we have already mentioned, our proof is a corrected version of the one appearing in [48]. Our estimate shows that the smallest eigenvalue depends on the "condition number" $\frac{\max_k m_{\Gamma k}}{\min_k m_{\Gamma k}}$, which is a property of the domain decomposition. If using an automatic mesh partitioner, this number would be large if the smallest subdomain had very few interface points compared to the largest subdomain.

Theorem 5.5.15. *Assume that the domain decomposition admits an (L_1, ℓ) path cover γ. Then, $\|EK\| < 1$ and, as $\ell \to \infty$,*

$$\|EK\| \approx 1 - L_1 \frac{\min_k m_{\Gamma k}}{\max_k m_{\Gamma k}} \frac{\pi^2}{8} \ell^{-2} + O(\ell^{-3}).$$

Proof. By (5.110), we arrive at

$$\|EK\| = \sqrt{1 - \lambda_{\min}(Z)} = \sqrt{1 - L_1 \frac{\min_k m_{\Gamma k}}{\max_k m_{\Gamma k}} \left(2 - 2\,\cos\left(\frac{\pi}{2(\ell-1)}\right)\right)} < 1,$$

and hence $\|EK\| < 1$, as required. □

We have proven that, for the one-level 2LM methods, the quantity $\|EK\|$ approaches 1 as $\ell = O(1/H)$ tends to infinity. As per our definition of weak scaling, and as per Remark 5.5.9, we find that the one-level 2LM methods **do not scale weakly.**

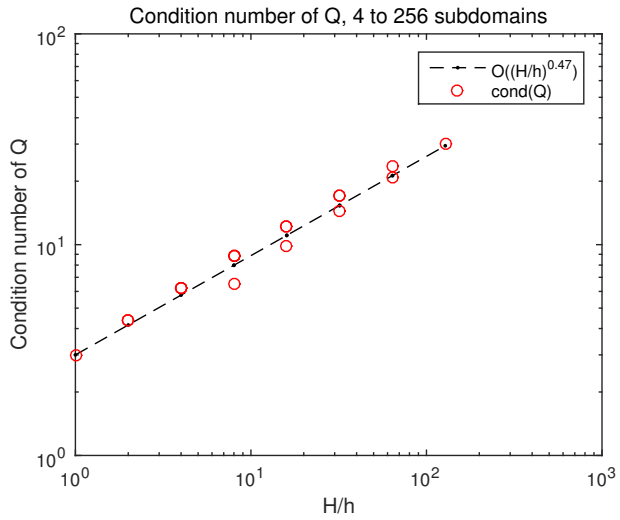


Figure 5.5: Loglog plot of condition numbers of Q, $a_{opt} = \sqrt{h/H}$

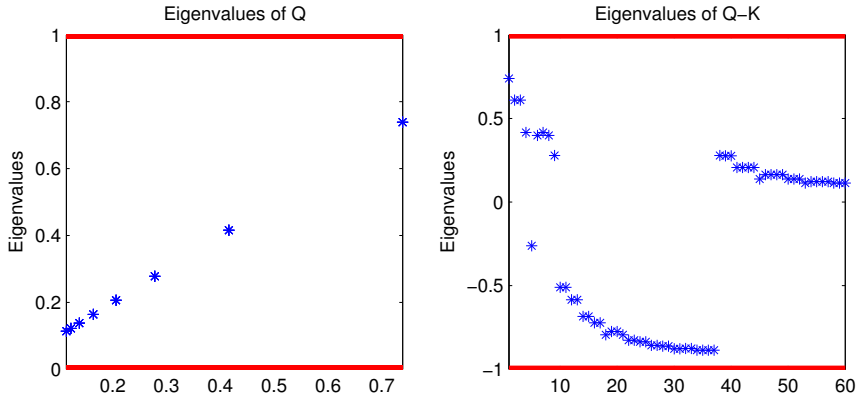


Figure 5.6: Loglog plot of eigenvalues of Q, $Q - K$

In Figure 5.5, we split the domain from 4 to 256 non-overlapping subdomains and we plot the condition number of Q with respect to the ratio H/h. By using the least squares fitting we obtain that the condition number of Q scales like $(H/h)^s$ where $s = 0.47$ very close to $1/2$, as we expected when a is close to the optimal Robin parameter a_{opt} which is defined in (5.5.3).

Definition 5.5.16 (2-Level 2-Lagrange multiplier methods)**.** *Let Q, K be as in Definition 5.5.7. Let E be the orthogonal projection onto the kernel of $I - Q$. Assume that $\|EK\| < 1$. We define the* **coarse grid preconditioner** *as*

$$P = I - EKE, \tag{5.116}$$

	h				
cond(Q)	0.062500	0.031250	0.015625	0.007813	0.003906
$H = 0.5000$	6.4940	9.788505	14.422220	20.993163	30.316758
$H = 0.2500$	6.2103	8.806499	12.238890	16.972340	23.617418
$H = 0.1250$			8.806499	12.238890	16.972340
$H = 0.0625$				8.806499	12.238890

	h				
cond($Q-K$)	0.062500	0.031250	0.015625	0.007813	0.003906
$H = 0.5000$	7.789970	11.235631	15.971733	22.617308	31.996406
$H = 0.2500$	7.608107	10.272807	14.133839	19.676555	27.585407
$H = 0.1250$	27.496822	38.137578	53.313436	74.900548	75.314923
$H = 0.0625$			105.524846	148.349634	209.068938

leading to the preconditioned matrices

$$A_{\text{2LS2LM}} = P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}} \quad \textit{and} \quad A_{\text{2L2LM}} = P^{-\frac{1}{2}}(I-2K)(Q-K)P^{-\frac{1}{2}}. \tag{5.117}$$

The terminology "2-level" comes from the fact that the action of the preconditioner P^{-1} on a residual can be efficiently computed by projecting onto a coarse grid defined by the subdomains.

Remark 5.5.17. *The matrix P is the "action of $Q-K$ on the range of E". Indeed, if we choose an orthonormal basis such that*

$$E = \begin{bmatrix} O & O \\ O & I \end{bmatrix},\ Q = \begin{bmatrix} Q_0 & O \\ O & I \end{bmatrix} \ \textit{and}\ \ K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}, \tag{5.118}$$

and where $\sigma(Q_0) \subset (\epsilon, 1-\epsilon)$, then we observe that

$$P = \begin{bmatrix} I & O \\ O & I - K_{22} \end{bmatrix} \quad \textit{and} \quad Q - K = \begin{bmatrix} Q_0 - K_{11} & -K_{12} \\ -K_{21} & I - K_{22} \end{bmatrix}. \tag{5.119}$$

In other words, the preconditioner P was obtained by "zeroing out" the off-diagonal blocks of $Q-K$ and replacing the top-left block by I.

Remark 5.5.18. *The "coarse space" is the range of E. For the model problem (5.2), the coarse space consists of piecewise constant functions, with one degree of freedom per floating subdomain (subdomains are said to be floating when they do not touch the natural boundary). Since nonzero piecewise constant functions are never continuous and K is an averaging operator, the condition $\|EK\| < 1$ is automatically satisfied.*

Remark 5.5.19. *From (5.119) we see that P is symmetric. Furthermore, P is positive definite provided that $\lambda_{\max}(K_{22}) < 1$. Note that K_{22} is the lower-right block of EKE and so $\|K_{22}\| \leq \|EK\|\|E\| = \|EK\|$. For the elliptic case $\|EK\| < 1$ is*

guaranteed see Remark 5.5.18. *Hence the matrix square roots* $P^{-1/2}$ *are well defined. Instead of computing inverse square roots of* P*, one would instead implement "left preconditioning":*

$$P^{-1}(Q-K) \quad \textit{and} \quad P^{-1}(I-2K)(Q-K).$$

These matrices can then be used inside a suitable implementation of GMRES or similar Krylov space method as a precoditioner.

Remark 5.5.20. *The matrix* A_{2LS2LM} *is symmetric and indefinite, while the matrix* A_{2L2LM} *is nonsymmetric. We will see in Subsection 5.5.4 that* A_{2LS2LM} *and* A_{2L2LM} *have equivalent condition numbers. Despite this spectral equivalence, we will see in Subsection 5.7.3 that GMRES tends to perform better with* A_{2L2LM} *than on* A_{2LS2LM} *– this may be explained by the spectral properties of* A_{2L2LM}*. Therefore, in practice it may be preferable to use the nonsymmetric matrix* A_{2L2LM} *instead of the indefinite matrix* A_{2LS2LM}*.*

5.5.3 The condition number of A_{2LS2LM}

We start from the following lemma,

Lemma 5.5.21 (A special case of [38, Corollary 6.3.4])**.** *Let* X *and* Y *be symmetric matrices of the same size. Let* $0 < \alpha < \beta < \gamma$ *be real numbers. Assume that the spectrum* $\sigma(X)$ *of* X *is contained in the interval* $[-\alpha, \alpha]$*, while* $|\sigma(Y)| \subset [\beta, \gamma]$*. Then,*

$$|\sigma(X+Y)| \subset [\beta-\alpha, \gamma+\alpha]. \tag{5.120}$$

In order to estimate the condition number of A_{2LS2LM}, we first consider the simplest case of $\kappa(Q-K)$ when 1 is not in the spectrum of Q, hence (cf. Definition 5.5.7) we have the spectral estimate $\sigma(Q) \subset [\epsilon, 1-\epsilon]$.

The operator K is an orthogonal projection, hence $\sigma(K) = \{0, 1\}$. In order to use Lemma 5.5.21 we will set $X := Q - \frac{1}{2}I$ and and $Y := -K + \frac{1}{2}I$. Then we will apply (5.120). We have that $\sigma(X) = [\epsilon - \frac{1}{2}, \frac{1}{2} - \epsilon]$, $\sigma(Y) = \{-\frac{1}{2}, \frac{1}{2}\}$ and $|\sigma(Y)| = \{1/2\}$. Hence $|\sigma(Q-K)| \subset [\epsilon, 1-\epsilon]$, $0 < \epsilon \leq \frac{1}{2}$.

When $1 \in \sigma(Q)$ unfortunately the result is not so trivial. When we apply the same shift, of $\frac{1}{2}I$, we get that $1/2 \in \sigma(X)$ and $-1/2 \in \sigma(Y)$, which means that it is possible to have a cancellation.

We have defined E as the orthogonal projection onto the kernel of $I - Q$, hence if we shift more than $\frac{1}{2}I$ in the E component in order to obtain $\sigma(X) \subset (-1/2, 1/2)$ we avoid any cancellations.

In order to estimate the spectrum of Y, one can use the following canonical form for pairs of orthogonal projections.

Lemma 5.5.22 (Halmos [35], see Appendix B)**.** *Let E and K be orthogonal projections. There is an orthogonal matrix U which simultaneously block diagonalizes E and K into 1×1 and 2×2 blocks. If we denote the kth block of the block-diagonalized E by E_k, and the kth block of the block-diagonalized K by K_k, we further have that*

$$E_k \in \left\{0, 1, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right\} \quad \text{and} \quad K_k \in \left\{0, 1, \begin{bmatrix} c_k^2 & c_k s_k \\ c_k s_k & s_k^2 \end{bmatrix}\right\}, \tag{5.121}$$

where $c_k = \cos(t_k) \neq 0$ and $s_k = \sin(t_k) \neq 0$ with real $t_k \in (0, \pi/2)$ for each k.

The ranges of E and K are hyperspaces, and the angles $\{t_k\}$ are the "principal angles" between these two hyperspaces.

Theorem 5.5.23 (Condition number and weak scaling of $P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}}$)**.** *Let $0 < \epsilon < \frac{1}{2}$ and assume that Q and K are as in Definition 5.5.7 and that E and P are as in Definition 5.5.16. Then we have the following spectral estimate:*

$$|\sigma(P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}})| \subset \frac{1}{2}\left[\sqrt{4+\epsilon^2} - 2 + \epsilon, \sqrt{4+\epsilon^2} + 2 - \epsilon\right]. \tag{5.122}$$

In particular,

$$\kappa(P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}}) \leq \frac{\sqrt{4+\epsilon^2}+2-\epsilon}{\sqrt{4+\epsilon^2}-2+\epsilon} \leq \frac{4}{\epsilon}, \tag{5.123}$$

recall $\frac{1}{\epsilon} = \sqrt{\frac{H}{h}}$.

Proof. We have defined $P = I - EKE$. In order to estimate the condition number of

$$P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}}, \tag{5.124}$$

we add and subtract $P^{-\frac{1}{2}}EP^{-\frac{1}{2}}$ to (5.124) we get

$$P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}} = P^{-\frac{1}{2}}(Q-E)P^{-\frac{1}{2}} + P^{-\frac{1}{2}}(E-K)P^{-\frac{1}{2}}. \tag{5.125}$$

We set $F = P^{-\frac{1}{2}}(E-K)P^{-\frac{1}{2}}$ and $Z = P^{-\frac{1}{2}}(Q-E)P^{-\frac{1}{2}}$. From (5.119) we know that after reordering

$$P = \begin{bmatrix} I & O \\ O & I - K_{22} \end{bmatrix} \tag{5.126}$$

and from (5.118), that

$$Q - E = \begin{bmatrix} Q_0 & O \\ O & O \end{bmatrix}. \tag{5.127}$$

Therefore if we combine (5.126) and (5.127), we conclude that $Z = Q - E$ and (5.125)

can be rewritten as

$$P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}} = (Q-E)+F.$$

From Lemma (5.5.22) we have that E, K, F can all be block-diagonalize simultaneously by an orthogonal matrix U. Matrix F can be simultaneously block diagonalized with E and K since it is a product of the block diagonal matrix $P^{-1/2}$ by $(E-K)$ and E, K are both diagonalizable by the orthogonal projection U. Furthermore we can present the blocks F_k of F as functions of blocks E_k and K_k of operators E and K respectively. More precisely, we get the following cases for each block F_k,

$$F_k = \begin{cases} 0 & \text{if } E_k = K_k = 0, \\ -1 & \text{if } E_k = 0 \text{ and } K_k = 1, \\ 1 & \text{if } E_k = 1 \text{ and } K_k = 0, \\ \begin{bmatrix} 1 & -c_k \\ -c_k & -s_k^2 \end{bmatrix} & \text{in the } 2\times 2 \text{ case.} \end{cases} \tag{5.128}$$

Since $||EK|| < 1$, the case where $E_k = 1$ and $K_k = 1$ is excluded. Let us assume that x, y are real positive parameters. In order to use Lemma (5.5.21), we rewrite

$$P^{-\frac{1}{2}}(Q-K)P^{-\frac{1}{2}} = (Q-E)+F = X+Y \tag{5.129}$$

where

$$X = (Q-E) - x(I-E) + yE \tag{5.130}$$

and

$$Y = F + x(I-E) - yE. \tag{5.131}$$

Moreover, we choose

$$x = x(\epsilon) = \frac{\sqrt{4+\epsilon^2}-\epsilon}{4} \quad \text{and} \quad y = y(\epsilon) = \frac{4-3\epsilon-\sqrt{4+\epsilon^2}}{4}. \tag{5.132}$$

We define $\text{hull}(H) := [\inf H, \sup H]$ for any subset $H \subset \mathbb{R}$. Since X can be written in a block diagonal form and Q_0 is the upper left block of matrix Q, we can easily estimate the spectrum of $X = \begin{bmatrix} Q_0 - xI & 0 \\ 0 & yI \end{bmatrix}$ as $\sigma(X) \subset \text{hull}\{\epsilon - x, 1-\epsilon-x, y\}$ and if (5.132), we get

$$\sigma(X) = \left[\frac{5}{4}\epsilon - \frac{1}{4}\sqrt{4+\epsilon^2}, \overbrace{1-\frac{3}{4}\epsilon-\frac{1}{4}\sqrt{4+\epsilon^2}}^{\alpha(\epsilon)}\right]. \tag{5.133}$$

Moreover, we have to estimate the eigenvalues of (5.131). This can be done by using the block diagonal decompositions of F, E and K as given by (5.128) and

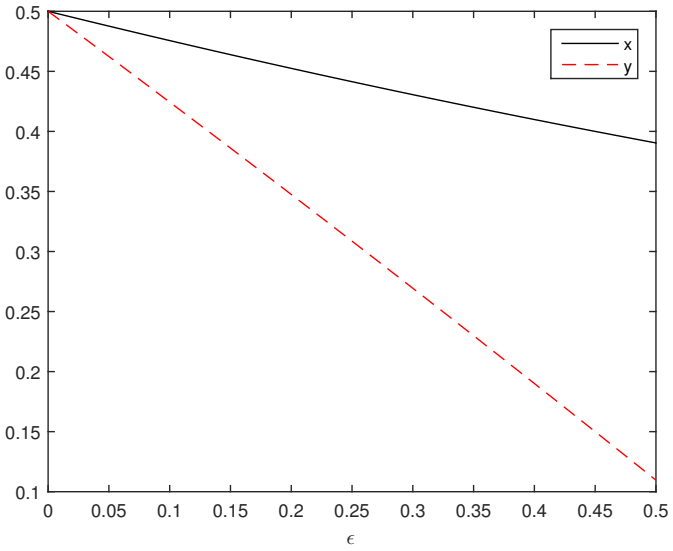


Figure 5.7: Functions $x(\epsilon)$, $y(\epsilon)$ of (5.132), $\epsilon \in (0, 0.5]$.

(5.5.22). Hence combining the sub-blocks F_k, K_k, E_k, we get that the spectrum of Y_k is

$$\sigma(Y_k) = \begin{cases} \{x\} & \text{if } E_k = K_k = 0, \\ \{x-1\} & \text{if } E_k = 0, K_k = 1, \\ \{1-y\} & \text{if } E_k = 1, K_k = 0, \\ \{\phi_{\pm}(c_k^2)\} & (2 \times 2 \text{ case}), \end{cases} \tag{5.134}$$

where

$$\phi_{\pm}(z) := \frac{1}{2}(z + x - y \pm \sqrt{z^2 + 2(x+y)z + (x+y-2)^2}). \tag{5.135}$$

Since $y < x$, (see Fig. 5.7), function ϕ_+ is positive, (see Fig. 5.8) and we can prove that it is also monotonically increasing. We have that $\frac{\partial \phi_+}{\partial z} = \frac{1}{2}(1 + (z^2 + 2(x+y)z + (x+y-2)^2)^{-\frac{1}{2}}(z+x+y))$. If we use (5.132) in order to substitute x and y we get

$$\frac{\partial \phi_+}{\partial z} = \frac{1}{2}\frac{\sqrt{(1+z-\epsilon)^2 + 4\epsilon} + (1+z-\epsilon)}{\sqrt{(1+z-\epsilon)^2 + 4\epsilon}} > 0$$

and hence $\phi_+(z)$ is monotonically increasing with respect to z. As a result

$$\phi_+(z) \in [\phi_+(0), \phi_+(1)] = \frac{1}{4}[3\epsilon + \sqrt{4+\epsilon^2}, \epsilon + 3\sqrt{4+\epsilon^2}] \subset (0, \infty). \tag{5.136}$$

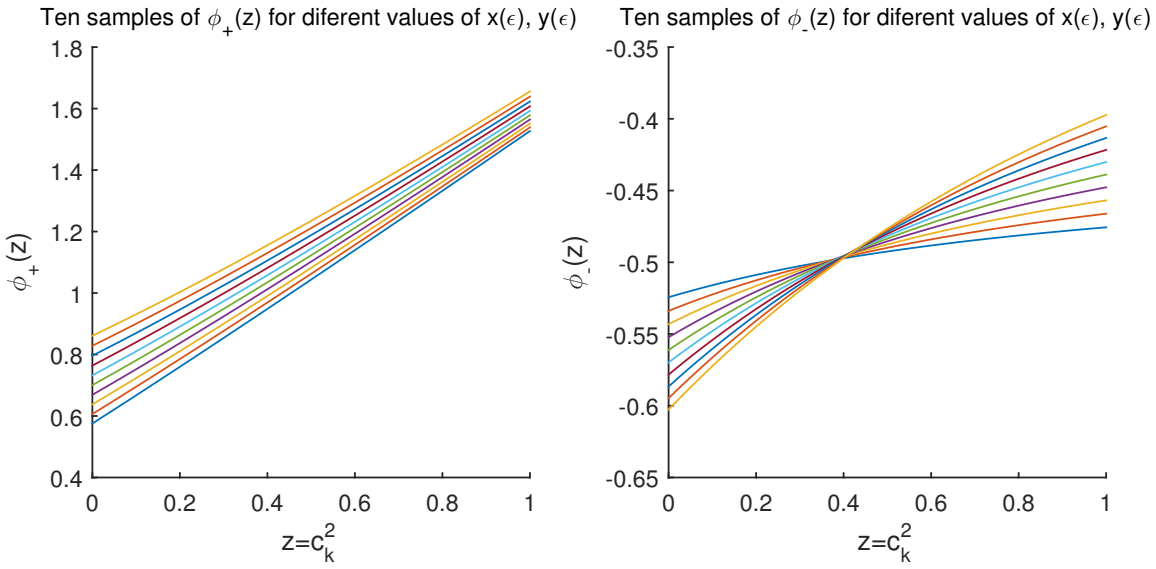


Figure 5.8: Functions ϕ_+, ϕ_-, defined in (5.135)

In the same manner ϕ_-,

$$\frac{\partial \phi_-}{\partial z} = \frac{1}{2}(1 - (z^2 + 2(x+y)z + (x+y-2)^2)^{-\frac{1}{2}}(z+x+y)).$$

using (5.132)

$$\frac{\partial \phi_-}{\partial z} = \frac{1}{2}\frac{\overbrace{\sqrt{(1+z-\epsilon)^2+4\epsilon}}^{>1+z-\epsilon>0} - (1+z-\epsilon)}{\sqrt{(1+z-\epsilon)^2+4\epsilon}} > 0$$

and hence $\phi_-(z)$ is monotonically increasing in z. Therefore,

$$\phi_-(z) \in [\phi_-(0), \phi_-(1)] = \frac{1}{4}\left[-\epsilon - 4 + \sqrt{4+\epsilon^2}, \epsilon - \sqrt{4+\epsilon^2}\right] \subset (-\infty, 0). \tag{5.137}$$

If we combine, (5.132), (5.134), (5.136) and (5.137), we obtain

$$\sigma(Y_k) \subset \left\{\overbrace{\frac{1}{4}\sqrt{4+\epsilon^2} - \frac{\epsilon}{4}}^{\beta(\epsilon)}\right\} \cup \left[\frac{3}{4}\epsilon + \frac{1}{4}\sqrt{4+\epsilon^2}, \overbrace{\frac{1}{4}\epsilon + \frac{3}{4}\sqrt{4+\epsilon^2}}^{\gamma(\epsilon)}\right] \tag{5.138}$$

$$\cup \frac{1}{4}\left[-\epsilon - 4 + \sqrt{4+\epsilon^2}, \epsilon - \sqrt{4+\epsilon^2}\right].$$

Finally, from (5.138) we get $|\sigma(Y)| \subset [\beta(\epsilon), \gamma(\epsilon)]$ and from (5.133), $\sigma(X) \subset [-\alpha(\epsilon), \alpha(\epsilon)]$, and the result (5.122) follows from Lemma (5.5.21). □

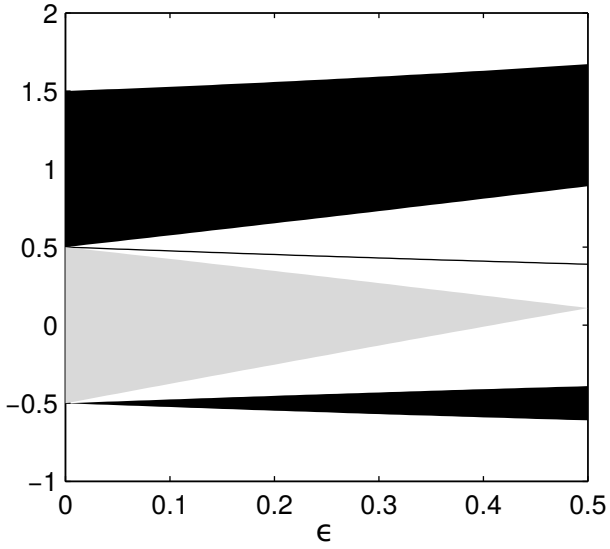


Figure 5.9: Right-hand-sides of (5.133) (lightly shaded area) and (5.138) (dark areas, including the dark curve $\beta(\epsilon)$).

Example 3. *Let $Q(q_1)$, $K(\theta)$ and $P(\theta)$ be defined by*

$$Q(q_1) = \begin{bmatrix} q_1 & 0 \\ 0 & 1 \end{bmatrix}, \quad K(\theta) = \begin{bmatrix} c^2 & cs \\ cs & s^2 \end{bmatrix}, \quad P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & c^2 \end{bmatrix}, \tag{5.139}$$

where $c = \cos\theta$ and $s = \sin\theta$ for some real parameter θ. Note that $\|EK\| = |s|$. Setting $z := c^2$, we find that $\sigma(P^{-\frac{1}{2}}(\theta)(Q(q_1) - K(\theta))P^{-\frac{1}{2}}(\theta)) = \{\psi_\pm(q_1, z)\}$ where

$$\psi_\pm(q_1, z) = \frac{1}{2}\left(1 + q_1 - z \pm \sqrt{z^2 - 2(1+q_1)z + q_1^2 - 2q_1 + 5}\right). \tag{5.140}$$

For $q_1 = 1 - \epsilon \in (0, 1)$, letting $z \to 0$ shows that (5.122) is sharp.

The condition number estimate (5.123) depends only on ϵ and hence, according to Definition 5.5.8, the 2-Level symmetric 2-Lagrange multiplier method is weakly scalable.

Remark 5.5.24. *The shifts x, y given by (5.132) were found using the following procedure. According to the discussion at the beginning of the present subsection, it is reasonable to want that $X \approx Q - \frac{1}{2}I$ and $Y \approx -K + \frac{1}{2}I$ when ϵ is small. By inspection of (5.5.3), we see that this means that $x, y \approx \frac{1}{2}$. We hypothesized that good choices of x, y would occur when some of the eigenvalue estimates of X and Y would coincide or equioscillate. We picked a small ϵ and some values of x, y slightly smaller than $\frac{1}{2}$, which seemed to indicate that the eigenvalues $1 - \epsilon - x$ and y of X should coincide, and the eigenvalues x and $\phi_-(1)$ of Y should cancel: $x + \phi_-(1) = 0$ in detail:*

$$2x + 2\phi_-(1) = 0 \tag{5.141}$$

$$2x + 1 + x - y - \sqrt{[1 + 2(x + y) + (x + y - 2)^2} = 0 \tag{5.142}$$

Now we use that $1 - \epsilon - x = y$,

$$2x + 1 + x - 1 + \epsilon + x - \sqrt{1 + 2(1 - \epsilon) + (-1 - \epsilon)^2} = 0 \tag{5.143}$$

$$4x + \epsilon - \sqrt{4x + \epsilon^2} = 0 \tag{5.144}$$

and finally,

$$x = \frac{\sqrt{4x + \epsilon^2}}{4}.$$

The values for y *can be now easily obtained by the equality* $1 - \epsilon - x = y$, *and hence we derive (5.132). Having found these values of* x, y, *we then verified that they indeed produce the estimates (5.122) and (5.123).*

5.5.4 The condition number of A_{2L2LM}

Note that

$$A_{\text{2L2LM}} = \overbrace{P^{-\frac{1}{2}}(I - 2K)P^{\frac{1}{2}}}^{Z} A_{\text{2LS2LM}} \tag{5.145}$$

Lemma 5.5.25. *Assume that* $\|EK\| < 1$ *and let* $Z = P^{-\frac{1}{2}}(I - 2K)P^{\frac{1}{2}}$. *Then,*

$$\kappa(Z) \leq \frac{\sqrt{2} + 1}{\sqrt{2} - 1} < 5.83. \tag{5.146}$$

Proof. We block diagonalize Z using Lemma 5.5.22 and (5.116). The blocks Z_k of Z are as follows:

$$Z_k = \begin{cases} 1 & \text{if } E_k = K_k = 0, \\ -1 & \text{if } E_k = 0 \text{ and } K_k = 1, \\ 1 & \text{if } E_k = 1 \text{ and } K_k = 0, \\ \begin{bmatrix} 1 - 2c_k^2 & -2c_k \\ -2s_k^2 c_k & 1 - 2s_k^2 \end{bmatrix} & \text{in the } 2 \times 2 \text{ case;} \end{cases} \tag{5.147}$$

where the case $E_k = K_k = 1$ is excluded by the hypothesis that $\|EK\| < 1$. Replacing

$s_k^2 = 1 - c_k^2$, we compute the singular values $\Sigma(Z_k)$ of Z_k to obtain

$$\Sigma(Z_k) \subset \left\{ 1, \sqrt{1 + 2c_k^6 \pm 2c_k^3\sqrt{1 + c_k^6}} \right\}. \tag{5.148}$$

Optimizing $c_k \in [0, 1]$ in (5.148) for the largest possible condition number (which occurs at $c_k = 1$) gives (5.146). □

We now prove our second main result.

Theorem 5.5.26 (Condition number of $A_{\text{2L2LM}} = P^{-\frac{1}{2}}(I - 2K)(Q - K)P^{-\frac{1}{2}}$)**.** *Let $0 < \epsilon < \frac{1}{2}$ and assume that Q and K are as in Definition 5.5.7 and that E and P are as per Definition 5.5.16. Then, we have the following condition number estimate:*

$$\kappa(A_{\text{2L2LM}}) < \frac{23.32}{\epsilon}. \tag{5.149}$$

Proof. We use the submultiplicativity of condition numbers on (5.145) combined with the estimates (5.123) and (5.146). □

According to Definition 5.5.8, A_{2L2LM} scales weakly.

5.6 Motivation for the non symmetric System

In Section 5.4, we discussed the relationship between A_{2L2LM} and the Optimized Schwarz method which is one motivation for the study of A_{2L2LM}. In the present section, we discuss another reason to prefer A_{2L2LM} over A_{2LS2LM} related to the spectral properties of these matrices.

The matrix A_{2LS2LM} is symmetric but typically indefinite (when K is a nontrivial projection) despite the fact that the initial problem (5.2) was symmetric and positive definite. Because A_{2LS2LM} is indefinite, it cannot be used with CG [37] but a method such as GMRES or MINRES (which also has a two-term recurrence) can be used. The performance of CG depends on the square root of the condition number, whereas the performance of MINRES depends on the condition number (without a square root). For instance, in the elliptic case, the matrix A_{2LS2LM} has the condition number $O(\sqrt{H/h})$ (using (5.123), provided (5.105) holds), using GMRES or MINRES on A_{2LS2LM} may have a performance comparable to using CG on, e.g., additive Schwarz with minimal overlap, which has a condition number of $O(H/h)$ [69, Theorem 3.13].

Note that in exact arithmetic, GMRES and MINRES applied to a symmetric indefinite matrix such as A_{2LS2LM} produce the same iterates. In machine arithmetic, MINRES suffers from a "loss of orthogonality" [61, p. 195] which means that its performance is usually worse than that of GMRES. We have briefly discussed MINRES

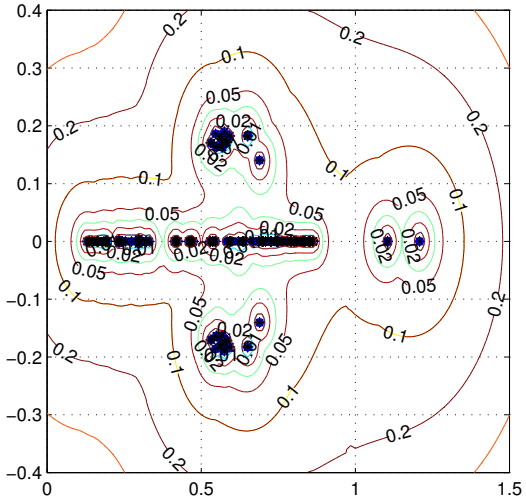


Figure 5.10: Spectrum and pseudospectrum of A_{2L2LM}.

because it is the go-to solver for symmetric indefinite systems, but in order to avoid these numerical complications, we focus on GMRES for the remainder of the thesis.

The matrix A_{2L2LM} is nonsymmetric[1]. There is no concrete result linking the condition number $\kappa(A_{\text{2L2LM}})$ to the performance of GMRES, since it is no-symmetric, but our numerical experiments suggest that GMRES applied to A_{2L2LM} is much more efficient than GMRES applied to A_{2LS2LM}. Note that the transformation from A_{2LS2LM} to A_{2L2LM} by left-multiplying by the reflection matrix $I - 2K$ is very similar to the positive definite reformulation of saddle point problems, which is experimentally known to be better for iterative solvers, even though there is no analysis, see Remark 4.3.5. Our experiments suggest the performance of GMRES may depend on $\sqrt{\kappa(A_{\text{2L2LM}})} = O((H/h)^{1/4})$, a significant improvement over $O(\sqrt{H/h})$. In Fig. 5.10, we have plotted the spectrum and pseudospectrum of A_{2L2LM}, computed numerically from an example for the problem (5.2) on the unit square with $h = 1/32$ and $H = 1/4$. In principle, the performance of GMRES can be analyzed by bounding the set of eigenvalues and using potential theory as in Section 4.3.1. In this context, our eigenvalue set consists of complex eigenvalues of mild moduli (those will not significantly impact convergence), and real or nearly-real eigenvalues that approach zero. Hence, the convergence behavior suggested by Fig. 5.10 is expected to be comparable to the case where the eigenvalues are in some interval $[\delta, 1.5]$, plus a few iterations to take care of the mild complex eigenvalues. We confirm this good behavior with numerical experiments in Section 5.7.3.

[1]Unless Q and K commute. This is unlikely to happen for a "random" matrix Q but can exceptionally happen, e.g., if there are two subdomains and $Q_1 = Q_2$.

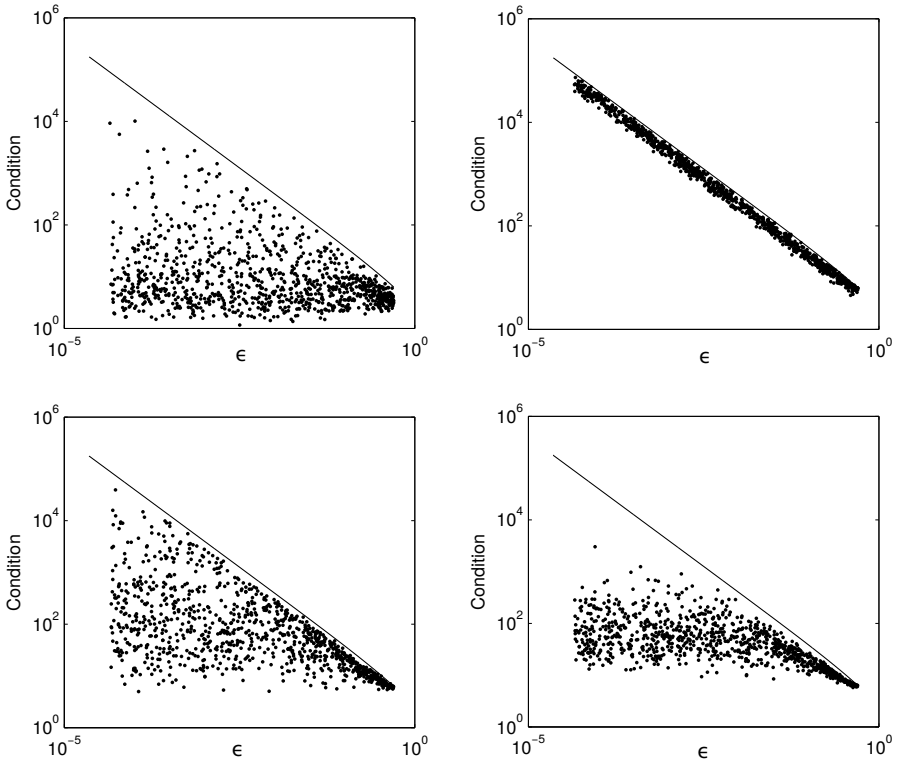


Figure 5.11: Condition numbers of A_{2LS2LM} for random choices of Q and K for various values (m, n, k) (dots) compared to (5.123) (solid curve). Top-left: $n = 4, k = 3, m = 2$; top-right: $n = 8, k = 4, m = 4$; bottom-left: $n = 15, k = 8, m = 7$; bottom-right: $n = 30, k = 18, m = 15$.

5.7 Numerical experiments

In this section we present three sets of experiments. In the first set (the "algebraic case"), we generate random matrices Q and K to validate (5.123). In the second set of experiments (the "elliptic case"), we use the model problem (5.2) with various values of h and H to validate the estimates (5.105), (5.123), (5.149). In the third set of experiments, we measure the performance of GMRES and GMRES(10) on A_{2LS2LM} and A_{2L2LM}.

In the next chapter we will present one more set of large scale experiments performed on HECToR (High End Computing Terascale Resources) supercomputer. The purpose of these experiments is to check the scalability of the two-level 2-Lagrange multiplier methods. Our massively parallel implementation described in Section 6.2.

5.7.1 Algebraic case

Our first series of numerical experiments (see Fig. 5.11) consists of generating random matrices Q and K and verifying the estimate (5.123). For each plot we generate 1000 random matrices Q and K. For each matrix Q we fix the ambient dimension n, the

Table 5.1: Condition numbers for the 2-Level **symmetric** 2-Lagrange multiplier matrix A_{2LS2LM} for the model problem (5.2).

	h				
	0.1250	0.0625	0.0313	0.0156	0.0078
$H = 0.2500$	7.1927	9.6909	13.2186	18.1121	25.0399
$H = 0.1250$		9.3567	12.2031	16.4196	22.3863
$H = 0.0625$			9.8847	12.8234	17.2216

Table 5.2: Condition numbers for the 2-Level **nonsymmetric** 2-Lagrange multiplier matrix A_{2L2LM} for the model problem (5.2).

	h				
	0.1250	0.0625	0.0313	0.0156	0.0078
$H = 0.2500$	6.1508	8.4162	11.6954	16.3073	22.9120
$H = 0.1250$		7.3623	9.9804	13.8537	19.3739
$H = 0.0625$			7.6480	10.3564	14.3824

parameter $\epsilon > 0$ and the number k of eigenvalues of Q that are less than 1. We then set the smallest eigenvalue to ϵ, the largest eigenvalue smaller than 1 to $1 - \epsilon$, and the remaining $k - 2$ eigenvalues are picked randomly and uniformly in the interval $[\epsilon, 1 - \epsilon]$. The matrix Q is then taken to be the corresponding block diagonal matrix.

We also generate K randomly as follows. First, we fix the dimension m of the range of K. Then we generate a matrix V of dimension $n \times m$ whose columns are orthonormal, and we set $K = VV^T$. The matrix V is generated randomly with the MATLAB command `orth(rand(n,m)-0.5)`.

Each such experiment produces a condition number for $P^{-\frac{1}{2}} A_{\text{S2LM}} P^{-\frac{1}{2}}$ which is plotted as a dot against the value of ϵ in Fig. 5.11. Although the resulting probability distribution of points does depend on the parameters (k, m, n), we find that the bound (5.123) (plotted as a solid line in Fig. 5.11) holds and seems to be sharp.

5.7.2 Elliptic case

Our second set of experiments is on the model problem (5.2). We have discretized the unit square with a regular grid of the fine grid diameter h. We have partitioned this square into square subdomains of side H, with up to 256 subdomains. We then assembled the matrices A_{2LS2LM} and A_{2L2LM} and computed the condition numbers using MATLAB's `cond`.[2] The results are summarized in Tables 5.1 (symmetric case)

[2]We found that `cond(X)` gives much less accurate results when `X` is a sparse matrix. This is because `cond` then uses the approximate condition number estimate `condest(X)`. In order to obtain more accurate results, we stored A_{2LS2LM} and A_{2L2LM} as dense matrices.

and 5.2 (nonsymmetric case).

The estimate (5.105) implies that the condition numbers along the diagonals of Tables 5.1 and 5.2 should be bounded, which appears to be the case (this is "weak scaling"). Furthermore, the estimate (5.105) implies that moving one column to the right ought to increase the condition number by a factor of $\sqrt{2} \approx 1.4$, which is also approximately verified. Indeed, the relative increases for the last column of Table 5.1 compared to the penultimate column are $1.38, 1.36, 1.34$. The corresponding ratios in Table 5.2 are $1.41, 1.40, 1.39$.

We highlight the fact that this set of numerical experiments include many cross points and a large number of floating subdomains. Our new 2-Level method is able to deal with these challenging situations without difficulty and with good scaling properties.

We also note that our estimate (5.149) of the condition number of A_{2L2LM} is $5.83\times$ worse than the estimate (5.123), but this is not borne out in our numerical experiments. Indeed, the matrix A_{2L2LM} appears to be better conditioned than the matrix A_{2LS2LM}. Our estimate of the condition number of A_{2L2LM} was obtained using the "rough" idea of the submultiplicativity of condition numbers, which is apparently very conservative in the present situation.

5.7.3 Performance with GMRES and GMRES(10)

Our third set of experiments (see Fig. 5.12) consists of using the GMRES and restarted GMRES(10) iterations on the matrices of Section 5.7.2, and where the initial residual is a column vector of ones. We now briefly discuss these results, starting with GMRES. Since A_{2LS2LM} is symmetric, we can use standard theory to estimate the convergence of GMRES (which in this case is equivalent to MINRES). A worst-case bound is, from (4.50),

$$\frac{\|r_k\|_2}{\|r_0\|_2} = O\left(\left(\frac{\kappa - 1}{\kappa + 1}\right)^{k/2}\right), \tag{5.150}$$

where $k = 0, 1, \ldots$ is the iteration count, r_k is the corresponding residual and the condition number $\kappa = 12.8$ from Table 5.1 was used. This is quite a slow convergence and this estimate is known not to be sharp when the spectrum exhibits some asymmetry about the origin. Indeed, we see that when h is large, GMRES on A_{2LS2LM} performs even better than

$$\frac{\|r_k\|_2}{\|r_0\|_2} = O\left(\left(\frac{\kappa - 1}{\kappa + 1}\right)^{k}\right), \tag{5.151}$$

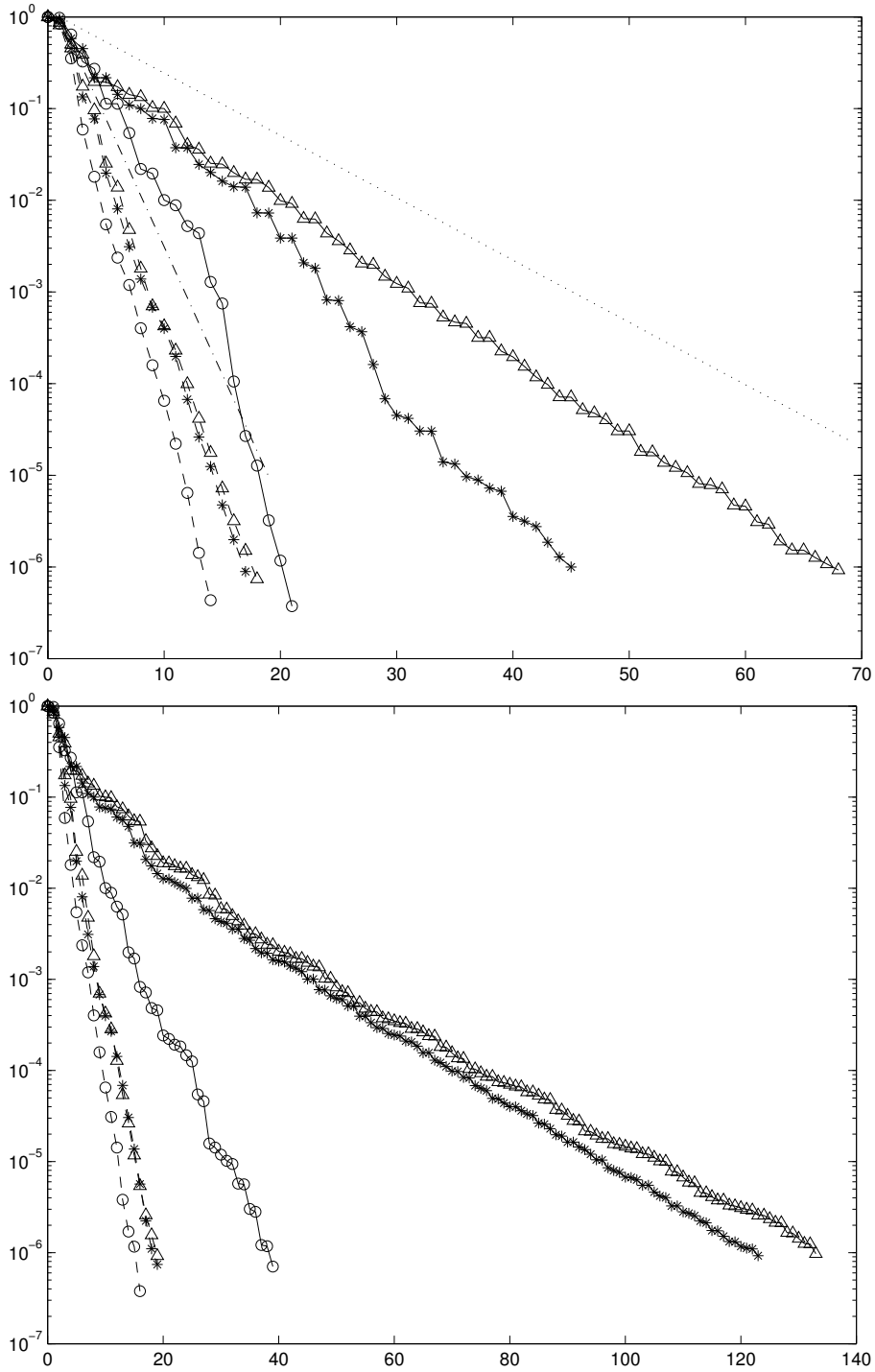


Figure 5.12: Convergence of the relative residual norm in the GMRES (top) and GMRES(10) (bottom) iterations (to a relative tolerance of 10^{-6}) with grid parameters $h = \frac{1}{4}H = \frac{1}{16}$ (circles), $h = \frac{1}{4}H = \frac{1}{32}$ (stars) and $h = \frac{1}{4}H = \frac{1}{64}$ (triangles). The solid lines correspond to A_{2LS2LM}, while the dashed lines correspond to A_{2L2LM}. In the top figure, the dotted line is (5.151) and the dot-dashed line is (5.152).

As can be observed, see Fig. 5.12, the linear estimate (5.151) is very pessimistic when h is large. As a result, the scalability of the algorithm is only apparent when h is very small.

As mentioned in Section 5.6, the matrix A_{2L2LM} has much better spectral properties. The present experiments suggest that the correct linear estimate for the convergence of GMRES applied to A_{2L2LM} is

$$O\left(\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right). \tag{5.152}$$

(The value $\kappa = 10.4$ from Table 5.2 was used.) This may be related to the fact that the spectrum of A_{2L2LM} is essentially a positive interval plus some complex eigenvalues of mild moduli, see Fig. 5.10.

We now turn to the GMRES(10) experiments (Fig. 5.12, bottom). The restarted GMRES algorithm can be used when the storage requirements of the full GMRES algorithm are too high. We have used the GMRES(10) algorithm, which restarts every tenth iteration. Although the performance of A_{2LS2LM} appears scalable, the iteration counts are now much higher. By contrast, the matrix A_{2L2LM} is scalable in all cases and the iteration counts are nearly the same as in the full GMRES algorithm (less than 20 in all cases).

Chapter 6

Massively parallel Implementations and Experiments

In this chapter we will discuss the parallel implementation of the 2-Level 2 Lagrange multiplier methods and the corresponding parallel experiments. We mainly refer to our work based on articles [41], [40] that correspond to Sections 6.1 and 6.2 respectively. We include the details of the parallel implementation and the large scale experiments that confirm the good scaling properties of the 2 level 2 Lagrange multiplier methods. The numerical experiments were run on the HECToR supercomputer, a Cray XE6 with 2816 compute nodes each comprising of two 16-core AMD Opeteron Interlagos processors. Each of the 16-core sockets is coupled with a Cray Gemini routing and communications chip.

6.1 First Implementation and Experiments

We have implemented the symmetric and nonsymmetric 2LM methods in C using the PETSc library [6]. We implemented three matrices K, Q and the coarse grid preconditioner P. The matrices P, Q are implemented as PETSc shell matrices while the K matrix is assembled into a `seqaij` matrix. In other words, the matrix K is assembled into PETSc's parallel compressed row storage sparse matrix format, while the matrices P and Q are not assembled but instead a matrix-vector multiplication routine is provided to PETSc. The matrices P and Q are not assembled because they are not sparse.

We use a PETSc parallel Krylov space solver on (5.17) or (5.18) as an "outer iteration". Each step of the outer iteration requires multiplying a given vector by the matrices P, Q, K. The matrix-vector product $K\lambda$ is a straightforward sparse matrix-dense vector product. The matrix-vector product $Q\lambda$ requires solving subdomain problems as per (5.8). These subdomain problems can in principle become large. Thus, (5.8) is solved using a PETSc sequential Krylov space solver (ie. a single-

processor solver) on (5.8). This is an "inner iteration" which occurs at each step of the outer iteration. Hence the overall algorithm has an inner-outer iteration structure. In our test implementation we use a finite difference implementation with a square domain and rectangular subdomains, with one domain assigned per MPI task with affinity to a single core.

The matrix K

The solution λ to the linear systems (5.17) or (5.18) is a multi-valued trace, with one function value per artificial interface point per subdomain. In PETSc, the rows of λ are distributed such that the indices of the same domain are assigned to a single processor,

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_p \end{bmatrix}.$$

Each entry in λ corresponds to an artificial interface grid point. When two or more subdomains are adjacent, then some entries of λ correspond to the same artificial interface point.

Each processor lists the physical grid points on its artificial interface and this information is shared with neighboring subdomains using MPI explicitly. When solving subdomain problems we work with small-dimensional local vectors. The Robin data λ_j on subdomain Ω_j has length $n_{\Gamma j}$; we write $\lambda_j = (\lambda_i^{(j)})_{i=1}^{n_{\Gamma j}}$. Mapping from the "local index" i to a "global offset" is achieved with the function $F_j(i) = i + \sum_{k<j} n_{\Gamma k}$. The size of the matrix K is $\sum_{k=1}^{p} n_{\Gamma k}$. Given this information, each processor is able to assemble its own rows of K.

The matrix Q

We begin by showing that the matrix-vector product $\lambda_k \mapsto Q_k \lambda_k$ can be computed by solving a local sparse problem. Setting $f = 0$ (and hence $g = 0$) in (5.8) and (5.13) shows that $Q_k \lambda_k = a u_{\Gamma k}$, where $u_{\Gamma k}$ is defined by,

$$\begin{bmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix}. \tag{6.1}$$

Thus, in order to calculate the matrix-vector product $Q\lambda$, each processor solves the Robin local problem (6.1) and outputs $Q_k \lambda_k = a u_{\Gamma k}$.

The local problem (6.1) can in principle be solved using e.g. a Cholesky decomposition. However, we found that using a Cholesky decomposition leads to large amounts of fill-in and poor performance. Thus, we solve the local problem (6.1) using

the Conjugate Gradient method with relative convergence tolerance 1e-10 and absolute convergence tolerance 1e-9. For the local problem (6.1) we use the incomplete Cholesky ICC(ℓ) preconditioner [16]. The incomplete Cholesky preconditioner is a compromise between higher fill-in (leading in the limit to a direct solver) and lower fill-in (leading in the limit to a diagonal preconditioner). We found that a "factor level" $\ell = 10$ gives better overall performance for our problem sizes.

The preconditioner P

The coarse grid preconditioner matrix P defined in (5.5.16) is in principle an enormous parallel matrix. Nevertheless, we will describe an efficient way to compute the matrix-vector product $\lambda \mapsto P^{-1}\lambda$ on a single processor (with some global communication). For $j = 1, \ldots, p$ we denote $n_{\Gamma j}$ the number of vertices on the artificial interface $\partial\Omega_j \cap \Gamma$ and we define the matrix $J := \mathrm{diag}(\frac{1}{\sqrt{n_{\Gamma 1}}}\mathbf{1}_{n_{\Gamma 1}}, \ldots, \frac{1}{\sqrt{n_{\Gamma p}}}\mathbf{1}_{n_{\Gamma p}})$ where $\mathbf{1}_j$ denotes the jth dimensional column vector of ones. The columns of J span the "coarse space" of piecewise constant functions, which are constant on each local artificial interface $\Gamma_k = \partial\Omega_k \cap \Gamma$. The coarse space for the preconditioner (5.116) is the kernel of S, which is contained in the column span of J. Thus, we define $E := JJ^T$ and,

$$P^{-1} := (I - EKE)^{-1} = I - JJ^T - J(\overbrace{J^TKJ - I}^{L})^{-1}J^T.$$

Note that although P^{-1} is dense, we can compute $\lambda \mapsto P^{-1}\lambda$ efficiently, in a matrix-free way, via the formula $P^{-1}\lambda = \lambda - J(J^T\lambda) - J(L^{-1}(J^T\lambda))$.

Given the assembled parallel sparse matrix J and its transpose J^T and the assembled (sparse) local matrix L, the algorithm for computing the matrix-vector product $\lambda \mapsto P^{-1}\lambda$ in a matrix-free way is as follows:

1. Given λ, compute the p-dimensional "coarse" vector $\lambda_c = J^T\lambda$ and collect its entries on a single processor as a sequential vector.

2. Define u_c by solving the locally, sparse linear problem $Lu_c = \lambda_c$.

3. Output $P^{-1}\lambda = \lambda - J\lambda_c - Ju_c$. Note that multiplication by J involves broadcasting the small local vectors λ_c and u_c to large parallel vectors $J\lambda_c$ and Ju_c.

The outer solve

The implementations of the shell matrices P and Q and the assembly of the sparse matrix K have been described. Building on these base implementations, we further form the shell matrices $\lambda \mapsto (Q - K)\lambda$ (implemented as `QminKmul`) and $\lambda \mapsto (I - 2K)(Q - K)\lambda$ (implemented as `Imin2KQminKmul`). The PETSc library enables us to use a variety of different solvers. For the outer iteration we experimented with

the Generalized Minimal Residual `KSPGMRES` and the Flexible Generalised Minimal Residual method `KSPFGMRES` on shell matrices `QminKmul` and `Imin2KmulQminK`, with the preconditioner P. For the `KSPFGMRES` solver we set the relative convergence tolerance $1e-7$ and the absolute convergence tolerance $1e-6$.

Recall that GMRES is an iterative method that computes the approximate solution $x_k \in x_0 + \text{span}\{r_0, Ar_0, \dots A^k r_0\}$ which minimizes the residual norm $\|b - Ax_k\|_2$. The efficient implementation of the least-squares problem relies on the identity

$$AV_k = V_{k+1}\tilde{H}_k, \tag{6.2}$$

where V_k is an orthonormal basis of the Krylov space and $\tilde{H}_k$ is an upper Hessenberg matrix (see [59] for details). The Flexible GMRES algorithm [58] replaces (6.2) by

$$AZ_m = V_{k+1}\tilde{H}_k, \tag{6.3}$$

and allows one to vary the preconditioner at each iteration, which required testing since our matrix-vector products are inexact.

Experiments at large scale

Results for the iteration counts of the S2LM and 2LM methods are presented. In both cases the Flexible GMRES algorithm for the outer solver and the Conjugate Gradient algorithm for the inner solver were used. The preconditioner for the outer solve is the shell matrix P, while the preconditioner for the inner solve is the incomplete Cholesky ICC(10) of (6.1).

The implementation used here is limited to a square domain in two dimensions using a finite difference discretization. This choice was made entirely for the simplicity of implementation. The domains vary from 100^2 to 10000^2 grid points (and hence the largest problem has 10^8 degrees of freedom). These domains are partitioned into 64 to 4096 subdomains, which again is limited to a square number. This domain decomposition is mapped to the MPI decomposition on the HECToR.

The symmetric (5.17) and nonsymmetric systems (5.18) are solved, with relative convergence tolerance $1e-7$ and the absolute convergence tolerance $1e-6$. The outer iteration counts are reported in Tables 6.1 and 6.2. The computational cost per outer iteration for a fixed domain and subdomain is constant. The inner iterations are not reported as the ICC preconditioner is used for simplicity rather than the optimal multigrid which would be used as first choice in a production implementation. In addition to these raw iteration counts, we also plot the scaling of the methods against the ratio H/h in Figs. 6.1 and 6.2

The S2LM performance is well explained by the condition number estimate of

Table 6.1: Iteration counts for S2LM.

# Procs.	Domain size 100^2	300^2	1000^2	3000^2
64	216	409	952	2472
256	173	316	782	1753
1024	144	220	411	1090
4096	-	-	301	665

Table 6.2: Iteration counts for 2LM.

# Procs.	Domain size 100^2	300^2	1000^2	3000^2	10000^2
64	30	58	114	229	-
256	37	35	72	135	-
1024	47	44	42	76	-
4096	-	-	53	50	82

Theorem 5.5.23. Indeed, the S2LM matrix is symmetric and indefinite and for such systems, one can show that the number of iterations is bounded by a quantity proportional to the condition number. This bound is only sharp when the spectrum of the matrix is perfectly symmetric about the origin. We find that some of our smaller systems perform slightly better than this theoretical estimate.

The 2LM performance appears to be between $O(H/h)^{1/3}$ and $O(H/h)^{1/2}$. The 2LM matrix is nonsymmetric. For nonsymmetric matrices, the condition number does not necessarily predict the performance of the GMRES algorithm. However, in our case, we find that the condition number explains well the performance of the algorithm and that we further get "Krylov acceleration" – the performance may be almost as good as $O(H/h)^{1/3}$.

6.2 Second Implementation: A Massively parallel implementation "black-box" solver

In our second implementation we created a "black-box" solver by using the `MATIS`[1] type matrix of PETSc library, such that the user gives as input a matrix of type `MATIS` that contains the local Neumman problem and information about the global numbering of the nodes of type `IS` and then without any other information it is able to solve the problem and provide the local solutions for each subdomain.

[1] `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Mat/MATIS.html`

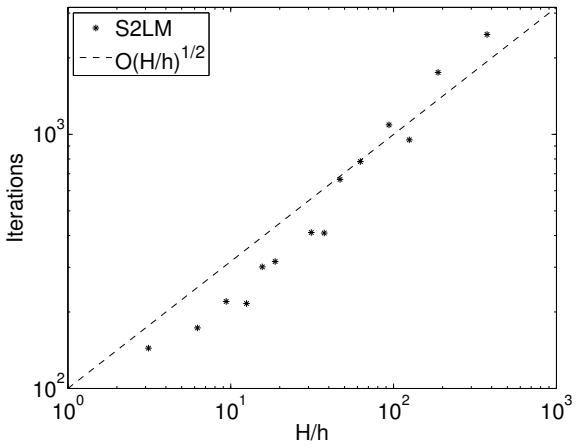


Figure 6.1: Scaling of S2LM.

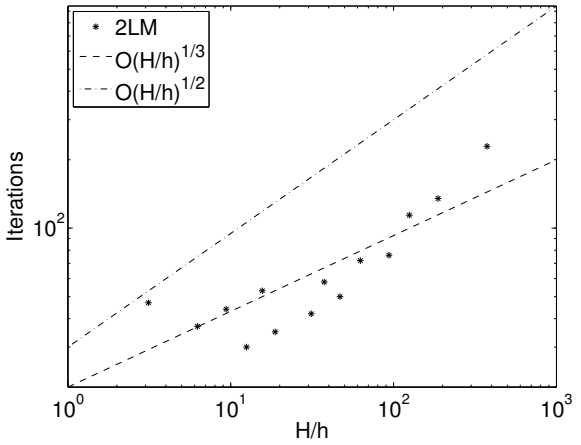


Figure 6.2: Scaling of 2LM.

We present the parallel implementation of the 2LS2LM and 2L2LM methods with cross points and with the coarse grid preconditioner (5.116) described by systems

$$P^{-1}(Q-K)\lambda = -P^{-1}Qg, \tag{6.4}$$

$$P^{-1}(I-2K)(Q-K)\lambda = -P^{-1}(I-2K)Qg. \tag{6.5}$$

We implemented these methods in C using the PETSc [6] library. Our code works for general domains Ω and subdomains Ω_i of arbitrary shapes.

Our objective was to create a 2-Lagrange multiplier "black-box solver" that takes as an input a parallel distributed matrix which holds the "splitting" (5.10). Then the solver functions algebraically on the given input information, in order to solve numerically problem (5.3), using either the 2LS2LM or the 2L2LM methods. The PETSc type `MATIS` is used as an input for our solver since it can efficiently encode and store (5.10).

Moreover we have implemented our own parallel mesh generation and partitioning algorithm, that has been used along with the Triangle 2D Mesh Generator and Delaunay Triangulator [63] and gives as an output (5.10).

6.2.1 Mesh Generation and assembly of local Neumann problems

We start from a seed mesh $\mathcal{T}_0$ that describes the general geometry of our problem. Then by further refinement of $\mathcal{T}_0$ a new coarse mesh $\mathcal{T}_H$ is created. Each triangle of $\mathcal{T}_H$ becomes a subdomain and is assigned to a unique processor. The user provides the number m, of vertices to be generated on each edge of the coarse mesh $\mathcal{T}_H$ and the mesh is refined accordingly, in order to create the desired refined mesh $\mathcal{T}_h$. The fine mesh can be in general very large so is created on a per subdomain basis.

Since $\mathcal{T}_h$ is not globally assembled, only the local numbering of the nodes is known to each processor. In order for each processor to acquire the global numbering of its nodes, without any communication cost, we designed the following algorithm.

For each subdomain Ω_i of the fine mesh we can compute the number of vertices that belong to it. Each vertex of the fine mesh ν_i is labeled with an integer $i = 1, \ldots, n$. Each processor has a corresponding subdomain Ω_i with neighbours Ω_j. For the subdomain Ω_i and its neighbours Ω_j such that $j < i$ the fine mesh is created, see Fig. 6.3. The information in Fig. 6.3 is sufficient to compute the global labels ℓ_i of the vertices $\nu_i \in \Omega_i$, without assembling the global fine mesh and without any `MPI` communication.

Next, we assign each of the fine vertices to a single owner subdomain in the following way. Vertices that lie in the interior of a subdomain Ω_i are assigned to subdomain Ω_i. Vertices along an edge $\partial\Omega_i \cap \partial\Omega_j$ but not at a cross point are assigned to the subdomain $\Omega_{min(i,j)}$. Vertices at a cross point $v^* \in \partial\Omega_i \cap \partial\Omega_j \cap \ldots \cap \partial\Omega_k$ are assigned to the subdomain $\Omega_{min(i,j,\ldots,k)}$.

Each subdomain Ω_i consists of vertices $v_i \in \overline{\Omega_i}$, some of which have been assigned to subdomain Ω_j while others to the neighbouring subdomains with smaller numbering. This way, the precise ownership of each $v_i \in \overline{\Omega_i}$ can be computed locally without the need of assembling the global fine mesh.

Once the global labels of the vertices of the fine mesh are computed, a "local to globlal mapping" which specifies the binary restriction matrices R_i is defined. Because R_i restricts to Ω_i, only the labels of the vertices in $\overline{\Omega_i}$ are required and hence no communication is needed in order to assemble R_i.

Likewise, the Neumann matrices A_{Ni} are computed and assembled as `seqaij` matrices without communication. The resulting objects $\{R_i, A_{Ni}\}$ form the PETSc distributed matrix of type `MATIS`.

6.2.2 The 2LS2LM and 2L2LM "black-box" solver

The solver takes as an input a matrix of type `MATIS` that contains the information about the local Neumann problems A_{Ni} and the restriction matrices R_i. The matrices

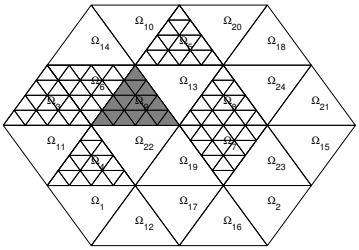


Figure 6.3: Processor that has been assigned the "gray" subdomain Ω_9, refines all the neighbouring subdomains Ω_j with $j < 9$.

K, Q and P in (6.4) and (6.5) are implemented. Matrix K is assembled as a parallel `mpiaij` matrix, in a compressed row storage matrix format. Since P, Q are dense matrices, they are not assembled explicitly but instead are implemented as PETSc "matrix-free" matrices by defining the matrix vector products $P\lambda$ and $Q\lambda$ respectively.

The matrix K is assembled as a parallel sparse matrix and is defined from the following product of matrices,

$$K = WR_\Gamma R_\Gamma^T, \text{ where } R_\Gamma = \begin{bmatrix} R_{\Gamma_1} \\ \vdots \\ R_{\Gamma_p} \end{bmatrix}, \tag{6.6}$$

$W = (\text{diag}(R_\Gamma R_\Gamma^T \mathbf{1}))^{-1}$ and $\mathbf{1}$ corresponds to a vector of ones.

The matrix Q is implemented in a "matrix-free" form. Since Q is a block diagonal matrix of submatrices Q_k, it is implemented as the matrix vector product $\lambda_k \mapsto Q_k\lambda_k$. This product can be computed by solving the local sparse Robin problem

$$\begin{bmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix} \tag{6.7}$$

for each λ_k of the mutli-valued trace vector λ.

Remark 6.2.1. *The solution λ of either* (6.4) *or* (6.5) *is a many-sided trace with one function value per artificial interface point per subdomain. The vector λ is a PETSc parallel vector object, the rows of which are distributed in a way such that the indices of the same domain are assigned to a single processor.*

The coarse grid preconditioner P defined in (5.116), can be assembled in a "black-box" manner. Let $\mathbf{1}_{n_k}$ denote the $n_k th$ dimensional column vector of ones, where n_k is the size of A_{Nk}, see (5.9). Since $\mathbf{1}_{n_k}$ spans $\ker A_{Nk}$, we can detect the floating subdomains by checking if the product $A_{Nk}\mathbf{1}_{n_k} = 0$, with some tolerance.

Then we are able to produce the basis of the coarse space as

$$J := \text{blkdiag}(\frac{1}{\sqrt{n_{\Gamma 1}}}\mathbf{1}_{n_{\Gamma 1}}, \ldots, \frac{1}{\sqrt{n_{\Gamma p}}}\mathbf{1}_{n_{\Gamma p}}),$$

where $n_{\Gamma k}$ is the number of vertices on the artificial interface $\partial\Omega_k \cap \Gamma$. This orthonormal basis for the range of E gives the formula $E = JJ^T$, allowing us to implement P^{-1} in a matrix-free way. We now use the block notation (5.119). Note that in this basis $J = \begin{bmatrix} O \\ I \end{bmatrix}$. We find that:

$$P^{-1} = \begin{bmatrix} I & O \\ O & (I - K_{22})^{-1} \end{bmatrix} = \begin{bmatrix} I & O \\ O & (I - J^T K J)^{-1} \end{bmatrix} = \overbrace{\begin{bmatrix} I & O \\ O & O \end{bmatrix}}^{I-E} + \overbrace{\begin{bmatrix} O & O \\ O & (I - J^T K J)^{-1} \end{bmatrix}}^{J(I-J^T K J)^{-1} J^T}.$$

Thus,

$$P^{-1} = I - JJ^T + J(I - J^T K J)^{-1} J^T. \tag{6.8}$$

The matrix J and J^T and the $p \times p$ coarse problem $L = I - J^T K J$ are assembled explicitly. Given λ, we compute the p-dimensional vector $\lambda_c = J^T \lambda$ and we gather its entries on a single processor as a sequential vector. Then, we solve the coarse problem $Lu_c = \lambda_c$ using LU decomposition. The sequential vector u_c is scattered to all processors and finally we get the desired output from,

$$P^{-1}\lambda = \lambda - J\lambda_c + Ju_c.$$

We also define the matrices for $(Q - K)$ and $(I - 2K)(Q - K)$ in a "matrix-free" form with the corresponding matrix-vector product operations, $\lambda \mapsto (Q - K)\lambda$ and $\lambda \mapsto (I - 2K)(Q - K)\lambda$.

In order to solve the systems (6.4), (6.5) we use the parallel Generalised Minimal Residual Krylov subspace method `KSPGMRES` provided by the PETSc library with the preconditioner P given by (6.8).

Finally, once the solution λ of either (5.17) or (5.18) is obtained, the solution of the global problem $Au = f$ is recovered locally by solving (6.7). This is due to the fact that the final step of solving (6.7) requires only the local part λ_k of the parallel vector λ.

6.2.3 Large scale experiments on HECToR supercomputer

In this subsection we present some results of the iteration counts and the walltime that correspond to the massively parallel implementation of A_{2L2LM} and A_{2LS2LM} described in Section 6.2. We solve the problem (5.3) with the constant function $f = 1$

as a right hand side, where Ω is the wrench-shaped domain in Fig.6.4. In this set of experiments in order to solve systems involving A_{2L2LM} or A_{2LS2LM} with the coarse grid preconditioner (5.116), we have used the generalized minimal residual method `KSPGMRES`, with 1e-7 relative tolerance and 1e-6 absolute tolerance, respectively.

The number of grid points in the domains varies from 10^5 to 10^8 grid points. Moreover, the domains are partitioned from 51 to 3264 subdomains. The experiments were performed on the HECToR supercomputer where one subdomain is assigned to each processor. The results in terms of iteration counts and the walltime are presented in Tables 6.3, 6.4, 6.5, 6.6. The full code for our A_{2L2LM} or A_{2LS2LM} solver implementation can be found online at `https://bitbucket.org/modios/matis_2lm`.

We see that the nonsymmetric method 2L2LM produces very moderate iteration counts (103 iterations in the very worst case) while the symmetric method 2LS2LM produces many more iterations. In principle this suggests one should use the nonsymmetric method to obtain better performance. However, the higher number of iterations is not always reflected in the wall clock time. This is partially because the HECToR supercomputer requires a significant amount of time to distribute our tasks to all the nodes in the cluster (for smaller problems, this is essentially all of our running time). However, for the largest problems we gain one order of magnitude in the wall clock time simply by using the nonsymmetric method.

The scaling properties are also better in the nonsymmetric method. For the symmetric method, going from $7 \cdot 10^6$ to $2 \cdot 10^7$ grid points increases the iteration counts by factors of $1898/947 \approx 2.004$ and $1108/617 \approx 1.796$. By comparison, the nonsymmetric method with the same number of processors only increases the iteration counts by factors of 1.209 and 1.396 respectively, so we have much better scaling properties from the 2L2LM method than the 2LS2LM method.

The communication overheads for PETSc on HECToR were significant and we can see in some cases that problems of a certain size require a longer wall clock time when processors are added. Although we made some effort to optimize this, we concluded that significant engineering efforts would be required to extract the most performance from this hardware.

6.2.4 More details on the parallel implementation

In this subsection we will present in detail the structure of our massively parallel implementation. We will emphasize on the most significant functions of our algorithm, which parts of the code are new and which ones are based on existing libraries. As we have discussed in the introduction of Section 6.2 the algorithm is implemented in the C language and the external libraries that we used are the following:

Table 6.3: Iteration counts for 2LS2LM.

	Number of grid points ≈				
# Procs.	10^5	$4 \cdot 10^5$	10^6	$7 \cdot 10^6$	$2 \cdot 10^7$
51	489	859	1369	1438	-
204	445	597	888	1363	-
816	316	510	616	947	1898
3264	-	-	-	617	1108

Table 6.4: Walltime for for 2LS2LM.

	Number of grid points ≈				
# Procs.	10^5	$4 \cdot 10^5$	10^6	$7 \cdot 10^6$	$2 \cdot 10^7$
51	27s	30s	54s	167s	-
204	27s	28s	35s	62s	-
816	43s	53s	61s	83s	172s
3264	-	-	-	610s	1055s

Table 6.5: Iteration counts for 2L2LM.

	Number of grid points ≈					
# Procs.	10^5	$4 \cdot 10^5$	10^6	$7 \cdot 10^6$	$2 \cdot 10^7$	10^8
51	43	50	58	72	-	-
204	44	51	64	77	92	-
816	41	48	55	67	81	103
3264	-	-	-	48	67	83

Table 6.6: Walltime for 2L2LM.

	Number of grid points ≈					
# Procs.	10^5	$4 \cdot 10^5$	10^6	$7 \cdot 10^6$	$2 \cdot 10^7$	10^8
51	25s	26s	30s	48s	-	-
204	26s	27s	27s	29s	51s	-
816	28s	28s	29s	31s	36s	68s
3264	-	-	-	90s	118s	135s

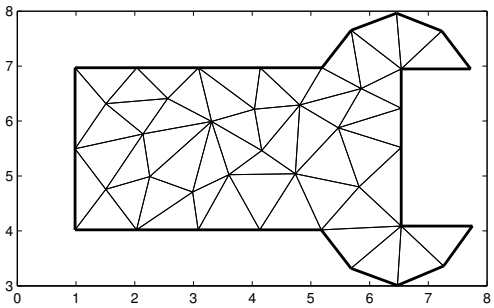


Figure 6.4: Wrench-shaped domain Ω.

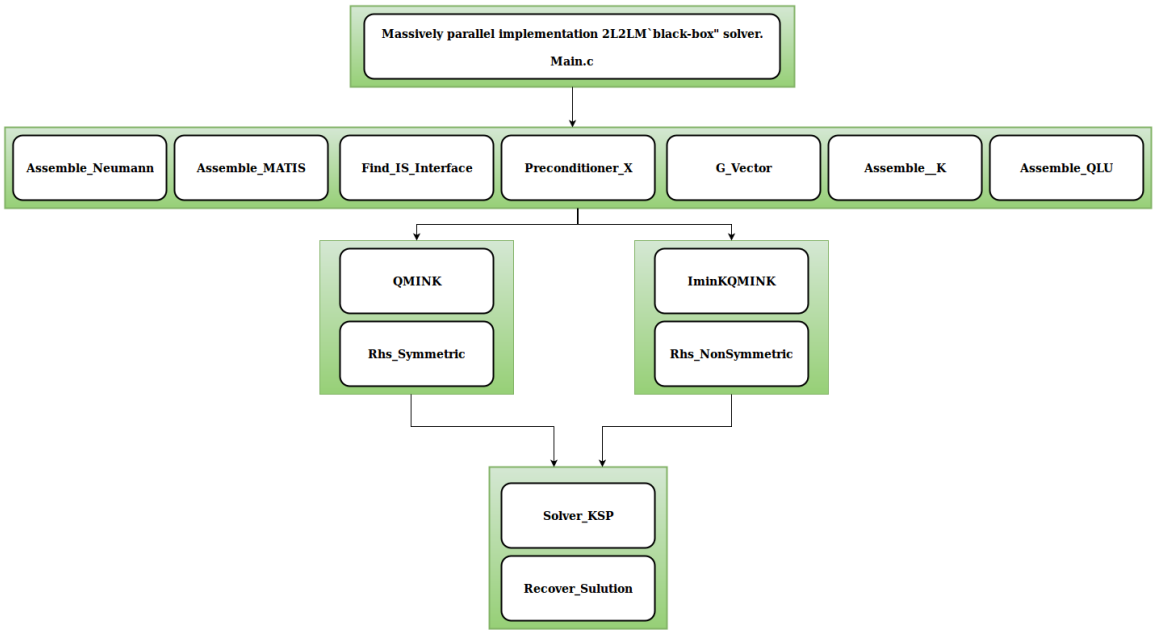


Figure 6.5: 2-Level 2-Lagrange Multiplier solver diagram.

- MPICH [2], a high performance and widely portable implementation of the message passing interface (MPI) standard.
- PETSc [3], a portable, extensible toolkit for scientific computation.
- Triangle [4], a two-dimensional quality mesh generator and delaunay triangulator.

The massively parallel implementation of the 2-Level 2-Lagrange Multiplier methods solver code can be split in three main layers. The first layer is the group of functions that are used in order refine and distribute the mesh, create the Neumann matrices, create the local to global numbering and find the interface vertices, initialise the Preconditioner P, create the parallel vector g in the right side of systems (6.4), (6.5) and finally create the LU decomposition for each local matrix Q_i of the

[2] `http://www.mpich.org/`
[3] `http://www.mcs.anl.gov/petsc/index.html`
[4] `https://www.cs.cmu.edu/~quake/triangle.html`

parallel block diagonal matrix Q. In the second layer, depending on the selection of the method (2LS2LM or 2L2LM), we initialize the appropriate Symmetric or non Symmetric systems. Finally in the third layer we solve the system and each processor retrieves the local discrete solution.

We can see these three layers in Figure 6.5 where we have created a diagram of the functions that are included in the main.c file. In the main.c file we start by defining the number of the refinement level of the mesh triangles, the number of subdomains, the Robin parameter and the type of the method that should be used, 2LS2LM or 2L2LM. Then we initialise the `MPI_COMM_WORLD` communicator using `PetscInitialize`, a function provided by the PETSc library. In the next steps of our code we create the mesh, the local Neumann matrices and finally we solve in parallel systems (6.4), (6.5).

Now we give some more details for each function of the diagram in Figure 6.5. We have implemented the function `Assemble_Neumman` which takes as inputs the processor id, the level of refinement and the number of subdomains; in our experiments the number of subdomains is equal to the number of processors. As we can see from the diagram in Figure 6.6 `Assemble_Neumman` contains several functions. The most important function that is included in `Assemble_Neumman` is function `Proc_Stiffness_Mat`. In the `Proc_Stiffness_Mat` function we create the initial coarse mesh which is identical for every processor. In order to create the coarse mesh and also have some flexibility on the geometries that we could test in large scale environments we have implemented the following three different functions. The function `Uniform_Mesh` refines uniformly our initial $2D$ mesh. The functions `Disk_Mesh` and `Square_Mesh` are essentially wrappers for the Triangle library [63]. We define some input parameters, for example the center and the length of the radius for the case of a disk, and then we use the Triangle library in order to get the Delaunay triangulation for these simple geometries. Additionally we have implemented the `Create_Offset` and `Set_Global_Offset` to create and set the offset of the vertices as described in detail in Subsection 6.2.1.

The function `Create_Sequential_Triangle` refines the initial triangles of the coarse mesh in a way that the numbering of the vertices follow a particular pattern, something that helps us to find the local and global numberings of the vertices when this is needed. Finally we calculate the elements of the Stiffness matrix A and then we use the functions provided by PETSc in order to set these values and create the local Neumann matrices A_{Ni}.

The `Find_IS_Interface` function finds the interface points of the global mesh. We use the `IS` data structure, provided by PETSc, in order to store the local to global mapping of the interface vertices for each subdomain Ω_i. In this function we have also used the PETSc Vector data structure and the provided functions to do operations on them like addition and multiplication and to scatter and gather the local values of

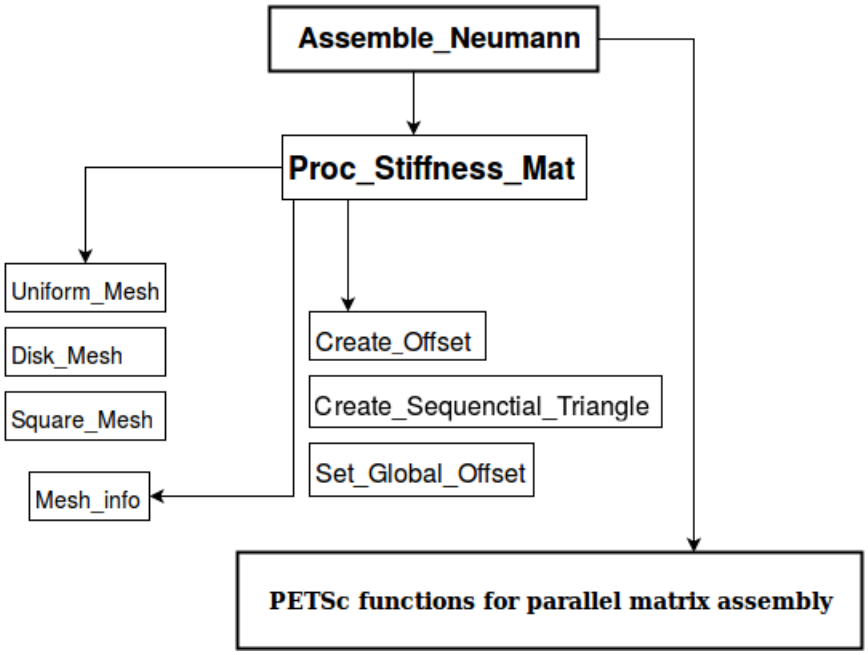


Figure 6.6: `Assemble_Neumann` function diagram.

the vectors between the processors.

In `Assemble_MATIS` we create the parallel stiffness matrix of type `MATIS`, a data structure provided by the PETSc library and holds the local to global numbering of the vertices and the local Neumann matrices A_{Ni}.

In `Preconditioner_X` we have implemented the Preconditioner P^{-1}, (6.8), as PETSc Shell matrix following the procedure in Subsection 6.2.2. This procedure includes the implementation of the J and J^T matrices as parallel `MATMPIAIJ` PETSc matrices. We have implemented the Preconditioner P in 3 different ways, hence we change the letter X such that the function corresponds to a different implementation of the Preconditioner.

In function `G_vector` we have implemented the g vector which is a common component of the right hand side for both systems (6.4) and (6.5) as a parallel PETSc vector.

After the implementation of the six functions of the first layer we have implemented the four functions of the second layer of our algorithm. The user can choose between 2LS2LM and 2L2LM hence we needed to provide the corresponding implementation for both. In function `QMINK` we implement the $Q-K$ matrix as a PETSc Shell matrix and we also implement the right hand side of 2LS2LM in funtion `RhS_Symmetric`.

For the non symmetric system 2L2LM, we implement $(I-2K)(Q-K)$ in function `IminKQMINK` and the corresponding right hand side `RhS_NonSymmetric`.

In the fourth layer of our algorithm we implement the function `Solver_KSP` where we use the parallel GMRES solver (`KSPGMRES`) which is provided by the PETSc, in order to solve the systems that we implemented in the second layer. Finally the

function `Recover_Solution` recovers the local solution u_i for any of the subdomains.

6.2.5 Comments on Scalability for the Massively Parallel Experiments

We observe that our massively parallel experiments in Subsection 6.2.3, the higher number of iterations is not always reflected in the total cpu time (wall clock time). This is partially because a significant time of our algorithm was spent in order to initialize matrices K, P, Q and for the case of the massively parallel experiments a significant amount of work was spent on the distribution of our tasks among the nodes. Here we split the cpu time cost in the following three different steps:
Step 1: Mesh manipulation and the assembly of the local Neumann matrices A_{Ni} for each subdomain.
Step 2: Initialization of matrices K, P, Q.
Step 3: GMRES parallel solver on 2L2LM.

Then we are able to estimate the cpu time costs for each category separately. In order to estimate the above costs we use the 2-Level 2-Lagrange multiplier method in order to solve problem (5.3) with the constant function $f = 1$ as a right hand side and Ω a square domain. We split Ω in $2, 4$ and 8 subdomains and we solve the corresponding problems in parallel. The size of the meshes that we use varies from 10^4 to 10^6 vertices.

In Figures 6.7, 6.8, 6.9 we can see the results of our experiments. With blue color we see the cost in cpu time of the mesh manipulation which includes the creation of the coarse mesh, the refinement of the local mesh that is assigned to each processor and the assembly of the local Neumann matrices A_{Ni} for each subdomain. With green color we observe the cost that it is associated with the initialization of the parallel matrices K, Q and the preconditioner P. Finally with red color we see the amount of cpu time that was needed by the parallel GMRES solver in order to solve the non-symmetric 2-Lagrange multiplier system. All the above cpu time costs are counted in seconds.

From Figures 6.7, 6.8, 6.9 we observe that the most significant time is spent for the initialisation of matrices K, Q, P. This cost can be associated with the time that is needed to create a parallel matrix explicitly, for the matrix K and the time that is needed to define the PETSc Shell matrices Q and P. For matrix Q for example each processor performs the LU decomposition of the local Robin problem (5.8).

The time that is needed by the GMRES solver (Step 3) in order to solve the 2L2LM system in parallel is minimal compared to the two previous steps. For example for the case that we have 8 subdmains and 10^6 vertices the total cost in terms of cpu time was 17.530 seconds. From this time 10.46 seconds were spent for the initialization

of matrices K, Q, P , 5.483 seconds for the operations related to the mesh and the assembly of the local Neumann problems A_{Ni} and 1.587 seconds needed by the GMRES solver in order to solve the 2L2LM system. This means that around 60% of the time was spent on the second matrices K, Q, P, around 31% was spent for the mesh manipulation and the assembly of the local Neumann problems and only 9% of the time was needed by the GMRES solver to solve the 2L2LM system in parallel.

From these experiments we conclude that the scalability of the current implementation is not limited by the underlying 2LM algorithms, but rather by setup costs. In future work, we hope to further parallelise these steps.

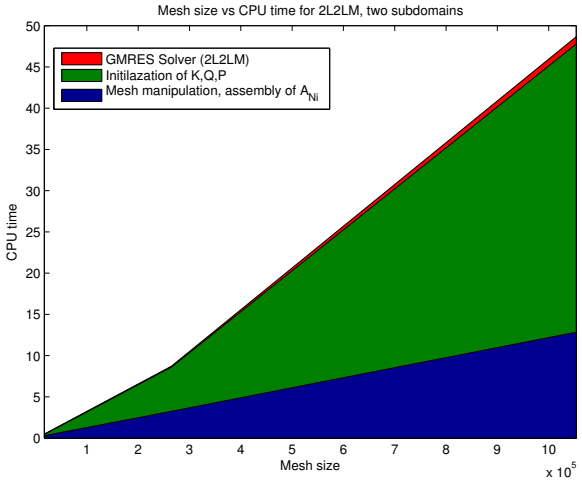


Figure 6.7: Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, two subdomains

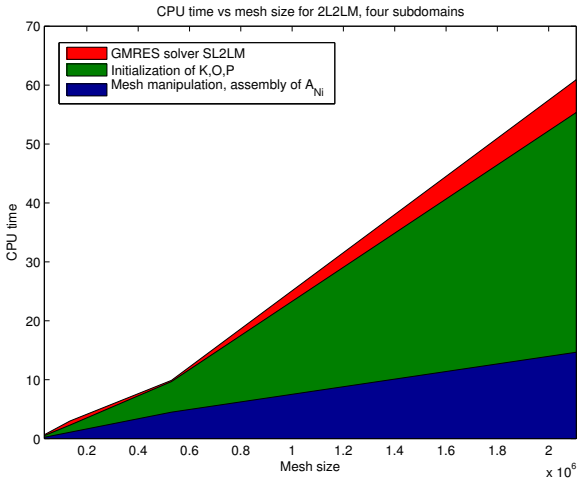


Figure 6.8: Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, four subdomains

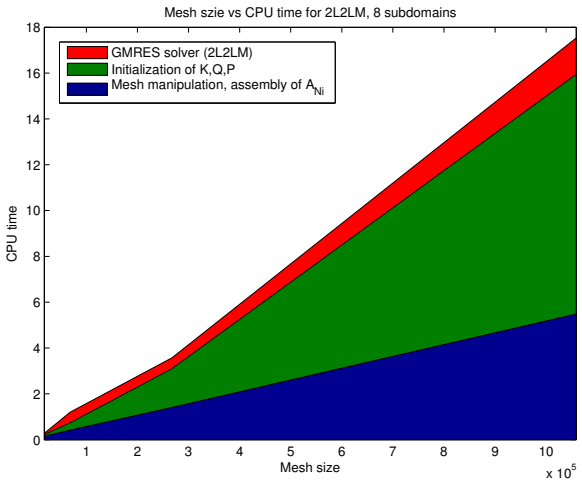


Figure 6.9: Total CPU time in seconds vs Mesh size for the 2L2LM parallel implementation, eight subdomains

Chapter 7

Conclusion and Future work

The symmetric 2-Lagrange multiplier method (2SLM) is a linear equation of the form

$$(Q - K)\lambda = -Qg \tag{7.1}$$

and the non-symmetric 2-Lagrange multiplier method (2LM) is a linear system of the form

$$(I - 2K)(Q - K)\lambda = -(I - 2K)Qg. \tag{7.2}$$

where Q is symmetric positive definite, K is an averaging operator, g is the data and λ is the unknown. From the solution λ we can can recover the solution of a problem $Au = f$ arising from the discretization of an elliptic problem for example

$$\Delta\tilde{u} = \tilde{f} \text{ in } \Omega \text{ and } \tilde{u} = 0 \text{ on } \partial\Omega \tag{7.3}$$

where the domain $\Omega \in \mathbb{R}^d$, $d = 2, 3$. In Chapter 3 we have developed the necessary background on Sobolev spaces for the solutions of elliptic PDEs and in Chapter 4 we have presented the discretization procedure for elliptic equations of elliptic problems such as (7.3), via the Finite Element Method (FEM). Moreover in Section 5.1.1 we have shown how we can obtain the 2SLM and 2LM.

One of the main goals of domain decomposition methods is to decompose the original initial domain Ω into smaller subdomains Ω_i, $i = 1, \ldots, p$ and then to solve problems such as (7.3) in parallel using a solver such as GMRES or MINRES. In Theorem 5.5.15 , [48], it is proved that the condition number of 2SLM and 2LM increases unboundedly when the number of subdomains p increases. In this Thesis we introduced the new 2-level 2-Lagrange multiplier methods featuring a coarse grid correction. The symmetric 2-Level 2-Lagrange multiplier method (2L2SLM) is a linear equation of the form,

$$P^{-1}(Q - K)\lambda = -P^{-1}Qg$$

and the non-symmetric 2-Lagrange multiplier method (2L2LM) is a linear system of the form

$$P^{-1}(I - 2K)(Q - K)\lambda = -P^{-1}(I - 2K)Qg. \tag{7.4}$$

The preconditiner P, a "coarse grid correction", leads to algorithms that scale weakly. We have estimated the condition number of our new methods and we have shown that the condition numbers scales weakly. This opens the door for the massively parallel implementation of these methods. Our algebraic estimates apply to the elliptic case for general domains and subdomains with general elliptic PDEs and the cross points do not pose any special difficulty. The theory has been confirmed by several sets of numerical experiments. The large scale implementation of 2LS2LM and 2L2LM is provided and massively parallel experiments performed on the HECToR supercomputer on thousands of processors. For the massively parllel implementation we have used the PETSc library, [6]. From these experiments we have concluded that the combination of the GMRES solver (with or without restart) with the 2L2LM is superior to 2LS2LM. The numerical experiments suggest that iteration count for the 2-Level non-symmetric method is $O(H/h)^{1/4}$.

The communication overhead for PETSc on HECToR were significant, and we observed that in some cases that problems of a certain size require a longer wall clock time when more processors are added.

The slow down is caused by expensive setup costs that are not yet efficiently parallelised. Future work will include parallelising the setup costs and finding specialized preconditioners, e.g for heterogeneous and multiscale problems.

Appendices

Appendix A

Notes on Massively Parallel Implementation

In this Appendix we include some parts of the code related to the parallel implementation that corresponds to Section 6.2 and article [41]. We will try to provide some insight on how we used PETSc in order to implement our methods. Additionally, we have tried to combine the code that we used for our massively parallel implementation with the deal.II library and we have also worked on a C++ parallel library that we call femH, inspired by deal.II [7] and the ifem [17] libraries. Some functions part of C++ classes of the femH code are included in Chapters 4 and 5.

All codes are open-source and available on-line in our repository: `https://bitbucket.org/modios`. All experiment where conducted in Unix/Linux environments.

During the progress of this thesis we found that the following list of libraries and tools, were very useful, in order to implement Domain Decomposition methods and then to visualise the results.

1. Message Passing Interface (MPI): `http://www.mpich.org/`
2. Portable, Extensible Toolkit for Scientific Computation: `http://www.mcs.anl.gov/petsc/`
3. Triangle, a Two-Dimensional Quality Mesh Generator and Delaunay Triangulator: `https://www.cs.cmu.edu/~quake/triangle.html`
4. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities: `http://geuz.org/gmsh/`
5. METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering: `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`
6. Boost, free peer-reviewed portable C++ source libraries.: `http://www.boost.org/`

7. deal.II, an open source finite element library: `https://www.dealii.org/`
8. ParaView, an open-source, multi-platform data analysis and visualization application: `http://www.paraview.org/`

A.1 `MATIS` "black box" solver

To run the `MATIS` code we need to have properly installed the $MPICH$ and $PETSc$ libraries in our system. In the PETSc web page there is a detailed documentation on how to install Petsc on the system, `http://www.mcs.anl.gov/petsc/documentation/installation.htmltml`

As we have explained in detail in Chapter 6, the user provides a PETSc matrix of type `MATIS` and then the solver uses the information is stored in the `MATIS` matrix, which includes the local Neumman problems and the local to global mapping of the nodes. Then the 2-Level 2 Lagrange multiplier methods are used to return the desired local solution for each subdomain.

In Algorithm 9 we include the main function. In the main.c file the steps of this implementation are clearly presented.

In lines 1 to 10 we include the necessary libraries. The "triangle.h" provides the `API` function calls for the Triangle library, "petsc.h" is necessary in order to include the PETSc libary and in "Functions2LM.h" we have "squeezed" all the necessary functions that we needed to implement our problem.

In line 14 the user initialises the number of nodes on the edges of the triangles. For example, triangles with 3 nodes per edge will be consisted in total from 15 nodes, since we have 3 times the number of edges, which is 9, plus the interior nodes, which is 3, plus the initial vertices of the triangle which are again 3. A formula that gives directly the number of node of the triangle if we know the number of nodes per edge is: $NEdges(NEdges-1)/2$. In line 15, we set the number of times that we apply uniform refinement on the seed mesh in order to obtain the triangles of the coarse mesh. The number of triangles of the coarse mesh should be equal to the number of processors. In lines 28 to 32 we initialise the global communicator `PETSC_COMM_WORLD` and then we get the rank and the size of this communicator. We also set the Robin parameter to be equal to $\sqrt{1/NEdges}$.

Function `Assemble_Neumann`, assembles the Local Neumann Stiffness matrices for each subdomain and provides the global numbering of the mesh. In `Assemble_MATIS` we use the information from `Assemble_Neumann` to assemble the `MATIS` matrix. Function `Find_IS_Interface` takes the `MATIS` matrix as an input and extracts all the necessary information that is needed in order to proceed. This information consists of the interface nodes, the interior nodes and the size of the arrays that hold their values.

Algorithm 9 main.c for the `MATIS` solver.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <assert.h>
#include <time.h>
#include "triangle.h"
#include "petsc.h"
#include "Functions2LM.h"


int main(int argc,char** argv){
  int NEdges=7;
  int subdivisions=1;

  int sizeALoc,precon=0;
  int  *localnumber,*globalumber,sum,*interface,*interfaceloc,count_interface,c  ount_interior,*interiorloc;

  PetscScalar Robin_par=0.73;
  PetscMPIInt rank,size;
  Mat M,K,Q,QminK,P;
  Mat A;
  Vec w,f,g,rhs;
  KSP solver;
  int n=1;
  Vec solution,lambda;


  PetscInitialize(&argc,&argv,(char *)0,help);
  MPI_Comm_rank(PETSC_COMM_WORLD,&rank);
  MPI_Comm_size(PETSC_COMM_WORLD,&size);
  PetscOptionsGetInt(NULL,"-n",&n,NULL);
  Robin_par=sqrt(1.0/(NEdges));


  Assemble_Neumann(rank,subdivisions,NEdges,\
                   &localnumber,&globalumber,&sizeALoc,&A,&f);

  Assemble_MATIS(A,localnumber,globalumber,sizeALoc,&M);


  Find_IS_Interface(M,&sum,&interface,&interfaceloc,&count_interface,&w,&interiorloc,&count_interior);

  Assemble_K(sum,interface,count_interface,&K,rank,size,w);

  Preconditioner_3(&P,M,count_interface,K,rank,size);

  G_Vector(M,&g,f,interfaceloc,count_interface,interiorloc,count_interior);


  Assemble_QLU(M,interface,interfaceloc,count_interface,Robin_par,&Q,&solver);

  if(n==0){
    QMINK(&QminK,K,Q,count_interface);
    RHSSymetric(&rhs,Q,g,count_interface);
    if(rank==0)  puts("=======␣␣␣␣␣␣␣␣Symmetric␣2LM␣Choosen␣␣␣␣␣␣␣========");
  }else{
    IminKQMINK(&QminK,K,Q,count_interface);
    RHSNONSymetric(&rhs,Q,K,g,count_interface);
    if(rank==0)  puts("=======␣␣␣␣␣␣␣␣Non-Symmetric␣2LM␣Choosen␣␣␣␣␣========");
  }


  Solver_KSP(QminK,P,count_interface,rank,precon,rhs,&lambda);

  Recover_Sulution(solver,f,lambda,&solution,interfaceloc,count_interface);

  MatDestroy(&K);
  MatDestroy(&P);
  MatDestroy(&Q);
  MatDestroy(&QminK);
  VecDestroy(&rhs);

  PetscFinalize();
  return 0;
}
```

In `Asseble_K` we use the interface points vector in order to determine what is the multiplicity of each node, or in different words how many triangles share each node. This information together with the total number of interface points counted with their multiplicity, is adequate in order to assemble the average orthogonal projection matrix K. Moreover function $\texttt{Preconditioner}_3$ defines the preconditioner as PETSc shell matrix. Since we use iterative solvers like GMRES for the outer iterations, we don't have to assemble explicitly the matrices.

The next step is to assemble the right hand side vector g, of "cumulative fluxes". Then, in a similar way like the case like the one of preconditioner P, we define Q as a shell matrix, and we use LU decomposition for the local robin sub-problems. In order to retrieve the solution, since the local Robin sub-problems are Positive definite, in this step instead of using an LU decomposition we can use either the Cholesky decomposition or any other non direct method that applies to symmetric and positive definite matrices, like the Conjucate Gradient method.

Since matrices K and Q has been already defined or assembled, we can create the shell matrices that correspond to $Q - K$ and $(I - 2K)(Q - K)$, which are named `QMINK` and `IminKKQMINK` respectively. The user has the option to decide which one of the two 2-Lagrange multiplier methods, the symmetric or non-symmetric, wants to use.

Finally we use the function `Solver_KSP`, a function that gets as an input the matrices, $Q - K$, the preconditioner P^{-1} and the right hand side vector G, and uses a parallel GMRES solver as an outer iteration in order to return the Lagrange multipliers for each processor, something that eventually leads to the approximate solution of the problem.

In order to get the "primal" solution u, we will substitute the acquired Lagrange multiplier vector λ on each local Robin sub-problem, and we will solve for each sub-domain. In the last few lines of the code, we free the dynamically allocated memory and we terminate the `MPI` execution environment.

On the following subsection we will present some key functions of the"`MATIS` black box solver" code.

Finding the Interface

In Algorithm 10 we assume that the user has provided the `MATIS` matrix. The function `Find_IS_Interface` in line 14 uses the PETSc function, $\texttt{MatISGetLocalMat}(\texttt{M}, \&\texttt{C})$, in order to extract the local Neumann matrix C. In the next line we get the local to global mapping and we assign it to variable `ltog`, through function $\texttt{MatGetLocalToGlobalMapping}(\texttt{C}, \&\texttt{ltog}, \&\texttt{ltog})$. In lines 35 to 37 we create a vector x that holds the multiplicity of each interface node, i.e if node i is an artificial interface point and is shared between 4 subdomains, hence $x[i] = 4$. Then we get the vector w of

Algorithm 10 Find_IS_Interface function.

```
extern PetscErrorCode Find_IS_Interface(Mat M,int *sumf,\
                int **interfacef,int**interfacelocf,int*count_interfacef,\
                Vec *wf,int**interiorlocf,int* count_interiorf){
        Mat C;
        Vec x,y,w;
        IS is1;
        PetscInt n,*interface,*interfaceloc,count_interface,i,*interiorloc;//step=1,
        ISLocalToGlobalMapping ltog;
        PetscErrorCode ierr;
        PetscScalar  one=1.0,zero=0.0, sumt;
        int     sum=0;
        PetscScalar *w_array;
        PetscInt *globalnumbering,*localnumbering;
        MatISGetLocalMat(M,&C); //Get the local matrix
        MatGetLocalToGlobalMapping(C,&ltog,&ltog); //get the global to local indices
        ISLocalToGlobalMappingGetSize(ltog,&n);

        ierr = VecCreate(PETSC_COMM_WORLD,&x);CHKERRQ(ierr);
        ierr = VecSetSizes(x,n,PETSC_DECIDE);CHKERRQ(ierr);
        ierr = VecSetFromOptions(x);CHKERRQ(ierr);
    ierr = VecSetLocalToGlobalMapping(x,ltog);CHKERRQ(ierr);
        for (i=0; i<n; i++) {
                ierr = VecSetValuesLocal(x,1,&i,&one,INSERT_VALUES);CHKERRQ(ierr);
        }
        ierr = VecAssemblyBegin(x);CHKERRQ(ierr);
        ierr = VecAssemblyEnd(x);CHKERRQ(ierr);
        VecSum(x,&sumt);
        VecDestroy(&x);
        sum=sumt;
        ierr = VecCreate(PETSC_COMM_WORLD,&x);CHKERRQ(ierr);
        ierr = VecSetSizes(x,PETSC_DECIDE,sum);CHKERRQ(ierr);
        ierr = VecSetFromOptions(x);CHKERRQ(ierr);
        ierr = VecSet(x,zero);CHKERRQ(ierr);
        ierr = VecSetLocalToGlobalMapping(x,ltog);CHKERRQ(ierr);
        for (i=0; i<n; i++) {
                ierr = VecSetValuesLocal(x,1,&i,&one,ADD_VALUES);CHKERRQ(ierr);
        }
        /******* We Assemble x and duplicate x to w **********/
        ierr = VecDuplicate(x,&w);CHKERRQ(ierr);
        ierr = VecAssemblyBegin(x);CHKERRQ(ierr);
        ierr = VecAssemblyEnd(x);CHKERRQ(ierr);
        /********   The vector W and we make it local ********/
        ierr = VecSet(w,one);CHKERRQ(ierr);CHKERRQ(ierr);
        ierr = VecPointwiseDivide(w,w,x);CHKERRQ(ierr);


        /*************************************************/
        PetscMalloc(n*sizeof(PetscInt),&globalnumbering);
        PetscMalloc(n*sizeof(PetscInt),&localnumbering);
        PetscMalloc(n*sizeof(PetscScalar),&w_array);
        for(i=0;i<n;i++){
                localnumbering[i]=i;
        }
        ISLocalToGlobalMappingApply(ltog,n,localnumbering,globalnumbering);
        ISCreateGeneral(MPI_COMM_SELF,n,globalnumbering,PETSC_COPY_VALUES,&is1);
        VecGetSubVector(w,is1,&y);
        VecGetArray(y,&w_array);

        count_interface=0;
        for(i=0;i<n;i++){
                if(w_array[i]<1){
                        count_interface=count_interface+1;
                }
        }
        /***************Interface Vertices****************/
        PetscMalloc(count_interface*sizeof(PetscInt),&interface);
        PetscMalloc(count_interface*sizeof(PetscInt),&interfaceloc);
        PetscMalloc((n-count_interface)*sizeof(PetscInt),&interiorloc);
        count_interface=0;
        int count_interior=0;
        for(i=0;i<n;i++){
                if(w_array[i]<1){
                        interface[count_interface]=globalnumbering[i];
                        interfaceloc[count_interface]=i;
                        count_interface=count_interface+1;
                }else{
                        interiorloc[count_interior]=i;
                        count_interior=count_interior+1;
                }
        }
        /*print_array_int(interfaceloc,count_interface,1); //TO REMOVE;
        print_array_int(interiorloc,count_interior,2);*/
        PetscFree(localnumbering);
        PetscFree(globalnumbering);
        VecRestoreArray(y,&w_array);
        VecRestoreSubVector(w,is1,&y);
        VecDestroy(&y);
        VecDestroy(&x);
        ISDestroy(&is1);
        PetscFree(w_array);
        VecDestroy(&x);
        ISLocalToGlobalMappingDestroy(&ltog);



        *wf=w;
        *count_interfacef=count_interface;
        *interfacef=interface;
        *interfacelocf=interfaceloc;
        *interiorlocf=interiorloc;
        *count_interiorf=count_interior;
        *sumf=sum;

        return(0);
}
```

weights, that will be used in order to assemble the averaging operator K. Afterwards we observe that it is easy to identify the interior points, since if $w\_array[i] = (1/x[i]) < 1$ holds, this means that the node i lays on the artificial interface Γ.

Assembly of Matrix K

Assume that G is the interface interpolation operator that interpolates from the global interface node to the local ones and W is a diagonal matrix with the vector of weights w, as it is defined in the previous subsection, in the diagonal. Then we can formulate K as $K = WGG^T$. This is exactly what we do in Algorithm 11, we assemble the diagonal parallel matrix W, and the matrices G and G^T. Then we use the `MatMatMult` PETSc function in order to calculate the product $K = WGG^T$ and finally we retrieve K.

Assembly of Matrix Q as a PETSc Shell Matrix

The Matrix Q is not assembled explicitly, instead we define this matrix as a PETSc Shell matrix, cf. Algorithms 12 and 13. In order to define the Q matrix as a PETSc Shell matrix, we need to define some basic properties of the matrix, such as the local and the global size the matrix. Pointer $*$`datainQ` holds the information that is necessary for the matrix-vetor multiplication routine which we have to define and is passed to the Shell matrix through function
`MatCreateShell`(`PETSC_COMM_WORLD`, `count_interface`
`count_interface`, `PETSC_DETERMINE`, `PETSC_DETERMINE`, `datainQ`, &`Q`).
Function `MatShellSetOperation`(`Q`, `MATOP_MULT`, (`void`($*$)(`void`))`QLUmult`), sets the matrix-vector multiplication operations that describe matrix Q and which are implemented in function `QLUmult`.

Since Q is block-diagonal we can easily calculate its blocks independently in parallel. Each block of Q is defined as $Q_k = a(S_k + aI_k)^{-1}$. Hence in order to calculate the product $Q_k\lambda_k$ we define the `QLUmult` function. From system 6.1 we have that $Q_k\lambda_k = au_{\Gamma k}$, hence in this function, instead of assembling or defining the local Schur complements, we perform an equivalent operation by solving,

$$\begin{bmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma\Gamma k} + aI \end{bmatrix} \begin{bmatrix} u_{Ik} \\ u_{\Gamma k} \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_k \end{bmatrix}$$

and then we scale $u_{\Gamma k}$ by the robin parameter a.

Next, depending on which of the two 2-Lagrange multiplier methods we want to use the, Symmetric or Non-symmetric, we choose functions `QMINK`(&`QminK`, `K`, `Q`, `count_interface`) and `RHSSymetric`(&`rhs`, `Q`, `g`, `count_interface`) or,
`IminKQMINK`(&`QminK`, `K`, `Q`, `count_interface`) and

Algorithm 11 Assembly of Matrix K.

```
1  extern PetscErrorCode Assemble_K(PetscInt sum,int* interface,          \
2                                   int count_interface,Mat *Kf,int rank,int size,Vec w){
3    Mat G,GT,W,K,GW;
4
5    int flag=1;
6    int i,rowStartG,rowEndG,row,col,count,rowStartW,rowEndW;
7    PetscScalar one=1.0,val;
8
9
10   MatCreate(PETSC_COMM_WORLD,&G);
11   MatSetSizes(G,count_interface,PETSC_DECIDE,PETSC_DETERMINE,sum);
12   MatSetType(G,MATAIJ);
13   MatSetUp(G);
14         //MatMPIAIJSetPreallocation(G,1,NULL,1,NULL);
15   MatSetFromOptions(G);
16   MatGetOwnershipRange(G,&rowStartG,&rowEndG);
17
18
19   count=0;
20   for(i=rowStartG;i<rowEndG;i++){
21     col=interface[count];
22     count=count+1;
23     MatSetValues(G,1,&i,1,&col,&one,INSERT_VALUES);
24   }
25
26
27   MatAssemblyBegin(G,MAT_FINAL_ASSEMBLY);
28   MatAssemblyEnd(G,MAT_FINAL_ASSEMBLY);
29
30
31   MatCreate(PETSC_COMM_WORLD,&W);
32   MatSetSizes(W,PETSC_DECIDE,PETSC_DECIDE,sum,sum);
33   MatSetType(W,MATAIJ);
34   //MatSetUp(W);
35   MatMPIAIJSetPreallocation(W,1,NULL,1,NULL);
36   MatSetFromOptions(W);
37   MatGetOwnershipRange(W,&rowStartW,&rowEndW);
38
39   for(i=rowStartW;i<rowEndW;i++){
40     val=0.0;
41     MatSetValues(W,1,&i,1,&i,&val,INSERT_VALUES);
42   }
43
44   MatAssemblyBegin(W,MAT_FINAL_ASSEMBLY);
45   MatAssemblyEnd(W,MAT_FINAL_ASSEMBLY);
46   MatDiagonalSet(W,w, INSERT_VALUES);
47   VecDestroy(&w);
48
49   MatCreate(PETSC_COMM_WORLD,&GT);
50   MatSetSizes(GT,PETSC_DECIDE,count_interface,sum,PETSC_DETERMINE);
51   MatSetType(GT,MATAIJ);
52   MatSetUp(GT);
53   //MatMPIAIJSetPreallocation(G,1,NULL,1,NULL);
54   MatSetFromOptions(GT);
55
56   int j;
57   //MatSetOption(GT,MAT_ROW_ORIENTED,PETSC_FALSE);
58   MatGetOwnershipRangeColumn(GT,&rowStartG,&rowEndG);
59
60   count=0;
61   for(j=rowStartG;j<rowEndG;j++){
62     row=interface[count];
63     col=j;
64     MatSetValues(GT,1,&row,1,&col,&one,INSERT_VALUES);
65     count=count+1;
66   }
67
68   MatAssemblyBegin(GT,MAT_FINAL_ASSEMBLY);
69   MatAssemblyEnd(GT,MAT_FINAL_ASSEMBLY);
70
71   //MatView(GT,PETSC_VIEWER_STDOUT_WORLD);
72   MatMatMult(G,W,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&GW);
73   MatDestroy(&G);
74   MatDestroy(&W);
75   MatMatMult(GW,GT,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&K);
76   //MatView(K,PETSC_VIEWER_STDOUT_WORLD);
77   MatDestroy(&GW);
78   MatDestroy(&GT);
79   *Kf=K;
80   return 0;
81 }
```

Algorithm 12 Assembly of Matrix Q (Shell Matrix routines).

```
typedef struct{
        PetscScalar Robin_parin;
        Vec temp,z;
        //IS is;
        int *interfaceloc;
        int sizeinterface;
        KSP solver;
}QLUdata;

extern int QLUmult(Mat A,Vec lambda,Vec Qlambda){

        QLUdata *data;
        PetscScalar *values,val,*values2;
        int i,m,Rlow,Rend;
        VecGetOwnershipRange(lambda,&Rlow,&Rend);
        MatShellGetContext(A,&data);
        VecGetArray(lambda,&values);
        for(i=0;i<(data->sizeinterface);i++){
                val=values[i];
                m=data->interfaceloc[i];
                VecSetValues(data->temp,1,&m,&val,INSERT_VALUES);
        }
        VecRestoreArray(lambda,&values);
        PetscFree(values);
        VecAssemblyBegin(data->temp);
        VecAssemblyEnd(data->temp);

        KSPSolve(data->solver,data->temp,data->z);

        VecScale(data->z,data->Robin_parin);
        PetscMalloc(data->sizeinterface*sizeof(PetscScalar),&values2);
        VecGetArray(data->z,&values2);
        for(i=Rlow;i<Rend;i++){
            m=data->interfaceloc[i-Rlow]+Rlow;
            val=values2[i-Rlow];
            VecSetValues(Qlambda,1,&m,&val,INSERT_VALUES);
          }
          VecAssemblyBegin(Qlambda);
          VecAssemblyEnd(Qlambda);
          VecRestoreArray(data->z,&values2);

        return 0;
}
```

`RHSNONSymetric`(&`rhs`, `Q`, `K`, `g`, `count_interface`). The matrices $Q - K$ and $(I - 2K)(Q - K)$ are defined as PETSc Shell matrices.

To recover the full local solution we substitute back into the initial local Robin system and we use a PETSc KSP solver in order to solve the local Robin system.

Assembly of peronditioner P as a PETSc Shell Matrix

In Algorithms 17, 18, 19 and 20 we include the functions an all the routines that are necessary in order to define the PETSc Shell matrix P.

A.2 The deal.II Implementation

We assumed that it might be useful for some users, to use a modified parallel 2-Lagrange implementation along with a well established open source Finite element library, like deal.ii. Hence we modified the above functions and in order to get a more user friendly code where it is easier to run for different examples and geometries in 2D and 3D. In Algorithm 21 we present the central class of the code named `L2Solver`. The full code can be found in the repository: `https://bitbucket.org/modios/deal_2lm`

Algorithm 13 Assembly of Shell Matrix Q (main part of the function).

```
1
2  extern PetscErrorCode Assemble_QLU(Mat M,int* interface,int* interfaceloc,\
3                                                              int count_interface,PetscScalar Robin_par,Mat *Qf,KSP* solverf){
4          Mat Q,RobI;
5          int i,m,n;
6          Mat FC;
7          Vec temp,z;
8          PetscInt nz=10,col;
9          PetscScalar one=1.0;
10         KSP  solver;
11         PC pc;
12         QLUdata* datainQ;
13
14
15         MatISGetLocalMat(M,&FC); //Get the local matrix
16
17         MatGetSize(FC,&m,&n);
18         VecCreateSeq(PETSC_COMM_SELF,n,&temp);
19
20         VecDuplicate(temp,&z);
21         MatCreate(PETSC_COMM_SELF,&RobI);
22         MatSetSizes(RobI,m,n,m,n);
23         MatSetType(RobI,MATSEQAIJ);
24         MatSeqAIJSetPreallocation(RobI,nz,NULL);
25
26         for(i=0;i<count_interface;i++){
27                 col=interfaceloc[i];
28          MatSetValues(RobI,1,&col,1,&col,&Robin_par,INSERT_VALUES);
29         }
30         MatAssemblyBegin(RobI,MAT_FINAL_ASSEMBLY);
31         MatAssemblyEnd(RobI,MAT_FINAL_ASSEMBLY );
32         MatAXPY(RobI,one,FC,DIFFERENT_NONZERO_PATTERN);
33         MatDestroy(&FC);
34
35          KSPCreate(PETSC_COMM_SELF,&solver);
36          KSPSetType(solver, KSPPREONLY);
37          KSPGetPC(solver,&pc);
38          PCSetType(pc,PCLU);
39          KSPSetOperators(solver,RobI,RobI,SAME_PRECONDITIONER);
40          KSPSetUp(solver);
41
42          PetscMalloc(sizeof(QLUdata),&datainQ);
43          datainQ->Robin_parin=Robin_par;
44          datainQ->interfaceloc=interfaceloc;
45          datainQ->sizeinterface=count_interface;
46          datainQ->temp=temp;
47          datainQ->z=z;
48          datainQ->solver=solver;
49         MatCreateShell(PETSC_COMM_WORLD,count_interface,count_interface,PETSC_D        ETERMINE,\
   PETSC_DETERMINE,datainQ,&Q);
50         MatShellSetOperation(Q,MATOP_MULT,(void(*)(void))QLUmult);
51         MatSetFromOptions(Q);
52         MatDestroy(&RobI);
53         *Qf=Q;
54         *solverf=solver;
55
56         return 0;
57 }
```

Algorithm 14 Assembly of Shell Matrix $Q - K$

```
typedef struct{
  Mat K,Q;
  Vec z,ll,lm,lz;
}QminKdata;


extern PetscErrorCode QminKmul_func(Mat QminKmul,Vec x, Vec y){

  QminKdata *QmK;
  PetscScalar minusOne=-1.0;

  MatShellGetContext(QminKmul,&QmK);
  MatMult(QmK->K,x,QmK->z);
  MatMult(QmK->Q,x,y);
  VecAXPY(y,minusOne,QmK->z);

  return 0;
}
extern PetscErrorCode QMINK(Mat *QminKf,Mat K, Mat Q,int count_interface){

  QminKdata *datain;
  Mat QminK;
  Vec z;

  PetscMalloc(sizeof(QminKdata),&datain);
  datain->K=K;
  datain->Q=Q;
  VecCreate(PETSC_COMM_WORLD,&z);
  VecSetSizes(z,count_interface,PETSC_DECIDE);
  VecSetFromOptions(z);
  VecAssemblyBegin(z);
  VecAssemblyEnd(z);
  datain->z=z;
  MatCreateShell(PETSC_COMM_WORLD,count_interface,count_interface,PETSC_DETERMINE,\
                 PETSC_DETERMINE,datain,&QminK);
  MatShellSetOperation(QminK,MATOP_MULT,(void(*)(void))QminKmul_func);
  MatSetFromOptions(QminK);
  *QminKf=QminK;
  return 0;
}
```

Algorithm 15 Assembly of Shell Matrix $(I - 2K)Q - K$

```
extern PetscErrorCode IminKmulQminK_func(Mat IminKmulQminK,Vec x, Vec y){

  QminKdata *QmK;
  PetscScalar minusOne,minusTwo;

  MatShellGetContext(IminKmulQminK,&QmK);

  MatMult(QmK->K,x,QmK->ll);
  MatMult(QmK->Q,x,QmK->z);
  minusOne=-1.0;
  minusTwo=-2.0;

  VecWAXPY(QmK->lz,minusOne,QmK->ll,QmK->z);
  MatMult(QmK->K,QmK->lz,QmK->lm);

  VecCopy(QmK->lz,y);
  VecAXPY(y,minusTwo,QmK->lm);

  return 0;
}



extern PetscErrorCode IminKQMINK(Mat *IminQminKf,Mat K, Mat Q,int count_interface){

  QminKdata *datain;
  Mat QminK;
  Vec z;

  PetscMalloc(sizeof(QminKdata),&datain);

  datain->K=K;
  datain->Q=Q;
  VecCreate(PETSC_COMM_WORLD,&z);
  VecSetSizes(z,count_interface,PETSC_DECIDE);
  VecSetFromOptions(z);
  VecDuplicate(z,&(datain->ll));
  VecDuplicate(z,&(datain->lz));
  VecDuplicate(z,&(datain->lm));
  datain->z=z;
  MatCreateShell(PETSC_COMM_WORLD,count_interface,count_interface,PETSC_DETERMINE,\
                 PETSC_DETERMINE,datain,&QminK);
  MatShellSetOperation(QminK,MATOP_MULT,(void(*)(void))IminKmulQminK_func);
  MatSetFromOptions(QminK);

  *IminQminKf=QminK;
  return 0;
}
```

Algorithm 16 Assembly of Shell Matrix $(I - 2K)(Q - K)$

```
int RecoverSulution(KSP solver,Vec f,Vec l,Vec* solution,int* interface,int size_intinterface){
  Vec temp;
  PetscScalar *lambdarray,one=1.0;
  VecDuplicate(f,&temp);
  VecZeroEntries(temp);
  VecGetArray(l,&lambdarray);
  VecSetValues(temp,size_intinterface,interface,lambdarray, INSERT_VALUES);
  VecAYPX(f,one,temp);
  KSPSolve(solver,f,temp);
  VecView(temp,PETSC_VIEWER_STDOUT_SELF);
  VecDestroy(&l);
  VecDestroy(&f);
  return 0;
}
```

Algorithm 17 Preconditioner support functions

```
static PetscErrorCode LExpicit(Mat *Lf,Mat J,Mat JT,Mat K,int count_interface,int size){
  Mat C,Ident,KJ;
  int rowStart,rowEnd,i;
  PetscScalar minusone=-1.0,one=1.0;

  MatCreate(PETSC_COMM_WORLD,&Ident);
  MatSetSizes(Ident,PETSC_DECIDE,PETSC_DECIDE,size,size);
  MatSetUp(Ident);
  MatSetFromOptions(Ident);
  MatGetOwnershipRange(Ident, &rowStart,&rowEnd);

  for (i=rowStart; i<rowEnd; i++){
    MatSetValue(Ident,i,i,one,INSERT_VALUES);
  }
  MatAssemblyBegin(Ident,MAT_FINAL_ASSEMBLY);
  MatAssemblyEnd(Ident,MAT_FINAL_ASSEMBLY);
  MatMatMult(K,J,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&KJ);
  MatMatMult(JT,KJ,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&C);
  MatAXPY(C,minusone,Ident,DIFFERENT_NONZERO_PATTERN);
  MatDestroy(&Ident);
  MatDestroy(&KJ);
  *Lf=C;
  return 0;
}

int MatParToSeq(Mat *OUTMAT,Mat INMAT,int size,int rank){
  PetscInt rowStartK,rowEndK,Globalrows,Globalcolumns;
  Mat *Lseq,F;
  PetscErrorCode ierr;
  IS *is;

  ierr= MatGetSize(INMAT,&Globalrows,&Globalcolumns);CHKERRQ(ierr);
  ierr=MatGetOwnershipRange(INMAT,&rowStartK,&rowEndK);CHKERRQ(ierr);
  PetscMalloc(sizeof(IS **),&is);

  ISCreateStride(PETSC_COMM_WORLD,Globalcolumns,0,1,is);

  MatGetSubMatrices(INMAT,1,is,is,MAT_INITIAL_MATRIX,&Lseq);
  ISDestroy(is);
  PetscFree(is);
  MatDuplicate(Lseq[0], MAT_COPY_VALUES,&F);
  MatDestroyMatrices(1,&Lseq);
  *OUTMAT=F;
  return 0;
}
```

Algorithm 18 Preconditioner Shell Matrix routines

```
typedef struct{
  Mat J,JT;
  Vec TempJ,TempJT,z,z1;
  int count_interface,flag,Rlow,RlowJ,RendJ,size,rank;
  KSP solver;
}Precondata3;

extern PetscErrorCode Precon3_func(Mat P,Vec lambda, Vec y){

  Precondata3 *datain;
  PetscScalar minusone=-1.0,one=1.0,*avec,*bvec;
  Vec vout,f;
  int i;
  IS isscat;

  MatShellGetContext(P,&datain);
  ISCreateStride(PETSC_COMM_WORLD,datain->rank==0?datain->size:0,datain->rank==0?0:datain->size,1,&isscat);
  MatMult(datain->JT,lambda,datain->TempJ);
  VecGetSubVector(datain->TempJ,isscat,&vout);

  if(datain->rank==0){
    VecGetArray(vout,&avec);
    VecCreateSeqWithArray(PETSC_COMM_SELF,1,datain->size,avec,&f);
    KSPSolve(datain->solver,f,datain->z1);
    VecRestoreArray(vout,&avec);
    VecDestroy(&f);
  }
  VecRestoreSubVector(datain->TempJ,isscat,&vout);
  if(datain->rank==0){
    VecGetArray(datain->z1,&bvec);
    for(i=0;i<datain->size;i++){
      VecSetValue(datain->z,i,bvec[i],INSERT_VALUES);
    }
    VecRestoreArray(datain->z1,&bvec);
  }
  VecAssemblyBegin(datain->z);
  VecAssemblyEnd(datain->z);
  MatMult(datain->J,datain->z,datain->TempJT);
  MatMult(datain->J,datain->TempJ,y);
  VecAXPBYPCZ(y,one,minusone,minusone,lambda,datain->TempJT);
  ISDestroy(&isscat);
  return 0;
}
```

A.3 The femH parallel C++ FEM library

This is a work in progress, with femH we try to create a parallel finite elements library inspired from our Matlab codes, ifem and deal.ii and tailored for overlapping and non-overlapping domain decomposition methods. Some classes of this library are presented during the Finite Element Methods Chapter of the Thesis and have been used for the `PCASM` experiments in Chapter 4.This code can be found at `https://bitbucket.org/modios/femh/`.

Algorithm 19 Assembly of Shell Matrix P, part 1.

```
1  PetscErrorCode Preconcondioner_3(Mat *Pf,Mat M,int count_interface,Mat K,int rank,int size){
2    Mat J,JT,P,TempA,L,LE;
3    Vec z,z1,out,TempJ,TempJT;
4    int n,flag,rowStartJ,rowEndJ,i,cumsum=0;//,rowId,colId;,m,
5    PetscScalar norm,one=1.0,val;//,zero=0.0;
6    Precondata3 *datain;
7    KSP solver;
8    PC pc;
9
10   MatISGetLocalMat(M,&TempA);
11   MatGetSize(TempA,&n,&n);
12   VecCreate(PETSC_COMM_SELF,&z);
13   VecSetSizes(z,n,n);
14   VecSetFromOptions(z);
15   VecSet(z,one);
16   VecDuplicate(z,&out);
17   MatMult(TempA,z,out);
18   //VecView(out,PETSC_VIEWER_STDOUT_SELF);
19   VecNorm(out,NORM_2,&norm);
20   //if(norm<1e-12){ flag=1;
21   if(norm<1e-6){ flag=1;
22   }else{ flag=0;}
23   //printf("%f\n",norm);
24   int *totalcount=malloc(sizeof(int)*size);
25   MPI_Allgather(&count_interface, 1, MPI_INT,totalcount,1,MPI_INT,
26                 PETSC_COMM_WORLD);
27   if(rank>0){
28     for(i=0;i<rank;i++){
29       cumsum=cumsum+totalcount[i];
30     }
31   }
32   //printf("RANK %d %d\n",rank,cumsum);
33   VecDestroy(&z);
34   VecDestroy(&out);
35
36   MatCreate(PETSC_COMM_WORLD,&J);
37   MatSetSizes(J,count_interface,1,PETSC_DETERMINE,PETSC_DETERMINE);
38   MatSetType(J,MATMPIAIJ);
39   MatMPIAIJSetPreallocation(J,1,NULL,1,NULL);
40   MatSetFromOptions(J);
41
42
43   MatCreate(PETSC_COMM_WORLD,&JT);
44   MatSetType(JT,MATMPIAIJ);
45   MatSetSizes(JT,1,count_interface,PETSC_DETERMINE,PETSC_DETERMINE);
46   MatSetUp(JT);
47
48
49   MatGetOwnershipRange(J,&rowStartJ,&rowEndJ);
50
51   if(flag){
52     for(i=rowStartJ;i<rowEndJ;i++){
53       val=1./sqrt(count_interface);
54       MatSetValues(J,1,&i,1,&rank,&val,INSERT_VALUES);
55     }
56   }
57
58   MatAssemblyBegin(J,MAT_FINAL_ASSEMBLY);
59   MatAssemblyEnd(J,MAT_FINAL_ASSEMBLY);
60
61   if(flag){
62     for(i=cumsum;i<cumsum+count_interface;i++){
63       val=1./sqrt(count_interface);
64       MatSetValues(JT,1,&rank,1,&i,&val,INSERT_VALUES);
65     }
66   }
67   MatAssemblyBegin(JT,MAT_FINAL_ASSEMBLY);
68   MatAssemblyEnd(JT,MAT_FINAL_ASSEMBLY);
```

Algorithm 20 Assembly of Shell Matrix P, part 2.

```
1
2   VecCreate(PETSC_COMM_WORLD,&TempJ);
3   VecSetSizes(TempJ,1,PETSC_DECIDE);
4   VecSetFromOptions(TempJ);
5
6   VecDuplicate(TempJ,&z);
7
8   VecCreate(PETSC_COMM_SELF,&z1);
9   VecSetSizes(z1,size,size);
10  VecSetFromOptions(z1);
11
12
13
14  VecCreate(PETSC_COMM_WORLD,&TempJT);
15  VecSetSizes(TempJT,count_interface,PETSC_DECIDE);
16  VecSetFromOptions(TempJT);
17
18
19
20  LExpicit(&LE,J,JT,K,count_interface,size);
21  MatParToSeq(&L,LE,size,rank);
22  MatDestroy(&LE);
23
24
25  KSPCreate(PETSC_COMM_SELF,&solver);
26  KSPSetOperators(solver,L,L,SAME_PRECONDITIONER);
27  KSPSetType(solver, KSPPREONLY);
28  KSPGetPC(solver,&pc);
29  //PCSetType(pc,PCCHOLESKY);
30  PCSetType(pc,PCLU);
31  KSPSetFromOptions(solver);
32  KSPSetUp(solver);
33
34
35
36
37  PetscMalloc(sizeof(Precondata3),&datain);
38  datain->JT=JT;
39  datain->J=J;
40  datain->TempJ=TempJ;
41  datain->TempJT=TempJT;
42  datain->z=z;
43  datain->z1=z1;
44  datain->rank=rank;
45  datain->size=size;
46  datain->solver=solver;
47
48
49  MatCreateShell(PETSC_COMM_WORLD,count_interface,count_interface,PETSC_DETERMINE,\
50                 PETSC_DETERMINE,datain,&P);
51  MatShellSetOperation(P,MATOP_MULT,(void(*)(void))Precon3_func);
52  MatSetFromOptions(P);
53  MatDestroy(&L);
54
55  *Pf=P;
56
57  return 0;
58  }
```

Algorithm 21 Main class for the deal.ii 2 Lagrange multiplier solver

```
template <int dim>
class L2Solver
{
 public:
  L2Solver ();
  ~L2Solver ();
  void run ();
 private:
  void make_grid ();
  void initialise_solution ();
  void renum_mesh();
  void find_GG();
  void setup_system();
  void output_results () const;
  void create_K();
  void create_AN();
  void penalty_AN();
  void Q_LU();
  void Precondtioner();
  void QMINK();
  void IminKQMINK();
  void G_Vector();
  void RHSSymetric();
  void RHSNONSymetric();
  void RecoverSulution();
  void Gather_solution();

  void write_coordinates();
  void write_vector(Vector <double> & v, std::string filename);
  void write_vector (std::vector <bool> &v, std::string filename);
  void write_vector (std::vector <PetscInt> &v, std::string filename);
  void Solver_KSP();

  Mat K;
  Mat AN;
  Mat Q;
  Mat Precond;
  Mat QminK,IminKQminK;
  Vec l;
  Vec local_solution;
  Vec final_solution;

  const PetscScalar minusone=-1.0,one=1.0,zero=0.0;

  Triangulation<dim>   triangulation;
  FE_Q<dim>            fe;
  DoFHandler<dim>      dof_handler;

  Vec      SQrhs,rhs;

  ConstraintMatrix             constraints;


  MPI_Comm mpi_communicator;

  IndexSet               locally_owned_dofs;
  PETScWrappers::MPI::Vector Mmod;
  PETScWrappers::MPI::Vector result;
  KSP KspRobin;
  Vec tempsolution;

  const unsigned int n_mpi_processes;
  const unsigned int this_mpi_process;
  ConditionalOStream pcout;
  TimerOutput      computing_timer;


  std::vector <PetscInt> global_to_local;
  std::vector<PetscInt>  local_indices;
  std::vector<PetscInt>  interface,local_interface,local_to_global_interface;
  std::vector<PetscInt>  local_boundary,local_interior;
  std::vector<PetscScalar> w;
  PetscScalar Robin_par=0.75;
};
```

Appendix B

Proof of Theorem 5.5.22

In this Appendix we provide the proof of Theorem 5.5.22 in a Matrix representation framework.

Lemma B.0.1 (Halmos [35]). *Let E and K be orthogonal projections. There is an orthogonal matrix U which simultaneously block diagonalizes E and K, into 1×1 and 2×2 blocks. If we denote the kth block of the block-diagonalized E by E_k, and the kth block of the block-diagonalized K by K_k, we further have that*

$$E_k \in \left\{0, 1, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right\} \quad \textit{and} \quad K_k \in \left\{0, 1, \begin{bmatrix} c_k^2 & c_k s_k \\ c_k s_k & s_k^2 \end{bmatrix}\right\}, \tag{B.1}$$

where $c_k = \cos(t_k) \neq 0$ and $s_k = \sin(t_k) \neq 0$ with real $t_k \in (0, \pi/2)$ for each k.

The ranges of E and K are hyperspaces, and the angles $\{t_k\}$ are the "principal angles" between these two hyperspaces.

Proof. Assume that E has the form,

$$E = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}$$

and we write the orthogonal projection in a blocks form with X,Z diagonalisable matrices

$$K = \begin{bmatrix} X & Y \\ Y^T & Z \end{bmatrix}.$$

If we apply the singular value decomposition on matrices X, Z we get $X = U\Sigma_1 U^T$, $Y = U^T Y V$ and $Z = V\Sigma_2 V^T$. If we set

$$M = \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix}$$

we get

$$M^T K M = \begin{bmatrix} U^T X U & U^T Y V \\ V^T Y^T U & U^T Z U \end{bmatrix} := \tilde{K}$$

and let $\tilde{X} = U^T X U$ and $\tilde{Z} = U^T Z U$, where both $\tilde{X}$, $\tilde{Z}$ are diagonal matrices. We use that $K^2 = K$, since K is an orthogonal projection and we get

$$\begin{bmatrix} \tilde{X}^2 + \tilde{Y}\tilde{Y}^T & \tilde{X}\tilde{Y} + \tilde{Y}\tilde{Z} \\ \tilde{Y}^T\tilde{X} + \tilde{Z}^T\tilde{Y}^T & \tilde{Z}^2 + \tilde{Y}\tilde{Y}^T \end{bmatrix} = \tilde{K}.$$

To make the notation simpler we drop the tildes for the rest of the proof. Since X and Y are diagonal we get that

$$(XY + YZ)_{ij} = Y_{ij}$$

which leads to

$$X_{ii}Y_{ij} + Y_{ij}Z_{jj} = Y_{ij}.$$

Hence, if $Y_{ij} \neq 0$ then we should have $X_{ii} + Z_{jj} = 1$ else, if $Y_{ij} = 0$, X_{ii}, Z_{jj} are arbitrary. Grouping repeated eigenvalues together leads to,

$$\tilde{K} = \left[\begin{array}{cccc|cccc} x_1 I_1 & & & & Y_{11} & Y_{12} & \cdots & Y_{1n} \\ & x_2 I_2 & & & \vdots & \vdots & \vdots & \vdots \\ & & \ddots & & \vdots & \vdots & \vdots & \vdots \\ & & & x_m I_m & Y_{m1} & Y_{m2} & \cdots & Y_{mn} \\ \hline Y_{11} & Y_{21} & \cdots & Y_{n1} & z_1 I_{m+1} & & & \\ \vdots & \vdots & \vdots & \vdots & & z_2 I_{m+2} & & \\ \vdots & \vdots & \vdots & \vdots & & & \ddots & \\ Y_{1m} & Y_{2m} & \cdots & Y_{nm} & & & & z_n I_n \end{array}\right]$$

By the preceding, we have that $Y_{ij} = 0$ unless x_i,$z_j \neq 0$. Hence we can also block diagonalize the matrix Y. Permuting again the rows and columns of K we get a block diagonal projection matrix with blocks

$$\begin{bmatrix} x_j I_j & Y_j \\ Y_j^T & z_j I_{m+j} \end{bmatrix}. \tag{B.2}$$

If we use singular the value decomposition on matrix Y_j in $(B.2)$ one attains

$$\begin{bmatrix} x_j & \sigma \\ \sigma & z_j \end{bmatrix}. \tag{B.3}$$

In order to get a better insight of 2×2 projections we categorise them in the following way.

1. Rank 0 projection: the only choice here is the Zero matrix.

2. Rank 2 projection: the only choice is the identity matrix I, with $\lambda_1 = \lambda_2 = 1$.

3. Rank 1 projection: Here we can either have $\lambda = 0$ or $\lambda = 1$. Since the projection is symmetric, each block of K can be written as $K_k = Q \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} Q^T$, where Q is orthogonal
$$\begin{bmatrix} c \\ s \end{bmatrix}$$
with $c^2 + s^2 = 1$, is the left column of Q, then
$$Q = \begin{bmatrix} c & * \\ s & * \end{bmatrix}.$$
Finally we get that
$$K_k = \begin{bmatrix} c^2 & cs \\ cs & s^2 \end{bmatrix}.$$

□

Bibliography

[1] Robert A Adams and John JF Fournier. *Sobolev Spaces*. Academic Press, 2003.

[2] Valeri I Agoshkov. Poincaré-Steklov operators and domain decomposition methods in finite dimensional spaces. In *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 73–112. SIAM Philadelphia, 1988.

[3] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951.

[4] Kendall Atkinson and Weimin Han. *Theoretical Numerical Analysis*. Springer, 2005.

[5] Ivo Babuska. The finite element method with Lagrangian multipliers. *Numerische Mathematik*, 20:179–192, 1972/73.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2014.

[7] Wolfgang Bangerth, Timo Heister, Luca Heltai, Guido Kanschat, Martin Kronbichler, Matthias Maier, Bruno Turcksin, and Toby D Young. The deal. ii library, version 8.1. *arXiv preprint arXiv:1312.2266*, 2013.

[8] Brad J C Baxter and Arieh Iserles. On the foundations of computational mathematics. *University of Cambridge department of applied mathematics and theoretical physics-report*, 2002.

[9] Michele Benzi, Gene H Golub, and Jorg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14(1):1–137, 2005.

[10] Thierry Braconnier and Nicholas J Higham. Computing the field of values and pseudospectra using the lanczos method with continuation. *BIT Numerical Mathematics*, 36(3):422–440, 1996.

[11] Susanne C Brenner and Carsten Carstensen. Finite element methods. *Encyclopedia of Computational Mechanics*, 2004.

[12] Susanne C Brenner and Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2008.

[13] Susanne C Brenner and Li-Yeng Sung. BDDC and FETI-DP without matrices or vectors. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1429–1435, 2007.

[14] Haim Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer, 2011.

[15] Arthur Cayley. A memoir on the theory of matrices. *Philosophical Transactions of the Royal Society of London*, 148:17–37, 1858.

[16] Tony F Chan and Henk A Van Der Vorst. Approximate and incomplete factorizations. *Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engeneering*, pages 167–202, 1997.

[17] Long Chen. Programming of finite element methods in matlab. *Preprint, University of California Irvine, http://math.uci.edu/˜ chenlong/226/Ch3FEMCode.pdf*, 2011.

[18] Zhangxin Chen. *Finite Element Methods and their Applications*. Springer, 2005.

[19] Ernst F Chladni. *Entdeckungen über die Theorie des Klanges*. Zentralantiquariat der DDR, 1787.

[20] Wolfgang Dahmen, Birgit Faermann, Ivan Graham, Wolfgang Hackbusch, and Stefan Sauter. Inverse inequalities on non-quasi-uniform meshes and application to the mortar element method. *Mathematics of Computation*, 73(247):1107–1138, 2004.

[21] Jack Dongarra and Francis Sullivan. Guest editors introduction: the top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.

[22] Tobin A Driscoll, Kim-Chuan Toh, and Lloyd N Trefethen. From potential theory to matrix iterations in six steps. *SIAM Review*, 40(3):547–578, 1998.

[23] Maksymilian Dryja and Olof Widlund. An additive variant of the Schwarz alternating method for the case of many subregions. Technical report, Technical Report, Courant Institute, 1987.

[24] Olivier Dubois, Martin J Gander, Sébastien Loisel, Amik St-Cyr, and Daniel B Szyld. The Optimized Schwarz method with a coarse grid correction. *SIAM Journal on Scientific Computing*, 34(1):A421–A458, 2012.

[25] Stanley C Eisenstat, Howard C Elman, and Martin H Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM Journal on Numerical Analysis*, 20(2):345–357, 1983.

[26] Charbel Farhat, Antonini Macedo, Michel Lesoinne, Francois-Xavier Roux, Frédéric Magoulés, and Armel de La Bourdonnaie. Two-level domain decomposition methods with Lagrange multipliers for the fast iterative solution of acoustic scattering problems. *Computer Methods in Applied Mechanics and Engineering*, 184(2):213–239, 2000.

[27] David A Field. Qualitative measures for initial meshes. *International Journal for Numerical Methods in Engineering*, 47(4):887–906, 2000.

[28] Joseph Fourier. Mémoire sur la propagation de la chaleur dans les corps solides. *Nouveau Bulletin des Sciences de la Société Philomathique de Paris*, 6:112–116, 1808.

[29] Martin J Gander. Optimized Schwarz methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, 2006.

[30] Martin J Gander. Schwarz methods over the course of time. *Electronic Transactions on Numerical Analysis*, 31(228-255):5, 2008.

[31] Martin J Gander and Felix Kwok. Best Robin parameters for optimized Schwarz methods at cross points. *SIAM Journal on Scientific Computing*, 34(4):A1849–A1879, 2012.

[32] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[33] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[34] Neil Greer and Sébastien Loisel. The optimised Schwarz method and the two-Lagrange multiplier method for heterogeneous problems in general domains with two general subdomains. *Numerical Algorithms*, pages 1–26, 2014.

[35] P. R. Halmos. Two subspaces. *Trans. Amer. Math. Soc.*, 144:381–389, 1969.

[36] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012.

[37] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1952.

[38] R. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[39] Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906.

[40] Anastasios Karangelis and Sébastien Loisel. Condition number estimates and weak scaling for 2-level 2-Lagrange multiplier methods for general domains and cross points. *SIAM Journal on Scientific Computing*, 37(2):C247–C267, 2015.

[41] Anastasios Karangelis, Sébastien Loisel, and Chris Maynard. Solving large systems on hector using the 2-Lagrange multiplier methods. In Jocelyne Erhel, Martin J. Gander, Laurence Halpern, Graldine Pichot, Taoufik Sassi, and Olof Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXI*, volume 98 of *Lecture Notes in Computational Science and Engineering*, pages 497–505. Springer International Publishing, 2014.

[42] George Karypis and Vipin Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.376&rep=rep1&type=pdf`, 1995.

[43] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[44] Jung-Han Kimn. A convergence theory for an overlapping Schwarz algorithm using discontinuous iterates. *Numerische Mathematik*, 100(1):117–139, 2005.

[45] Mats G Larson and Fredrik Bengzon. *The Finite Element Method: Theory, Implementation, and Applications: Theory, Implementation, and Applications*. Springer, 2013.

[46] Jörg Liesen and Zdenek Strakos. *Krylov Subspace Methods: Principles and Analysis*. Oxford University Press, 2012.

[47] Pierre-Louis Lions. On the Schwarz alternating method. iii: a variant for nonoverlapping subdomains. In *Third international symposium on domain decomposition methods for partial differential equations*, volume 6, pages 202–223. SIAM, Philadelphia, PA, 1990.

[48] Sébastien Loisel. Condition number estimates for the nonoverlapping optimized Schwarz method and the 2-Lagrange multiplier method for general domains and cross points. *SIAM Journal on Numerical Analysis*, 51(6):3062–3083, 2013.

[49] Sébastien Loisel and Daniel B Szyld. On the geometric convergence of optimized Schwarz methods with applications to elliptic problems. *Numerische Mathematik*, 114(4):697–728, 2010.

[50] William Charles Hector McLean. *Strongly Elliptic Systems and Boundary Integral Equations*. Cambridge University Press, 2000.

[51] Gordon E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[52] Peter Pacheco. *An Introduction to Parallel Programming*. Elsevier, 2011.

[53] Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[54] Per-Olof Persson. *Mesh Generation for Implicit Geometries, PhD thesis*. 2005.

[55] G Ciarlet Philippe. The Finite Element Method for Elliptic Problems, 1978.

[56] W Ritz. Über eine neue methode zur lösung gewisser randwertaufgaben. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1908:236–248, 1908.

[57] Torgeir Rusten and Ragnar Winther. A preconditioned iterative method for saddlepoint problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):887–904, 1992.

[58] Youcef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

[59] Youcef Saad and Martin H Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[60] Yousef Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37(155):105–126, 1981.

[61] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[62] Hermann Amandus Schwarz. *Ueber einen Grenzübergang durch alternirendes Verfahren*. Zürcher u. Furrer, 1870.

[63] Jonathan R Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222. Springer, 1996.

[64] David Silvester and Andrew Wathen. Fast iterative solution of stabilised Stokes systems part ii: using general block preconditioners. *SIAM Journal on Numerical Analysis*, 31(5):1352–1367, 1994.

[65] Barry Smith, Petter Bjorstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[66] John Miles Smith and Philip Yen-Tang Chang. Optimizing the performance of a relational algebra database interface. *Communications of the ACM*, 18(10):568–579, 1975.

[67] Amik St-Cyr, Martin J Gander, and Stephen J Thomas. Optimized multiplicative, additive, and restricted additive Schwarz preconditioning. *SIAM Journal on Scientific Computing*, 29(6):2402–2425, 2007.

[68] James William Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*. Springer Science & Business Media, 1995.

[69] Andrea Toselli and Olof Widlund. *Domain Decomposition Methods: Algorithms and Theory*. Springer, 2005.

[70] Lloyd N Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000.

[71] Lloyd N Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.

[72] Henk A Van der Vorst. Parallel iterative solution methods for linear systems arising from discretized PDE's. *Special Course on Parallel Computing in CFD, available from www.staff.science.uu.nl/ vorst102/agard.ps.gz*, 1995.

[73] Henk A Van der Vorst and Cees Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1(4):369–386, 1994.

[74] Andrew Zisserman and Richard Hartley. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.