UNIVERSITÀ DEGLI STUDI DI MILANO

Scuola di Dottorato in
Matematica e Statistica per le Scienze Computazionali
Ciclo XXVIII

Dipartimento di Matematica
"F. Enriques"

Parallel Additive Schwarz Preconditioning for Isogeometric Analysis

Federico Marini

TUTOR: Prof. Luca Franco Pavarino
COORDINATORE: Prof. Giovanni Naldi

Anno Accademico
2014/2015

Contents

1 Introduction **4**

2 B-spline and NURBS **8**
2.1 Univariate B-splines 8
2.2 Tensor product and geometries 10
2.3 Refinements 13
2.3.1 h-refinement 14
2.3.2 p-refinement 16
2.3.3 k-refinement 18
2.4 NURBS 19
2.5 Basis functions in physical domain 20
2.6 Approximation estimates 23
2.6.1 Approximation in parametric domain 25
2.6.2 Approximation in physical domain 26
2.6.3 Approximation for advection-diffusion equations 27

3 Additive Schwarz Preconditioning **30**
3.1 Abstract theory 30
3.2 Decomposition strategies 33
3.2.1 Strategy 1 34
3.2.2 Strategy 2 35
3.2.3 Strategy 3 36
3.3 Isogeometric Decomposition setting 37
3.4 Stable splitting 40
3.4.1 Univariate stable splitting 41
3.4.2 Multivariate stable splitting 46
3.4.3 Stable splitting in physical domain 49
3.5 On the advection-diffusion equation 51

4 Numerical Results **53**
4.1 The scalar elliptic equation 58
4.1.1 2D results: the square 60
4.1.2 2D results: the quarter annulus 66
4.1.3 3D results: the cube 72
4.1.4 3D results: the quarter hose 74
4.2 The advection-diffusion equation 80
4.2.1 2D results: the square 81
4.2.2 3D results: the cube 86
4.2.3 Preliminary results on deformed NURBS domains 89

5 Implementation **91**
5.1 The main program 91
5.2 The projection operator 98
5.3 The shell preconditioner 102
5.4 On the computational times 106

6 Conclusions **110**

Bibliography **112**

Chapter 1

Introduction

Isogeometric Analysis (IGA) was first introduced in 2005 by T. Hughes, Y. Bazilevs *et. al.* in [25] as a first attempt to merge Computer Aided Design (CAD) and Computer Aided Engineering (CAE), and then further explored in [8].
IGA shares many aspects with Finite Element Method (FEM) in the numerical solution of partial differential equations, but has a different way of representing domains and discrete function spaces.
The IGA framework is based on B-spline ([31]) and non-uniform rational B-splines (NURBS) basis functions ([28]). B-splines are piecewise polynomial functions of assigned degree p, and they can enjoy high regularity up to $\mathcal{C}^{p-1}$, a distinctive feature for IGA. B-splines have compact support, form a partition of unity and have optimal approximation properties ([7]), making them a suitable choice as basis functions for Petrov-Galerkin discretizations.
NURBS basis functions are weighted B-spline functions, i.e. they are rational piecewise polynomials and share all the properties of B-splines. They are a *de facto* industrial standard in CAD for representing common curved geometries, such as conics, spheres or cylinders. Geometries in CAD (and now with IGA, PDE domains) are represented as image of a *parametric domain* $[0,1]^D$ via a map F obtained as linear combination of NURBS basis functions. Geometry representation is exact and invariant under any type of mesh refinement: knot insertion (h-refinement), degree elevation (p-refinement) and k-refinement, a third type of refinement that has no equivalent in FEM. In k-refinement, knot insertion and degree elevation are performed at the same time in order to decrease the mesh size and increase the regularity of the basis functions.
In classic FEM, geometries are typically represented with polygonal meshes, with consequent approximation for curved boundaries representation, and implementation of high regularity basis functions is cumbersome. The FEM, on the other hand, has been extensively researched and applied for decades, and many techniques have been developed in order to solve any flavor of engineering problems.

Among these techniques there are the Domain Decomposition Methods, introduced and developed for parallel computing in [32] and [33]. The theory and implementation is well-known for the FEM case, but it is still in development for IGA. Recent works on IGA and Overlapping Domain Decomposition are [9], [11], [12], [14], and [23], among with application-oriented works [18], [19]. Non-overlapping techniques have been developed in an IGA framework in [10], [13], [17], [26], while multigrid solvers have been explored in [21] and [22]. Some studies of iterative solvers performance in IGA and Domain Decomposition in general can be found respectively in [20] and [4].
The main focus of this thesis is the construction of a multi-level Domain Decomposition preconditioner of Overlapping Additive Schwarz type that can be effectively used in a massively parallel environment to solve linear systems arising from scalar elliptic and advection-diffusion PDEs.
In order to show its scalability, two model problems are considered. The first model problem is a scalar elliptic equation:

$$\begin{cases} -\nabla \cdot (k(x)\nabla u) = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

It will be shown that the diffusivity coefficient k does not affect the preconditioner scalability with respect to the number of processors involved in the computation, even in case of highly discontinuous jumps between subdomains.
The second model problem is the advection-diffusion equation:

$$\begin{cases} -\nabla \cdot (k(x)\nabla u) + \mathbf{b} \cdot \nabla u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

Although (and to our knowledge) there is no theory in support of the preconditioner for the advection-diffusion problem, numerical results are shown with a strong emphasis on the advection-dominated case, in which the advective part of the equation dominates the diffusivity part. This is a well-known problem that brings some of the difficulties that can be found in the Navier-Stokes equations. The performance of the preconditioner deteriorates with the advection domination, but it can still be recovered with SUPG stabilization.
We are now ready to introduce the preconditioner in its two different forms. As Domain Decomposition suggests, the domain Ω of a PDE is decomposed in several overlapping subdomains. Associated with each subdomain there is a local basis function space:

$$\Omega = \bigcup_{j=1}^{N} \Omega_j, \qquad \Omega \longleftrightarrow \mathcal{V} = \text{span}\left\{N_i^p,\ i = 1, ..., n\right\}, \qquad \Omega_j \longleftrightarrow \mathcal{V}_j \subset \mathcal{V}.$$

where N_i^p are NURBS basis functions. There are at least three strategies for constructing $\mathcal{V}_j$, but they can be generally thought as spaces in which basis functions have support in Ω_j.
From the Petrov-Galerkin discretization of the two model problems previously introduced, we obtain a linear system $Au = f$, where

$$A = [a(N_\ell^p, N_\kappa^p)], \qquad \forall N_\ell^p, N_\kappa^p \in \mathcal{V},$$

with $a(\cdot, \cdot)$ being the bilinear form coming from the variational forms of the PDEs. A local system matrix is associated to each subdomain:

$$\Omega_j \longleftrightarrow \mathcal{V}_j \longleftrightarrow A_j = [a(N_\ell^p, N_\kappa^p)], \quad \forall N_\ell^p, N_\kappa^p \in \mathcal{V}_j.$$

If we denote with $R_j : \mathcal{V} \to \mathcal{V}_j$ the projection operator that restricts a vector of $\mathcal{V}$ into a vector of $\mathcal{V}_j$, with R_j^T being the extension-to-zero transposed operator, then the 1-level Additive Schwarz Preconditioner is defined as:

$$\left(A_{ASO}^{(1)}\right)^{-1} = \sum_{j=1}^{N} R_j^T A_j^{-1} R_j,$$

This preconditioner can be constructed with no knowledge of the underlying PDE structure, i.e. it can be constructed by algebraic means in a black-box fashion, although theory and numerical results show that it does not scale with the number of domain (or processors) involved in the decomposition. The higher the number of subdomains, the higher the condition number of the preconditioned system matrix, the higher the number of iterations required by the iterative solver in order to compute a discrete solution.
The main reason of this behavior is that the information stored in the local subsystems A_j requires many iteration to flow from one subdomain to the ones that are not directly adjacent. In other words, the 1-level OAS preconditioner does not have a global view of the linear system.
This motivates the introduction of a global coarse problem to be added (literally) to the local problems. The mesh arising from the domain decomposition can be used as a mesh for the coarse problem, so that in each iteration each processor has to solve both a fine local problem for detailed analysis and the coarse global problem for the global analysis.
The construction is the following: $\mathcal{V}_0 \subset \mathcal{V}$ is the coarse space, and it is a proper subspace of $\mathcal{V}$. The mesh used for $\mathcal{V}_0$ has the subdomains as elements. We denote with $R_0 : \mathcal{V} \to \mathcal{V}_0$ the fine-to-coarse projection operator, while its transposed R_0^T is the coarse-to-fine operator. Due to the properties of B-spline and NURBS, and the fact that h-refinement generates collections of nested vector spaces, the

operator R_0^T is exact, i.e. there is no approximation in extending a vector from $\mathcal{V}_0$ to $\mathcal{V}$. The 2-level Additive Schwarz Preconditioner is

$$\left(A_{ASO}^{(2)}\right)^{-1} = R_0^T A_0^{-1} R_0 + \sum_{j=1}^{N} R_j^T A_j^{-1} R_j,$$

with A_0 being the system matrix associated with the coarse space V_0.
It will be proved that the 2-level preconditioner is scalable with the number of subdomains. If we denote with h the fine mesh size and H the coarse mesh size, then for the linear system arising from the scalar elliptic equation there exists a constant $C > 0$, independent on h and H and the number of subdomains N, such that the following bound for the condition number holds:

$$\text{cond}\left(\left(A_{ASO}^{(2)}\right)^{-1} A\right) \leq C\left(1 + \frac{H}{h}\right).$$

Numerical results will be shown as a confirmation of the theory. Both 2D and 3D tests are taken into account, and in both cases the model problem is defined over simple geometries (squares and cubes) with B-spline discretizations and complex curved geometries with a full NURBS discretization.
The structure of this work is the following: in Chapter 2 we give formal definitions of B-spline and NURBS, their usage for geometry representation and as basis function for discrete functional spaces, explanations on the three different types of refinement and finally a brief overview of their approximation properties.
Chapter 3 is firstly devoted to the abstract setting of overlapping Domain Decomposition and their theoretical requirements. We then present the steps required to plug Isogeometric Analysis into the abstract setting. Three decomposition strategies are proposed and one of them is chosen for both theoretical and implementation settings.
Chapter 4 presents the numerical results on the performance of the 2-level Additive Schwarz Preconditioner. First are presented the scalar elliptic equations tests with different diffusivity coefficients, for both 2 and 3 dimensions. The advection diffusion equations results will follow, with an emphasis on the preconditioner behavior with respect to advection-dominated problems.
Finally, Chapter 5 presents an overview the main program of the code used for this work and the two core functions: `Projection` and `ShellPCApply`. The former is the parallel implementaion of the construction of the coarse-to-fine projection operator R_0^T, while the latter is the implementation of its action during a preconditioned solver iteration. Most of the code, such as the implementation of the grid construction and the element loop-based matrix assemble, is omitted for brevity, as it comprise thousands of line of code. The analysis of the execution times will conclude the Chapter befor the Conclusions.

Chapter 2

B-spline and NURBS

In this Chapter we sketch the main mathematical object at the foundation of Isogeometric Analysis. Starting with the definition of B-spline basis functions, we then explain their extension to the multivariate case and how to use them for mesh representation with different types of refinement. The presentation of NURBS basis functions will follow, along with their usage as basis functions in physical domain for Petrov-Galerkin discretizations. We refer to [8] for deeper analysis. Finally, we will present some theoretical results that ensure the good approximation properties of B-spline and NURBS, for which we refer to [7].

2.1 Univariate B-splines

Definition 2.1. Let $p \in \mathbb{N}$ be the polynomial degree[1] and $n \in \mathbb{N}$. We define *knot vector* the ordered set of real numbers

$$\Xi = \{0 = \xi_1 \leq \xi_2 \leq ... \leq \xi_{n+p} \leq \xi_{n+p+1} = 1\}. \tag{2.1}$$

The elements of the sets ξ_i are called *knots*. Repetitions of the knots in the knot vector are allowed. We denote with $m_i = m(\xi_i)$ the multiplicity of the knot ξ_i. A knot vector is said to be *open* if

$$\xi_1 = ... = \xi_{p+1} = 0, \qquad \xi_{n+1} = ... = \xi_{n+p+1} = 1,$$

while it is said to be *uniform* if knots are equally spaced.

Definition 2.2. Given a knot vector $\Xi = \{\xi_1 \leq ... \leq \xi_{n+p+1}\}$ we call *univariate B-spline basis functions* (for brevity *B-splines*) the n functions $\hat{B}_i^p : [0,1] \to \mathbb{R}$

[1] In CAD literature it is more common to refer to the polynomial degree with the term *order* m, for which it holds $m = p + 1$.

recursively defined with the Cox-De Boor formula:

$$\begin{aligned} \hat{B}_i^0(\xi) &= \begin{cases} 1 & \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ \hat{B}_i^p(\xi) &= \frac{\xi - \xi_i}{\Delta_i^p} \hat{B}_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\Delta_{i+1}^p} \hat{B}_{i+1}^{p-1}(\xi), \end{aligned} \qquad i = 1, ..., n, \tag{2.2}$$

where $\Delta_i^p = \xi_{i+p+1} - \xi_i$ with the convention of considering zero every fraction for which $\Delta_i^p = 0$. This is well-defined because $\Delta_i^p = 0$ if and only if $\hat{B}_i^{p-1}(\xi)$ is zero. We denote the linear space of B-spline functions with the following notation:

$$\hat{\mathcal{B}} = \hat{\mathcal{B}}(\Xi, p) = \text{span}\left\{ \hat{B}_i^p,\ i = 1, ..., n \right\}. \tag{2.3}$$

We define *parametric element* an interval $q = [\xi_i, \xi_{i+1}]$ with positive measure and *parametric mesh* the set of elements $\mathcal{Q} = \{Q\}$. Finally, we define *parametric element size* the number $\hat{h}_Q = \text{diam}(Q)$ and *parametric mesh size* $\hat{h} = \max_{Q \in \mathcal{Q}} \hat{h}_Q$.

B-spline basis functions are piecewise polynomial functions and have properties that make them a suitable choice as basis functions for a Petrov-Galerkin discretization:

- **Local support**. The support of the B-spline $\hat{B}_i^p$ is $[\xi_i, \xi_{i+p+1}]$. The measure of the support depends on the multiplicity of the knots $\xi_i, ..., \xi_{i+p}$. The support is reduced whenever there is a knot in $[\xi_i, \xi_{i+p+1}]$ with a multiplicity greater than 1.

- **Regularity**. B-splines are $\mathcal{C}^{p-m_i}$ in the knot ξ_i, with $\mathcal{C}^{p-1}$ the highest regularity, and $\mathcal{C}^\infty$ elsewhere. Multiplicity $m_i = p$ leads to a single nonzero basis function on the knot ξ_i, belonging to $\mathcal{C}^0$ and interpolatory, i.e. $\hat{B}(\xi_i) = 1$. Multiplicity $m_i = p + 1$ is allowed and leads to discontinuous and interpolatory basis functions. This motivates the use of open knot vectors for representing geometry boundaries and boundary conditions.

- **Partition of unity**. B-splines form a partition of unity:

$$\sum_{i=1}^{n} \hat{B}_i^p(\xi) = 1 \quad \forall \xi \in [0, 1].$$

- **Derivatives**. B-spline derivatives can be easily computed with lower degree B-splines:

$$\frac{\partial}{\partial \xi} \hat{B}_i^p(\xi) = p \left(\frac{\hat{B}_i^{p-1}(\xi)}{\Delta_i^p} - \frac{\hat{B}_{i+1}^{p-1}(\xi)}{\Delta_{i+1}^p} \right). \tag{2.4}$$

  As usual, a denominator Δ_i^p can be zero if and only if the B-spline at the numerator is zero, hence the fraction has to be considered null.

- **Dual basis**. There exists a set of linear functionals $\{\lambda_i^p : \hat{\mathcal{B}}(\Xi, p) \longrightarrow \mathbb{R}\}_{i=1}^n$ defined as a local integral:

$$\lambda_i^p[f] = \int_{\xi_i}^{\xi_{i+p+1}} f(\xi)\Lambda(\xi)\, d\xi, \quad i = 1, ..., n, \tag{2.5}$$

where Λ is a particular B-spline function. We refer to [31] for a deeper analysis. The functionals constitute a dual basis for $\hat{\mathcal{B}}(\Xi, p)$, i.e. $\lambda_i^p[\hat{B}_j^p] = \delta_{ij}$. Moreover, for $q \in [1, +\infty]$ and $f \in L^q(\text{supp}\hat{B}_i^p)$ it holds:

$$|\lambda_i^p[f]| \leq (2p+3)9^p(\Delta_i^p)^{-\frac{1}{q}}\|f\|_{L^q(\text{supp}\hat{B}_i^p)}. \tag{2.6}$$

Figure 2.1 shows the B-spline basis functions of degree $p = 2$ defined over the open knot vector

$$\Xi = \left\{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1, 1\right\},$$

so that it represents a mesh of four elements:

$$\hat{\mathcal{Q}} = \left\{\left[0, \frac{1}{4}\right], \left[\frac{1}{4}, \frac{1}{2}\right], \left[\frac{1}{2}, \frac{3}{4}\right], \left[\frac{3}{4}, 1\right]\right\}.$$

The first and last $p + 1 = 3$ knots are repeated, hence the first basis function (depicted in blue) and the last one (depicted in gray) are interpolatory and have a minimum support of one parametric element. Boundary conditions are easily assigned to those functions. The multiplicity of the knots $\xi_3 = \xi_4 = 1/4$ is 2 so that basis functions in $\xi = 1/4$ have regularity $\mathcal{C}^{p-m_3} = \mathcal{C}^0$. In any other knot, regularity is $\mathcal{C}^{p-1} = \mathcal{C}^1$.

2.2 Tensor product and geometries

Tensor product is the main tool for the construction of multivariate discrete function spaces. Due to the strong structure, tensor product spaces are easy to implement and can be computed efficiently. On the other side they don't allow for adaptive techniques such as local mesh refinement.
Because of the fact that tensor product structures bring heavy notations, we will introduce a compact notation for multiindeces:

$$\mathbf{i} \longleftrightarrow (i_1, ..., i_D) \in \mathbb{N}^D.$$

The index i_d ranges from 1 to n_d, while $\mathbf{i}$ can be interpreted as the full multidimensional index or as an integer ranging from 1 to $\mathbf{n} = \prod_{d=1}^D n_d$. The code written

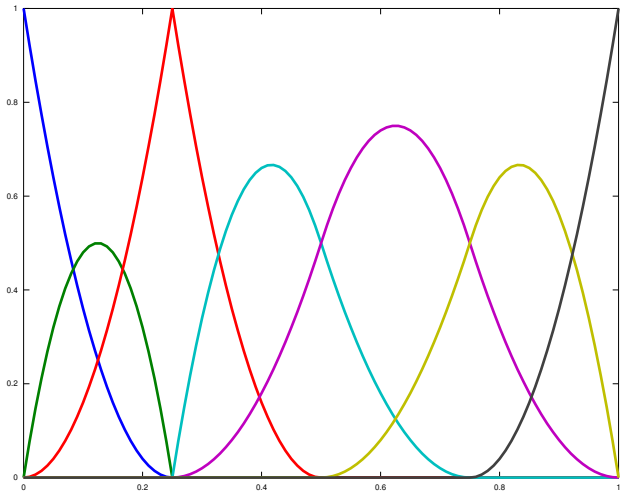


Figure 2.1: B-spline basis functions.

for this thesis is implemented with the following bijection from the multiindex $(i_1, ..., i_D)$ to the index $\mathbf{i}$:

$$\mathbf{i} = \sum_{d=1}^{D} P_d i_d, \qquad i_d = \lfloor \mathbf{i}/P_d \rfloor \mathrm{n}_d, \quad d = 1, ..., D. \tag{2.7}$$

where

$$P_d = \begin{cases} 1 & d = 1, \\ \prod_{d'=1}^{d} n_{d'} & d > 1. \end{cases}$$

The bijection respects the counting order from the first dimension to the last one. In three dimensions, for example, we have:

$$\begin{aligned} &(0,0,0), (1,0,0), ..., (n_1 - 1, 0, 0), \\ &(0,1,0), (1,1,0), ..., (n_1 - 1, 1, 0), \\ &(0,2,0), ... \\ &... \\ &..., (n_1 - 1, n_2 - 1, n_3 - 2), (n_1 - 1, n_2 - 1, n_3 - 1). \end{aligned}$$

Note that, in pure C programming language style, n indexes range from 0 to $n-1$. For theoretical studies in the following pages we will also adopt the more reader-friendly 1-based index convention: $i = 1, ..., n$. Due to the bijection, there is

no ambiguity from $\mathbf{i}$ to $(i_1, ..., i_D)$ and vice-versa. For consistency, index and multiindex with the same letter are considered the same index, i.e. $(i_1, ..., i_D)$ corresponds only to $\mathbf{i}$ and not $\mathbf{j}$.
The bold notation will be extended to non-ranging entities such as polynomial degrees or sets of knot vectors:

$$(p_1, ..., p_d) = \mathbf{p}, \qquad (\Xi_1, ..., \Xi_d) = \boldsymbol{\Xi}.$$

Summation over multiindeces are intended as multiple summations over the associated indexes:

$$\sum_{\mathbf{i}=1}^{\mathbf{n}} = \sum_{i_1=1}^{n_1} ... \sum_{i_D=1}^{n_D}.$$

Definition 2.3. Let $D \in \mathbb{N}$ be an integer and $\boldsymbol{\Xi} = (\Xi_1, ..., \Xi_D)$ knot vectors with

$$\Xi_d = \left\{0 = \xi_1^d \leq \xi_1^d \leq ... \leq \xi_{n_d+p_d}^d \leq \xi_{n_d+p_d+1}^d = 1\right\}, \qquad d = 1, ..., D,$$

with $\mathbf{p} = (p_1, ..., p_D)$ being the polynomial degrees. We define *multivariate B-spline basis functions* (or again for brevity *B-splines*) the functions

$$\hat{B}_{\mathbf{i}}^{\mathbf{p}} = \hat{B}_{i_1,...,i_D}^{p_1,...,p_D} : [0,1]^D \longrightarrow \mathbb{R},$$

$$\hat{B}_{\mathbf{i}}^{\mathbf{p}}(\xi) = \hat{B}_{i_1,...,i_D}^{p_1,...,p_D}(\xi_1, ..., \xi_D) = \prod_{d=1}^{D} \hat{B}_{i_d}^{p_d}(\xi_d). \tag{2.8}$$

The B-spline function space is denoted with

$$\begin{aligned} \hat{\mathcal{B}} &= \hat{\mathcal{B}}(\Xi_1, ..., \Xi_D, p_1, ..., p_D) = \hat{\mathcal{B}}(\boldsymbol{\Xi}, \mathbf{p}) = \\ &= \bigotimes_{d=1}^{D} \hat{\mathcal{B}}(\Xi_d, p_d) = \operatorname{span}\left\{\hat{B}_{\mathbf{i}}^{\mathbf{p}},\ \mathbf{i} = 1, ..., \mathbf{n}\right\}. \end{aligned} \tag{2.9}$$

The notion of elements $Q = \bigotimes_{d=1}^{D} [\xi_{i_d}^d, \xi_{i_d+1}^d]$, mesh $\mathcal{Q}$, element sizes and mesh size are extended to the multivariate case.

The properties listed in Section 2.1 are trivially valid for multivariate spaces as well.
With multivariate basis functions it is possible to construct multidimensional geometries such as surfaces or volumes. This geometry representation is one of the fundamental aspect of IGA: domains of a PDE are parametrically described with B-spline or NURBS geometric functions.

Definition 2.4. Given dimensions $D, D' \in \mathbb{N}$, a set of knot vectors $\boldsymbol{\Xi}$ with polynomial degrees $\mathbf{p}$ and a set of control points $\{C_{i_1,\dots,i_{D'}}\}_{i_1,\dots,i_{D'}=1}^{n_1,\dots,n_{D'}} = \{C_{\mathbf{i}}\}_{\mathbf{i}=1}^{\mathbf{n}} \subset \mathbb{R}^{D'}$, we define *B-spline geometry* the function

$$\mathbf{F} : [0,1]^D \to \mathbb{R}^{D'}, \qquad \mathbf{F}(\xi) = \mathbf{F}(\xi_1, ..., \xi_D) = \sum_{\mathbf{i}=1}^{\mathbf{n}} C_{\mathbf{i}} \hat{B}_{\mathbf{i}}^{\mathbf{p}}(\xi_1, ...\xi_D). \tag{2.10}$$

In case of $D = 1, 2, 3$ the geometry can be also referred respectively as *curve*, *surface* or *volume*.

The key concept is that a domain Ω of a PDE is represented as image of the parametric domain $[0,1]^D$ via a geometric map $\mathbf{F}$:

$$\Omega = \mathbf{F}\left([0,1]^D\right).$$

With Finite Element analysis in mind, many aspects have to be considered in order to obtain a good geometrical description of Ω: shape regularity, non-zero Jacobians, mesh topology and many others. The construction of a high quality geometry can be trivial for a CAD user and cumbersome for a FEM one, especially in the 3D case. The generation of complex and analysis-suitable geometries are beyond the scope of this thesis. In this work all treated geometries are simply connected, shape-regular and embedded in $\mathbb{R}^{D'}$ with $D' = D$.

2.3 Refinements

A knot vector, and subsequently a B-spline space, can be refined in three different ways: the h-refinement, in which new knots are added in the knot vector; the p-refinement, in which the polynomial degree is elevated; the k-refinement, which is a combination of h-refinement and k-refinement that allows an increase in both element number (and thus a decrease of mesh size) and basis functions regularity. This particular refinement has no equivalent in FEM.
It has to be noticed that B-spline spaces depend only on the knot vectors, which means that in order to generate a richer B-spline space (in terms of both mesh size and degree) it suffices to define new knot vectors. The refining procedures work only on a coefficient level, which is not needed with B-spline but it is required for NURBS, as we will see in Section 2.4.
Any type of refinement that can be performed with B-splines (and NURBS) does not change the shape and the parametrization of a geometry. The refinement process is performed dimension-wise, hence it will be shown in details for univariate spaces only.

2.3.1 h-refinement

The h-refinement, or *knot insertion* in CAD literature, is used to refine the knot vector in order to obtain a finer one. h-refinement is the IGA equivalent of the FEM mesh refinement.
Given a knot vector Ξ, let $\bar{\xi} \in [\xi_k, \xi_{k+1}]$ be a new knot for which we construct $\bar{\Xi} = \Xi \cup \{\bar{\xi}\}$. It can be easily shown that $\hat{\mathcal{B}}(\Xi, p) \subset \hat{\mathcal{B}}(\bar{\Xi}, p)$ with $\dim \hat{\mathcal{B}}(\bar{\Xi}, p) = \dim \hat{\mathcal{B}}(\Xi, p) + 1$, so that a function $v \in \hat{\mathcal{B}}(\Xi, p)$ belongs also to $\hat{\mathcal{B}}(\bar{\Xi}, p)$ and has a representation in terms of B-splines in both spaces:

$$\hat{\mathcal{B}}(\Xi, p) \ni \sum_{i=1}^{n} v_i B_i^p(\xi) = v(\xi) = \sum_{i=1}^{n+1} \bar{v}_i \bar{B}_i^p \in \hat{\mathcal{B}}(\bar{\Xi}, p).$$

The following algorithms computes the new Fourier coefficient $\{\bar{v}_i\}_{i=1}^{n+1}$ from the Fourier coefficient $\{v_i\}_{i=1}^{n}$:

$$\bar{v}_i = \theta_i v_i + (1 - \theta_i) v_{i-1}(\xi), \qquad \theta_i = \begin{cases} 1 & 1 \leq i \leq k-p, \\ \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k-p+1 \leq i \leq k, \\ 0 & k+1 \leq i \leq n+p+2. \end{cases} \tag{2.11}$$

We first notice that h-refinement is a local procedure: the insertion of $\bar{\xi} \in [\xi_k, \xi_{k+1}]$ affects coefficients $i = k-p+1, ..., k$, the other coefficients remaining unchanged.
We then notice that the new knot can be equal to knots that already exist. In this particular case, the multiplicity of the knot is augmented and the regularity of the basis functions whose support includes the new knot is reduced. Geometrical regularity and the parametrization of v is unchanged.
In case of v being a geometric function as in Definition 2.2, the algorithm has to be applied to the set of control points component-wise.
The procedure can also be seen as a matrix-vector multiplication: if $\mathbf{v} = [v_i]_{i=1}^{n}$ and $\bar{\mathbf{v}} = [\bar{v}_i]_{i=1}^{n+1}$ are the coefficients vectors belonging respectively to $\mathbb{R}^n$ and $\mathbb{R}^{n+1}$, then

$$\bar{\mathbf{v}} = R\mathbf{v}, \tag{2.12}$$

with $R = R(\Xi, \bar{\Xi})$ being a $(n+1) \times n$ matrix called *refining matrix*. It does not depends on the particular Fourier coefficient but only on the two knot vectors Ξ and $\bar{\Xi}$. It can be easily deduced from 2.11 that the entries r_{ij} of R are

$$r_{ij} = \begin{cases} 1 & 1 \leq i \leq k-p & \text{and } i = j \\ \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k-p+1 \leq i \leq k & \text{and } i = j \\ 1 - \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k-p+1 \leq i \leq k & \text{and } i+1 = j \\ 1 & k+1 \leq i \leq n+p+2 & \text{and } i+1 = j, \\ 0 & \text{otherwise.} & \end{cases} \tag{2.13}$$

If multiple knots have to be added, then the h-refinement has to be applied multiple times. The order of insertion is irrelevant, i.e. inserting $\bar{\xi}$ and then $\tilde{\xi}$ leads to the same coefficients as inserting first $\tilde{\xi}$ and then $\bar{\xi}$.
In terms of refining matrices, if $\bar{\xi}_1, ..., \bar{\xi}_m$ are the new knots then an ordered sequence of nested knot vectors is created:

$$\Xi_0 = \Xi,$$

$$\Xi_j = \Xi_{j-1} \cup \{\bar{\xi}_j\} \quad j = 1, ..., m,$$
$$\Xi_m = \Xi \cup \{\bar{\xi}_1, ..., \bar{\xi}_m\} = \bar{\Xi}.$$

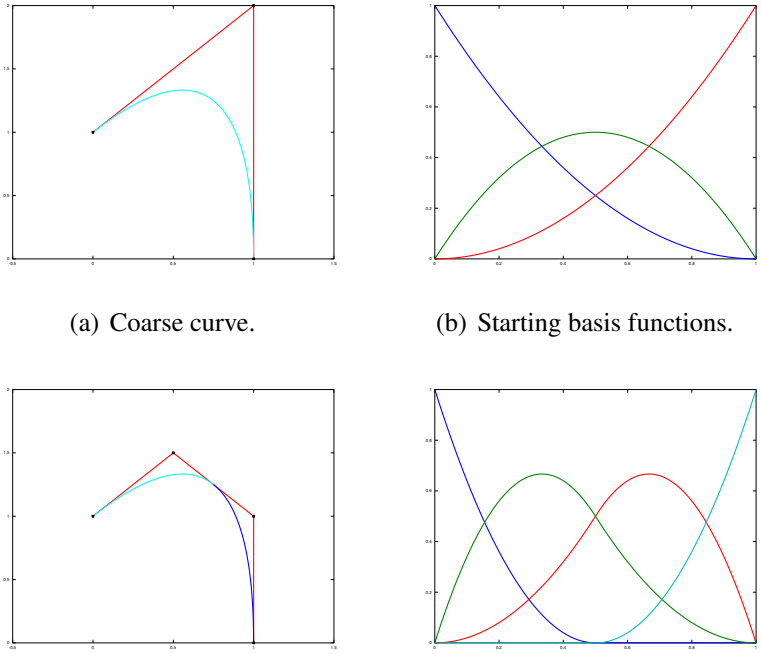
Each time a new knot is added, a new refining matrix $R(\Xi_j, \Xi_{j+1}) \in \mathbb{R}^{(n+j+1)\times(n+j)}$ is computed. The complete h-refining process from Ξ to $\bar{\Xi}$ is then expressed[2] by the products of the refining matrices:

$$R(\Xi, \bar{\Xi}) = \prod_{j=1}^{m} R(\Xi_{m-j}, \Xi_{m-j+1}). \tag{2.14}$$

Once again the full refining matrix $R(\Xi, \bar{\Xi})$ does not depends on the order of the insertion of the knots, although the order used for the creation of the nested knot vectors must be kept in the productory.
In the multivariate case, if the knot vectors $\boldsymbol{\Xi} = \{\Xi_d\}_{d=1}^{D}$ are h-refined to $\bar{\boldsymbol{\Xi}} = \{\bar{\Xi}_d\}_{d=1}^{D}$, then the refining matrix is computed with the Kronecker product:

$$R(\boldsymbol{\Xi}, \bar{\boldsymbol{\Xi}}) = \bigoplus_{d=1}^{D} R(\Xi_d, \bar{\Xi}_d).$$

[2]The author is confident that a more efficient procedure exists in order to apply multiple knots insertion, although for IGA purposes the computational costs of mesh refining are negligible.



(a) Coarse curve.

(b) Starting basis functions.

(c) h-refined curve.

(d) h-refined basis functions.

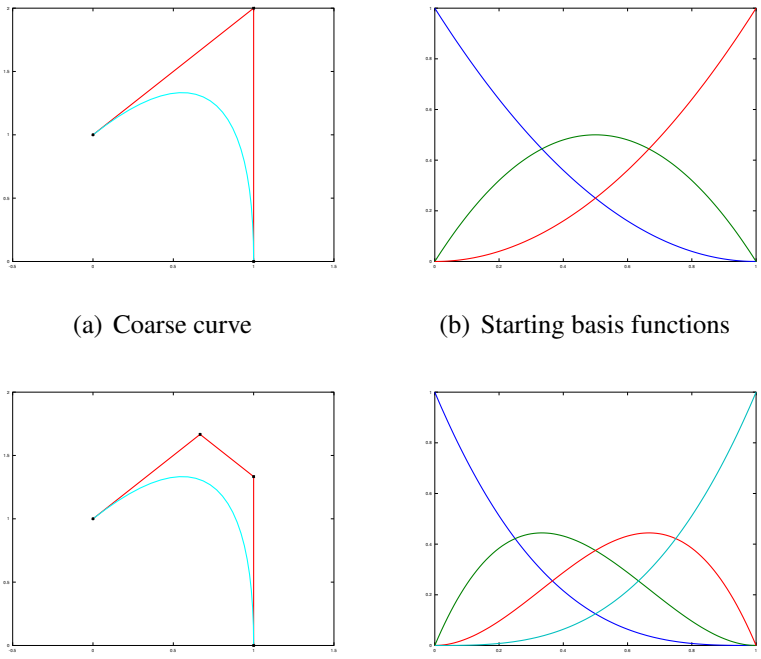
Figure 2.2: h-refinement at its finest.

Figure 2.2 shows how the h-refining process affects B-spline basis functions and geometries. From the open knot vector $\Xi = \{0, 0, 0, 1, 1, 1\}$ consisting of only one element we have three quadratic B-spline basis functions (b). The curve is generated from the three control points $\{(0, 1), (1, 2), (1, 0)\} \subset \mathbb{R}^2$, graphically connected in (a) with the red segments. For the h-refinement, knot $\bar{\xi} = 1/2$ is inserted. The mesh now owns two elements: $[0, 1/2]$, represented in light blue on the curve, and $[1/2, 1]$, represented in blue on the curve. The curve is generated from 4 basis functions (d) and hence the 4 control points (c) obtained from 2.11. Before the h-refinement the curve is a linear combination of polynomial functions, therefore it is $\mathcal{C}^\infty([0, 1])$. After the h-refinement the curve is a linear combination of piecewise polynomials that are globally $\mathcal{C}^1$, but it still preserve the previous geometrical regularity $\mathcal{C}^\infty$.

2.3.2 p-refinement

The p-refinement, or *degree elevation* in CAD literature, is a procedure aimed to raise the polynomial degree of the B-spline space. It is similar to the degree

refinement of the spectral methods.
There are different algorithms that perform p-refinement, and we refer to [28] for the theoretical concepts and to [24] for the algorithm used in the code.
Independently from the used algorithm, p-refinement raises the polynomial degree of the entire B-spline space, dimension-wise. One can elevate the degree along a single direction but cannot mix different degrees along the same direction.
Moreover, p-refinement does not change the shape and parametrization of a geometry. It also maintain the regularity at the knots so that the multiplicities of the knots are elevated as well. This means that, for example, a globally $\mathcal{C}^1$ B-spline space of degree $p = 3$, once is p-refined by 4 degrees, will become a $\mathcal{C}^1$ B-spline space of degree 7. Any internal knot of multiplicity $m = 2$ will become a knot of multiplicity $m = 6$.



(a) Coarse curve (b) Starting basis functions

(c) p-refined curve (d) p-refined basis functions

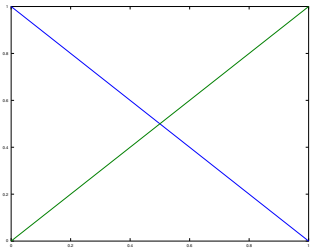
Figure 2.3: p-refinement at its finest.

Figure 2.3 shows the p-refinement process applied to the curve of Figure 2.2. The p-refinement consists of augmenting the degree from 2 to 3. The number of B-spline basis functions goes from 3 to 4 because the coarse knot vector contains only one element. As already stated, p-refinement doesn't rise the number of

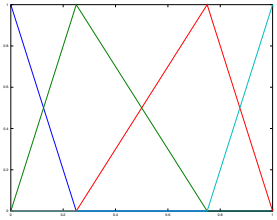
elements nor the global regularity. Once again, the curve is geometrically and parametrically unchanged. It is worth to notice that the new 4 control points are different from those obtained with h-refinement.

2.3.3 k-refinement

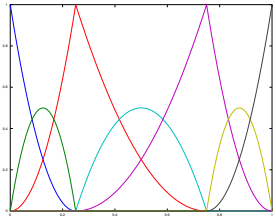
The k-refinement arises from the fact that h-refinement and p-refinement don't commute. The h-refinement adds one basis function for each inserted knot, while p-refinement adds a basis function for each element $\hat{Q}$ in the knot vector. If we start from an open knot vector of degree p with no intermediate knots and hence only one element, there are $p+1$ basis functions. With an h-refinement of n new distinct knots with multiplicity 1, we obtain $p+1+n$ basis functions over $n+1$ mesh elements, with global regularity $\mathcal{C}^{p-1}$. Now, p-refining the knot vector m times adds a basis function for each element, therefore the p-refined B-spline space has dimension $p+1+n+m(n+1)$ and global regularity $\mathcal{C}^{p-1}$.
Starting again from the single element knot vector, the application of p-refinement m times generates a B-spline space of $p+1+m$ functions belonging to $\mathcal{C}^{p+m-1}$. The h-refinement adds n basis function and we obtain a B-spline space with dimension $p+1+n+m$ and maximum regularity $\mathcal{C}^{p+m-1}$.
This means that with a wise application of p- and h-refinement, it is possible to both elevate the polynomial degree and regularity, and decrease the mesh size.
It is clear that a coarse mesh size consisting of only one element is not a real world scenario: complex geometries of industrial or engineering interest are described with more than 1 element. Anyway, applying h-refinement before p-refinement will results in an overabundance of basis functions with no regularity gains, at least across the new element created from h-refinement. With the wise order there are no regularity gains across coarse mesh knots but basis functions are kept in a minimum number with the highest regularity across the new knots.
Figure 2.4 shows how h-refinement and p-refinement don't commute. Starting from a one-element degree 1 open knot vector, the h-refinement consists in inserting knots $\xi = 1/4$ and $\xi = 3/4$, while p-refinement augments the polynomial degree by 4.



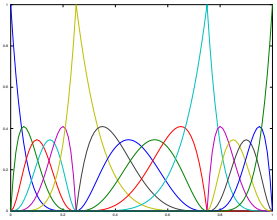
(a) Basis functions of degree $p = 1$ from knot vector $\Xi = \{0, 0, 1, 1\}$.



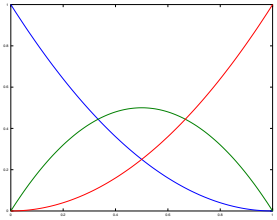
(b) h-refinement of (a).



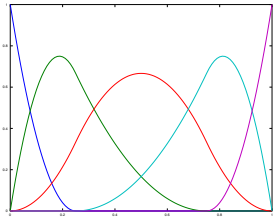
(c) p-refinement of (b).



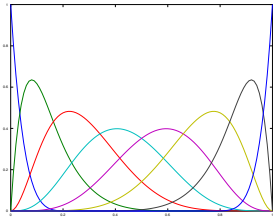
(d) Unwise result.



(e) p-refinement of (a).



(f) h-refinement of (e).



(g) k-refined basis functions.

Figure 2.4: k-refinement at its finest.

2.4 NURBS

Definition 2.5. Given a knot vector $\Xi = \{\xi_1 \leq ... \leq \xi_{n+p+1}\}$ and a set of *weights* $w = \{w_i\}_{i=1}^n$ we call *univariate NURBS basis functions* (for brevity *NURBS*) the n functions $\hat{N}_i^p : [0, 1] \longrightarrow \mathbb{R}$ defined as

$$\hat{N}_i^p(\xi) = \frac{w_i \hat{B}_i^p(\xi)}{W(\xi)}, \tag{2.15}$$

where $W : [0,1] \longrightarrow \mathbb{R}$ is the so-called *weight function* defined as

$$W(\xi) = \sum_{i=1}^{n} w_i \hat{B}_i^p(\xi). \tag{2.16}$$

We denote the linear space of NURBS functions with the following notation:

$$\hat{\mathcal{N}} = \hat{\mathcal{N}}(\Xi, p, w) = \text{span}\left\{\hat{N}_i^p,\ i = 1, ..., n\right\}. \tag{2.17}$$

The multivariate tensor product NURBS space is denoted with

$$\hat{\mathcal{N}} = \hat{\mathcal{N}}(\boldsymbol{\Xi}, \mathbf{p}, \mathbf{w}) = \text{span}\left\{\hat{N}_{\mathbf{i}}^{\mathbf{p}},\ \mathbf{i} = 1, ..., \mathbf{n}\right\}. \tag{2.18}$$

NURBS basis functions are piecewise rational polynomials. They inherit all the properties of B-splines and allow for the construction of geometries that are common in industrial applications, such as conics, spheres, cylinders and many others. See Figures 4.1 and 4.2 for two examples of NURBS exact geometries representing curved shapes.
We note that when the weights are equally 1, because of the partition of unity property the weight function W is identically 1, and NURBS basis functions coincide with B-splines. NURBS are hence a superset of B-splines.
Due to being a B-spline function itself, the weight function W is invariant under any type of refinement, therefore it is set at the coarsest level of the mesh. The collection of weights, though, has to be refined in order to use NURBS basis functions as a discrete basis for a Petrov-Galerkin discretization. This is the only motivation for a full implementation of the three refining processes in an IGA code.

2.5 Basis functions in physical domain

In IGA the physical domain Ω of a PDE is described with a B-spline or NURBS geometry $\mathbf{F}$, while basis functions are defined only in the parametric space $[0,1]^D$. The construction of discrete function spaces in physical domain is performed via push-forward: if we denote $\Omega = \mathbf{F}([0,1]^D)$ and $x = \mathbf{F}(\xi)$, we define (again) B-spline basis functions in physical domain as

$$B_{\mathbf{i}}^{\mathbf{p}} : \Omega \longrightarrow \mathbb{R}, \qquad B_{\mathbf{i}}^{\mathbf{p}}(x) = \hat{B}_{\mathbf{i}}^{\mathbf{p}}(\mathbf{F}^{-1}(x)) = \hat{B}_{\mathbf{i}}^{\mathbf{p}} \circ \mathbf{F}^{-1}(x). \tag{2.19}$$

where $\hat{B}_{\mathbf{i}}^{\mathbf{p}} \in \hat{\mathcal{B}}(\boldsymbol{\Xi}, \mathbf{p})$. The space of B-splines on physical domain is denoted

$$\mathcal{B} = \mathcal{B}(\boldsymbol{\Xi}, \mathbf{p}). \tag{2.20}$$

The same "hat/no-hat" notation is used for NURBS basis functions and spaces:

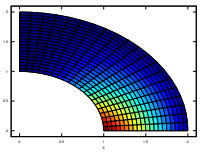
$$N_{\mathbf{i}}^{\mathbf{p}} : \Omega \longrightarrow \mathbb{R}, \qquad N_{\mathbf{i}}^{\mathbf{p}}(x) = \hat{N}_{\mathbf{i}}^{\mathbf{p}}(\mathbf{F}^{-1}(x)) = \hat{N}_{\mathbf{i}}^{\mathbf{p}} \circ \mathbf{F}^{-1}(x). \tag{2.21}$$

$$\mathcal{N} = \mathcal{N}(\boldsymbol{\Xi}, \mathbf{p}, \mathbf{w}). \tag{2.22}$$

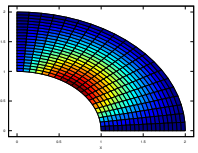
From an implementation point of view, the evaluation of basis functions in physical domain is trivial, while the evaluation of derivatives require the computation of the Jacobian matrix of $\mathbf{F}$ or, for second derivatives, the Hessian tensor. The first derivatives are computed in the following way:

$$\nabla B_{\mathbf{i}}^{\mathbf{p}} = \nabla \hat{B}_{\mathbf{i}}^{\mathbf{p}} J\mathbf{F}^{-1}.$$

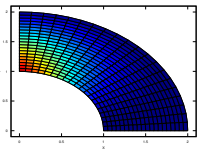
Meshes and elements are mapped via $\mathbf{F}$ to form equivalent objects in physical domain. We then define *physical element* the set $K = \mathbf{F}(Q)$ with $Q \in \mathcal{Q}$, and *physical mesh* the set $\mathcal{K}$ of physical elements. The *physical element size* is $h_K = \|\nabla \mathbf{F}\|_{L^\infty(Q)} h_Q$ and *physical mesh size* is $h = \max_{K \in \mathcal{K}} h_K$.



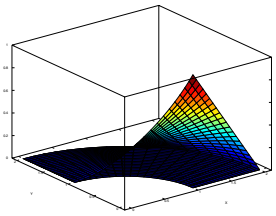
(a) $N_{1,1}$



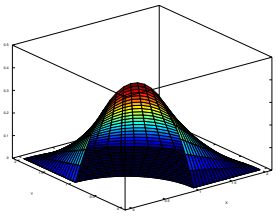
(b) $N_{2,1}$



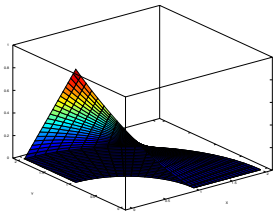
(c) $N_{3,1}$



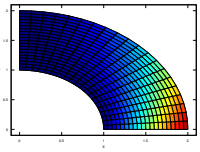
(d) $N_{1,1}$



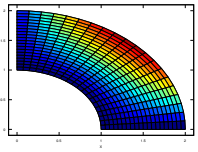
(e) $N_{2,1}$



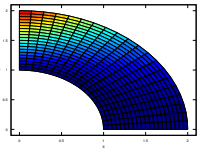
(f) $N_{3,1}$



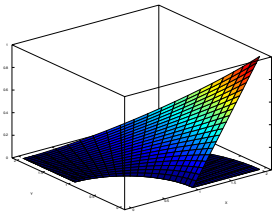
(g) $N_{1,2}$



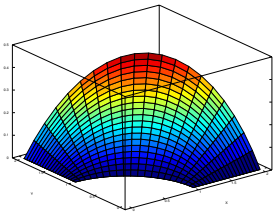
(h) $N_{2,2}$



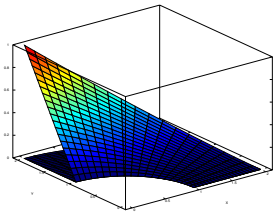
(i) $N_{3,2}$



(j) $N_{1,2}$



(k) $N_{2,2}$



(l) $N_{3,2}$

Figure 2.5: Examples of degrees $p_1 = 2, p_2 = 1$ NURBS basis function in the quarter annulus of Figure 4.1 as physical domain.

2.6 Approximation estimates

We present now some approximation estimates for B-spline and NURBS spaces with decreasing mesh size h, the polynomial degree kept fixed. The main idea is to adapt the well-known Bramble-Hilbert Lemma ([15]) to B-spline functions, and then extend it first to NURBS and second to the physical domain $\Omega = \mathbf{F}([0,1]^D)$. For the full proofs of the following Lemmas and Theorems, we refer to [7].
Before starting with the approximation results, some machinery and the introduction of a functional space must be fixed:

- We consider a family of meshes $\{\mathcal{Q}_h\}_h$ associated to the family of knot vector $\{\mathbf{\Xi}_h\}$, h being the mesh size of $\mathcal{Q}_h$. We assume that the family is shape regular, i.e. the mesh size h_Q of each element is bounded uniformly with respect to Q and h. As usual, the equivalent physical mesh family $\{\mathcal{K}_h\}_h$ is defined.

- We consider the corresponding families of B-spline and NURBS spaces constructed from the meshes $\mathcal{Q}_h$:

$$\left\{\hat{\mathcal{B}}_h = \hat{\mathcal{B}}_h(\mathbf{\Xi}_h, \mathbf{p})\right\}_h, \qquad \left\{\hat{\mathcal{N}}_h = \hat{\mathcal{N}}_h(\mathbf{\Xi}_h, \mathbf{p}, \mathbf{w})\right\}_h,$$
$$\left\{\mathcal{N}_h = \mathcal{N}_h(\mathbf{\Xi}_h, \mathbf{p}, \mathbf{w})\right\}_h.$$

  For B-spline or NURBS discrete spaces in physical domain, there will only be considered the notation $\mathcal{N}_h$ as B-splines are a particular case of NURBS.

- We denote with C a positive dimensionless constant which depends only on dimension D, polynomial degrees $\mathbf{p}$ and the shape regularity of $\{\mathcal{Q}_h\}_h$. Another constant C_{shape} is considered, depending on h and the geometry of Ω. It does not depend on the size of the domain, i.e. it is homogeneous of degree 0 with respect to the weight function W and the Jacobian $J\mathbf{F}$, although it depends on $W/\|W\|_{L^\infty}$ and $J\mathbf{F}/\|J\mathbf{F}\|_{L^\infty}$.

- The natural number p as polynomial degree has to be considered as the minimum of the degrees in $\mathbf{p}$.

- We define *support extension* $\tilde{Q}$ of a parametric element $Q \in \mathcal{Q}$ the open set

$$\tilde{Q} = \bigotimes_{d=1}^{D} \left(\xi_{i_d}^d - m_{i_d}, \xi_{i_d}^d + m_{i_d} + 1\right) \subset \mathbb{R}^D.$$

  with m_{i_d} the multiplicity of the knot ξ_{i_d} as in Definition 2.1. The support extension of an element is the union of all supports of B-splines that don't vanish on Q. Via push-forward, the support extension is defined on physical elements: the support extension of the physical element K is $\tilde{K} = \mathbf{F}(\tilde{Q})$.

- For $D > 1$, we need the notion of reduced regularity across two adjacent elements of a mesh, i.e. two elements that share an edge, a face or a $D-1$-dimensional hypercube in general, but not geometric entities whose dimension is below $D-1$. We denote with m_{Q_1,Q_2} the regularity of the basis functions along the direction going from element Q_1 to the adjacent element Q_2. If the edge (or face) shared between Q_1 and Q_2 correspond to the knot $\xi_{i_d}^d$, then
$$m_{Q_1,Q_2} = p_d - m_{i_d},$$

Finally, we need a functional space living between the classic Sobolev space $H_0^1(\Omega)$ ([3]), and the broken Sobolev space ([27]), typical from the Discontinuous Galerkin Methods.

Definition 2.6. We define *bent Sobolev space* $\mathcal{H}^m = \mathcal{H}^m(\Omega)$ the set

$$\mathcal{H}^m = \left\{ v \in L^2\left((0,1)^d\right) \;\middle|\; \begin{array}{l} v_{|Q} \in H^m(Q) \quad \forall Q \in \mathcal{Q}, \\ \\ \nabla^k v_{|Q_1} = \nabla^k v_{|Q_2} \text{ on } \partial Q_1 \cap \partial Q_2 \\ \forall k = 0, \dots, \min\{m_{Q_1,Q_2}, m-1\} \\ \forall Q_1, Q_2 \in \mathcal{Q},\ \partial Q_1 \cap \partial Q_2 \neq \emptyset \end{array} \right\}, \tag{2.23}$$

where ∇^k is the k-th differentiation operator in the trace sense, with $\nabla^0 v = v$. This space, endowed with seminorms and norm

$$|v|_{\mathcal{H}^i}^2 = \sum_{Q \in \mathcal{Q}} |v|_{H^i(Q)}^2, \quad i = 0, \dots, m, \qquad ||v||_{\mathcal{H}^m}^2 = \sum_{i=0}^{m} |v|_{\mathcal{H}^i}^2, \tag{2.24}$$

is a Hilbert space. The restriction of $\mathcal{H}^m$ to the extension support of an element is denoted with

$$\mathcal{H}^m(\tilde{Q}) = \left\{ v_{|\tilde{Q}} | v \in \mathcal{H}^m \right\},$$

$$|v|_{\mathcal{H}^i(\tilde{Q})}^2 = \sum_{Q \in \mathcal{Q}, Q \cap \tilde{Q} \neq \emptyset} |v|_{\mathcal{H}^i(Q)}^2, \qquad \|v\|_{\mathcal{H}^m(\tilde{Q})}^2 = \sum_{i=1}^{m} |v|_{\mathcal{H}^i(\tilde{Q})}^2.$$

It can be noticed from the definition that bent Sobolev spaces depend on an underlying mesh. For the following analysis mesh families have been introduced and so is the family of spaces $\{\mathcal{H}_h^m\}_h$, where $\mathcal{H}_h^m$ is the bent Sobolev space associated with mesh $\mathcal{Q}_h$. We are now ready for the approximation results.

2.6.1 Approximation in parametric domain

Lemma 2.7. Let k and l be integer indexes with $0 \leq k \leq l \leq p+1$. Given $Q \in \mathcal{Q}_h$, its extension support $\tilde{Q}$ and $v \in \mathcal{H}_h^l$, there exist $b \in \hat{\mathcal{B}}_h$ such that

$$|v - b|_{\mathcal{H}_h^k(\tilde{Q})} \leq C h_Q^{l-k} |v|_{\mathcal{H}_h^l(\tilde{Q})}. \tag{2.25}$$

Proof. See [7]. □

We now introduce a family of projectors on the space $\hat{\mathcal{B}}_h$:

$$\Pi_{\hat{\mathcal{B}}_h} : L^2\left([0,1]^D\right) \longrightarrow \hat{\mathcal{B}}_h, \qquad \Pi_{\hat{\mathcal{B}}_h} v = \sum_{\mathbf{i}}^{\mathbf{n}} \lambda_{\mathbf{i}}[v] \hat{B}_{\mathbf{i}}. \tag{2.26}$$

where the functionals $\lambda_{\mathbf{i}}$ are as in 2.5. The projectors have the two following properties:

- **Spline preserving**: $\Pi_{\hat{\mathcal{B}}_h} b = b$ for all $b \in \hat{\mathcal{B}}_h$;
- **Local stability**: $\|\Pi_{\hat{\mathcal{B}}_h} v\|_{L^2(Q)} \leq C \|v\|_{L^2(\tilde{Q})}$ for all $v \in L^2([0,1]^D)$ and for all $Q \in \mathcal{Q}_h$.

Lemma 2.8. Let k and l be integer indexes with $0 \leq k \leq l \leq p+1$. Then, for all $Q \in \mathcal{Q}_h$ we have

$$|v - \Pi_{\hat{\mathcal{B}}_h} v|_{H^k(Q)} \leq C h_Q^{l-k} |v|_{\mathcal{H}_h^l(\tilde{Q})}, \qquad \forall v \in \mathcal{H}_h^l(\tilde{Q}) \cap L^2([0,1]^D). \tag{2.27}$$

Proof. See [7]. □

The passage from B-splines to NURBS in parametric domain requires a new projector operator, inherited from 2.26:

$$\Pi_{\hat{\mathcal{N}}_h} : L^2([0,1]^D) \longrightarrow \hat{\mathcal{N}}_h, \qquad \Pi_{\hat{\mathcal{N}}_h} v = \frac{\Pi_{\hat{\mathcal{B}}_h}(Wv)}{W}, \tag{2.28}$$

where W is the weight function as in 2.16.

Lemma 2.9. Let k and l be integer indexes with $0 \leq k \leq l \leq p+1$. Then

$$|v - \Pi_{\hat{\mathcal{N}}_h} v|_{H^k(Q)} \leq C_{shape} h_Q^{l-k} \|v\|_{\mathcal{H}_h^l(\tilde{Q})}, \qquad \forall v \in \mathcal{H}_h^l(\tilde{Q}),\ \forall Q \in \mathcal{Q}_h. \tag{2.29}$$

Proof. See [7]. □

2.6.2 Approximation in physical domain

The last projector we need is the push-forward version of 2.28.

$$\Pi_{\mathcal{N}_h} : L^2(\Omega) \longrightarrow \mathcal{N}_h, \qquad \Pi_{\mathcal{N}_h} v = \left(\Pi_{\hat{\mathcal{N}}_h}(v \circ \mathbf{F})\right) \circ \mathbf{F}^{-1}. \tag{2.30}$$

Theorem 2.10. Let k and l be integer indexes with $0 \leq k \leq l \leq p+1$. Given $Q \in \mathcal{Q}_h$, $K = \mathbf{F}(Q)$ and their support extensions $\tilde{Q}$ and $\tilde{K}$, for all $v \in L^2(\Omega) \cap H^l(\tilde{K})$ we have

$$|v - \Pi_{\mathcal{N}_h} v|_{H^k(K)} \leq C_{shape} h_K^{l-k} \sum_{i=0}^{l} \|\nabla \mathbf{F}\|^{i-l}_{L^\infty(\tilde{Q})} |v|_{H^i(\tilde{K})}. \tag{2.31}$$

Moreover, for all $v \in H^l(\Omega)$ we have

$$\sum_{K \in \mathcal{K}_h} |v - \Pi_{\mathbf{n}_h} v|^2_{\mathcal{H}^k_h(K)} \leq C_{shape} \sum_{K \in \mathcal{K}_h} h_K^{2(l-k)} \sum_{i=0}^{l} \|\nabla \mathbf{F}\|^{2(i-l)}_{L^\infty(\mathbf{F}^{-1}(Q))} |v|^2_{H^i(K)}. \tag{2.32}$$

Proof. See [7]. □

From Theorem 2.10 we note that NURBS discrete spaces (and B-spline as a consequence) deliver optimal rate of convergence, as in classic FEM.
Finally, the application of Dirichlet boundary conditions can be proven with small efforts. Let $\partial\Omega$ be the boundary of the domain Ω, with Γ_D being the positive-measured part of $\partial\Omega$ where Dirichlet conditions hold. We assume for simplicity that Γ_D consists only of edges or faces of the mesh. Let $\gamma_D = \mathbf{F}(\Gamma_D)^{-1}$ be the Dirichlet boundary in the parametric domain $[0,1]^D$. We consider also the two functional spaces

$$H^1_{\Gamma_D} = \left\{ v \in H^1(\Omega) | v = 0 \text{ on } \Gamma_D \right\}, \tag{2.33}$$

$$H^1_{\gamma_D} = \left\{ v \in H^1([0,1]^D) | v = 0 \text{ on } \gamma_D \right\}, \tag{2.34}$$

and the adapted version of the projectors:

$$\Pi^0_{\hat{\mathcal{B}}_h} : H^1_{\gamma_D}((0,1)^D) \longrightarrow \hat{\mathcal{B}}_h \cap H^1_{\gamma_D}([0,1]^D),$$

$$\Pi^0_{\hat{\mathcal{B}}_h} v = \sum_{\substack{\mathbf{i}=1, \\ \hat{B}_\mathbf{i} \in H^1_{\gamma_D}((0,1)^D)}}^{\mathbf{n}} \lambda_\mathbf{i} \hat{B}_\mathbf{i}; \tag{2.35}$$

$$\Pi^0_{\hat{\mathcal{N}}_h} : H^1_{\gamma_D}((0,1)^D) \longrightarrow \hat{\mathcal{N}}_h \cap H^1_{\gamma_D}((0,1)^D), \qquad \Pi^0_{\hat{\mathcal{N}}_h} v = \frac{\Pi^0_{\hat{\mathcal{B}}_h} W v}{W}; \tag{2.36}$$

$$\Pi^0_{\mathcal{N}_h} : H^1_{\Gamma_D}(\Omega) \longrightarrow \mathcal{N}_h \cap H^1_{\Gamma_D}(\Omega), \qquad \Pi^0_{\mathcal{N}_h} v = \left(\Pi^0_{\hat{\mathcal{N}}_h}(v \circ \mathbf{F})\right) \circ \mathbf{F}^{-1}. \tag{2.37}$$

Lemma 2.11. Let k and l be integer indexes with $0 \leq k < l \leq p+1$ and $l \geq 1$. Given $Q \in \mathcal{Q}_h$, its support extension $\tilde{Q}$ and $v \in \mathcal{H}_h^l(\tilde{Q}) \cap H^1_{\gamma_D}((0,1)^D)$, there exists $b \in \hat{\mathcal{B}}_h \cap H^1_{\gamma_D}((0,1)^D)$ such that

$$|v-b|_{\mathcal{H}_h^k(\tilde{Q})} \leq C h_Q^{l-k} |v|_{\mathcal{H}_h^l(\tilde{Q})}. \tag{2.38}$$

Proof. See [7]. □

Theorem 2.12. Let k and l be integer indexes with $0 \leq k < l \leq p+1$. Then, for all $v \in H^l(\Omega) \cap H^1_{\Gamma_D}(\Omega)$ we have

$$\sum_{K \in \mathcal{K}_h} |v - \Pi^0_{\mathcal{N}_h} v|^2_{\mathcal{H}_h^k(K)} \leq C_{shape} \sum_{K \in \mathcal{K}_h} h_K^{2(l-k)} \sum_{i=1}^{l} \|\nabla \mathbf{F}\|^{2(i-l)}_{L^\infty(\mathbf{F}^{-1}(K))} |v|^2 H^i(K). \tag{2.39}$$

Proof. See [7]. □

2.6.3 Approximation for advection-diffusion equations

As first introduced in Chapter 1, the advection-diffusion problem is finding a function $u : \Omega \longrightarrow \mathbb{R}$ such that

$$\begin{cases} -\nabla \cdot (\mathbf{K} \nabla u) + \mathbf{b} \cdot \nabla u = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma_D, \end{cases} \tag{2.40}$$

where the ingredients are:

- $\Omega \subset \mathbb{R}^D$ is an open domain with $\Gamma_D \subset \partial\Omega$;
- $f : \Omega \longrightarrow \mathbb{R}$ is the body force;
- $\mathbf{K} : \Omega \longrightarrow \mathbb{R}^{D \times D}$ is the diffusivity tensor. For simplicity we assume that $\mathbf{K} = \kappa \mathbf{Id}$, with κ a positive constant. The tensor is hence symmetric, positive definite and represent an isotropic diffusion;
- $\mathbf{b} : \Omega \longrightarrow \mathbb{R}^D$ is the velocity field. Again for simplicity we assume divergence-free field, i.e. $\nabla \cdot \mathbf{b} = 0$.

Integrating the assumption on $\mathbf{K}$ we deduce the following operators from 2.40:

$$\mathcal{L} v = \mathcal{L}_{adv} v + \mathcal{L}_{diff} v, \qquad \mathcal{L}_{adv} v = \mathbf{b} \cdot \nabla v, \quad \mathcal{L}_{diff} v = -\kappa \Delta v. \tag{2.41}$$

The weak formulation of the problem is stated as follow:

$$\text{Find } u \in H^1_{\Gamma_D}(\Omega) \text{ such that } a(u,v) = F(v), \text{ for all } v \in H^1_{\Gamma_D}(\Omega). \tag{2.42}$$

where

$$a(v,w) = (\mathcal{L}v, w)_{L^2(\Omega)} = -\int_\Omega \nabla v \cdot (bw - \kappa \nabla w)\, dx, \tag{2.43}$$

$$F(v) = (f, v)_{L^2(\Omega)} = \int_\Omega vf\, dx. \tag{2.44}$$

It is well-known ([29]) that the well-posedness of the associated discrete problem depends on whether the diffusive part $\mathcal{L}_{diff}$ or the advective part $\mathcal{L}_{adv}$ of the operator is dominant. To ensure continuity and coercivity of the bilinear form $a(\cdot,\cdot)$ one can prove that

$$a(v,v) \geq \alpha \|v\|^2_{H^1(\Omega)}, \qquad \forall v \in H^1_(\Omega),$$

$$a(v,w) \leq M \|v\|_{H^1(\Omega)} \|w\|_{H^1(\Omega)}, \qquad \forall v, w \in H^1_(\Omega),$$

where

$$\alpha = \frac{\kappa}{1 + C^2_\Omega}, \qquad M = \kappa + \|b\|_{L^\infty(\Omega)},$$

with C_Ω the Cauchy inequality constant over the domain Ω. If the Sobolev space $H^1(\Omega)$ is approximated with a discrete space $\mathcal{V}_h$, it can be shown for the discrete solution $u_h \in \mathcal{V}_h$ that

$$\|\nabla u_h\|_{L^2(\Omega)} \leq \frac{C_\Omega}{\kappa} \|f\|_{L^2(\Omega)}, \tag{2.45}$$

$$\|u - u_h\|_{H^1(\Omega)} \leq \frac{M}{\alpha} \inf_{v_h \in \mathcal{V}_h} \|u - v_h\|_{H^1(\Omega)}. \tag{2.46}$$

Inequality 2.45 shows that if κ tends to 0, the control over the discrete solution derivatives may explode and be inaccurate. On the other hand, the constant $\frac{M}{\alpha}$ in 2.46 is proportional to $\frac{\|b\|_{L^\infty(\Omega)}}{\kappa}$, therefore the error estimate may be inaccurate whenever there is a considerable jump between the diffusivity and advective coefficients. The main concept here is that a dominant advective part in the equations leads to ill-posed discrete problems, especially whenever the solution of the problem presents the so-called *boundary layers*, region of the domain where the solution gradient is relatively high.

As an indicator of the well- or ill-posedness of the discrete problem, we introduce the *Péclet number*

$$\mathbb{P}\mathrm{e} = \frac{\|b\|_{L^\infty(\Omega)} h}{2\kappa},$$

for which, depending on the chosen mesh-size h, $\mathbb{P}\mathrm{e} < 1$ leads to well-posed problems while $\mathbb{P}\mathrm{e} > 1$ to ill-posed problems.

The ill-posedness is thus avoidable with a sufficient small mesh size h, although if $\kappa \ll 1$ and hence $\mathbb{P}\mathrm{e} \gg 1$ the required h may be computationally unfeasible. This

motivates the adoption of stabilization methods. The streamline upwind Petrov-Galerkin (SUPG) stabilization, first introduced in [16], is a technique designed to enhance stability of the Galerkin approach without compromising its accuracy. If we denote with $\tilde{\Omega}$ the union of the interiors of elements $K \in \mathcal{K}$, the SUPG-stabilized problem is stated as follow:

$$\text{Find } u_h \in \mathcal{V}_h \text{ such that } a_{SUPG}(u_h, v_h) = F_{SUPG}(v_h), \text{ for all } v_h \in \mathcal{V}_h. \quad (2.47)$$

where the bilinear form and the functional are respectively defined as

$$a_{SUPG}(v, w) = a(u, v) + (\tau \mathbf{b} \cdot \nabla v, \mathcal{L}u)_{L^2(\tilde{\Omega})}, \quad (2.48)$$

$$F_{SUPG}(v) = F(v) + (\tau \mathbf{b} \cdot \nabla v, f)_{L^2(\tilde{\Omega})}. \quad (2.49)$$

The choice of the stabilization parameter τ is critical for accuracy, stability and convergence of the methods. Several choices have been explored in literature.

Lemma 2.13. The bilinear form 2.48 satisfies the following inequalities:

$$a_{SUPG}(v, v) \geq C||w||^2_{(1)}, \qquad \forall v \in \mathcal{N}_h, \quad (2.50)$$

$$a_{SUPG}(v, w) \leq C||v||_{(1)}||w||_{(2)}, \qquad \forall v \in \mathcal{N}_h, \quad \forall w \in H^1_{\Gamma_D}(\Omega) \cap H^2(\tilde{\Omega}), \quad (2.51)$$

where the two norms $\|\cdot\|_{(1)}$ and $\|\cdot\|_{(2)}$ are defined as follow:

$$||v||^2_{(1)} = \kappa \|\nabla v\|^2_{L^2(\Omega)} + \|\sqrt{\tau}\mathbf{b} \cdot \nabla v\|^2_{L^2(\Omega)}. \quad (2.52)$$

$$\|v\|^2_{(2)} = k\|\nabla v\|^2_{L^2(\Omega)} + \|\sqrt{\tau} v\|^2_{L^2(\Omega)} + \|\sqrt{\tau}\mathbf{b} \cdot \nabla v\|^2_{L^2(\Omega)} + \|\sqrt{\tau}\kappa \Delta v\|^2_{L^2(\Omega)}. \quad (2.53)$$

Proof. See [7]. □

A proper combination of the above inequalities gives a bound on the numerical error in the discrete solution in terms of the interpolation error. If u is solution of 2.40 and u_h is a solution of 2.47, then

$$\|u - u_h\|_{(1)} \leq C\|u - \Pi^0_{\mathcal{N}_h} u\|_{(2)}. \quad (2.54)$$

Finally, we have the following Theorem establishing convergence rates for the SUPG-stabilized NURBS-discretized advection-diffusion problem:

Theorem 2.14. If the solution u of 2.40 belongs to $H^{p+1}(\Omega)$, then there exists $C_{shape} > 0$ such that

$$\|u - u_h\|^2_{(1)} \leq C \sum_{K \in \mathcal{K}} \left(\|\mathbf{b}\|_{L^\infty(K)} h_K^{2p+1} + \kappa h_K^{2p} \right) \sum_{i=0}^{p+1} \|\nabla \mathbf{F}\|^{2(i-p-1)}_{L^\infty(\mathbf{F}^{-1}(K))} |u|^2_{H^1(K)}. \quad (2.55)$$

Proof. See [7]. □

Chapter 3

Additive Schwarz Preconditioning

In this Chapter we briefly sketch the abstract theory at the foundation of Schwarz Preconditioning ([33]). A detailed explanation of the construction of local spaces and the coarse space in IGA will follow. We will finally prove a bound for the condition number of the preconditioned matrix arising from the IGA discretization of the scalar elliptic equation. We refer to [9] for the original setting.

3.1 Abstract theory

Abstract theory on multilevel Additive Schwarz Preconditioning relies on three Assumptions. The theory is built upon a finite-dimensional Hilbert space $\mathcal{V}$, a symmetric positive definite bilinear form $a : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ and a linear functional $f : \mathcal{V} \to \mathbb{R}$. The abstract problem states:

$$\text{Find } u \in \mathcal{V} \text{ such that } a(u, v) = f(v) \quad \forall v \in \mathcal{V}. \tag{3.1}$$

As $\mathcal{V}$ is a finite-dimensional space, it is spanned by a finite basis so that functions in $\mathcal{V}$ can be expressed as linear combination of the basis functions:

$$\mathcal{V} = \text{span}\,\{\phi_1, ..., \phi_n\} \quad \Longrightarrow \quad u = \sum_{i=1}^{n} u_i \phi_i, \qquad v = \sum_{i=1}^{n} v_i \phi_i. \tag{3.2}$$

With a small abuse of notation, the Problem 3.1 can be rewritten as a linear system $Au = f$, where

$$A = [a_{ij}]_{i,j=1}^{n} \in \mathbb{R}^{n \times n}, \qquad a_{ij} = a(\phi_i, \phi_j); \tag{3.3}$$

$$f = [f_i]_{i=1}^{n} \in \mathbb{R}^{n}, \qquad f_i = f(\phi_i). \tag{3.4}$$

We assume that the space $\mathcal{V}$ is decomposed in a family of subspaces $\{\mathcal{V}_j\}_{j=0}^N$ such that

$$\mathcal{V} = R_0^T \mathcal{V}_0 + \sum_{j=1}^{N} R_i^T \mathcal{V}_j, \tag{3.5}$$

where $R_j^T : \mathcal{V}_j \to \mathcal{V}$ are interpolation operators. The decomposition of $\mathcal{V}$ is not unique, nor the subspaces $\mathcal{V}_j$ have to be proper linear subspaces. Spaces $\mathcal{V}_j$ with $j = 1, ..., N$ are typically associated to local subdomains of the PDE domain Ω, and hence referred as *local spaces*. The interpolation operators R_j^T are typically represented as matrices of 1 and 0. Their action consists in extending functions to zero from $\mathcal{V}_j$ to $\mathcal{V}$, while in the other direction the transposed operators R_j select the part of a function of $\mathcal{V}$ belonging to $\mathcal{V}_j$.
Generally speaking, even the space $\mathcal{V}_0$ has not to be a proper subspace of $\mathcal{V}$, although it is in a suitable IGA context. It is typically associated to a coarse global mesh and it will be referred as *coarse space*. The operator R_0^T is a proper interpolation operator from $\mathcal{V}_0$ to $\mathcal{V}$ while its transposed R_0 is a projection operator from $\mathcal{V}$ to $\mathcal{V}_0$.
Associated with the decomposition $\{\mathcal{V}_j\}$ there are the local bilinear forms and local system matrices:

$$\tilde{a}_j : \mathcal{V}_j \times \mathcal{V}_j \to \mathbb{R}, \qquad \tilde{A}_j : \mathcal{V}_j \to \mathcal{V}_j, \qquad j = 0, ..., N. \tag{3.6}$$

These local forms can be an approximation of the local action of $a(\cdot, \cdot)$. In our case, local forms and local matrices are defined directly from the global ones:

$$\tilde{a}_j (v_j, w_j) = a \left(R_j^T v_j, R_j^T w_j \right), \qquad v_j, w_j \in \mathcal{V}_j, \tag{3.7}$$

$$\tilde{A}_j = R_j A R_j^T. \tag{3.8}$$

With this explicit definition (the so-called *use of exact solver*), local forms $\tilde{a}_j$ are symmetric, positive definite bilinear forms.
Schwarz operators P_i are then defined on the operators $\tilde{P}_j : \mathcal{V} \to \mathcal{V}_j$, where $\tilde{P}_j v$ is such that

$$\tilde{a}_j \left(\tilde{P}_j v, w_j \right) = a \left(v, R_j^T w_j \right), \qquad \forall w_j \in \mathcal{V}_j. \tag{3.9}$$

The definition of $\tilde{P}_j$ is well-defined since the local forms are coercive. We then define the operators

$$P_j = R_j^T \tilde{P}_j : \mathcal{V} \to R_j^T \mathcal{V}_j \subset \mathcal{V}, \qquad j = 0, ..., N. \tag{3.10}$$

The element $P_j v$ is the only element that satisfies

$$a \left(P_j v, R_j^T w_j \right) = a \left(v, R_j^T w_j \right), \qquad \forall w_j \in \mathcal{V}_j. \tag{3.11}$$

It can be proven ([33]) that in matrix form, P_j can be written in the following way:

$$P_j = R_j^T \tilde{A}_j^{-1} R_j A, \qquad j = 0, ..., N. \tag{3.12}$$

Moreover, in case of exact solver, P_j is a projection, i.e. $P_j^2 = P_j$.
We now have all the ingredients to define the Additive Schwarz Operator.

Definition 3.1. Given a finite-dimensional Hilbert space $\mathcal{V}$ with decomposition $\{\mathcal{V}_j\}_{j=0}^N$, a symmetric positive definite bilinear form $a : \mathcal{V} \times \mathcal{V} \longrightarrow \mathbb{R}$ and projection operators as in 3.10, we define the *1-level Additive Schwarz Operator* the operator

$$P_{ASO}^{(1)} = \sum_{j=1}^{N} P_j = \sum_{j=1}^{N} R_j^T \tilde{A}_j^{-1} R_j A. \tag{3.13}$$

We define *1-level Additive Schwarz Preconditioner* the operator

$$\left(A_{ASO}^{(1)}\right)^{-1} = \sum_{j=1}^{N} R_j^T \tilde{A}_j^{-1} R_j. \tag{3.14}$$

Analogously we define the *2-level Additive Schwarz Operator* the operator

$$P_{ASO}^{(2)} = P_0 + \sum_{j=1}^{N} P_j = R_0^T \tilde{A}_0^{-1} R_0 A + \sum_{j=1}^{N} R_j^T \tilde{A}_j^{-1} R_j A. \tag{3.15}$$

and the *2-level Additive Schwarz Preconditioner* the operator

$$\left(A_{ASO}^{(2)}\right)^{-1} = R_0^T \tilde{A}_0^{-1} R_0 + \sum_{j=1}^{N} R_j^T \tilde{A}_j^{-1} R_j. \tag{3.16}$$

In order to prove lower and upper bounds respectively for the minimum and maximum eigenvalues of $P_{ASO}^{(2)}$, and hence estimates for its condition number, we need three Assumptions on the decomposition.

- **Assumption 1: Stable Decomposition.** There exists a constant $C_0 > 0$ such that every $v \in \mathcal{V}$ admits a decomposition

  $$v = \sum_{j=0}^{N} R_j^T v_j, \qquad v_j \in \mathcal{V}_j, \tag{3.17}$$

  that satisfies

  $$\sum_{j=0}^{N} \tilde{a}_j \left(v_j, v_j\right) \leq C_0^2 a(v, v). \tag{3.18}$$

  This assumption ensures a positive lower bound for the minimum eigenvalue $\lambda_{min}\left(P_{OAS}\right)$.

- **Assumption 2: Strengthened Cauchy-Schwarz Inequalities.** There exist constants $\epsilon_{i,j} \in [0,1]$ with $i,j = 1,...,N$ such that for all $v_i \in \mathcal{V}_i$ and $v_j \in \mathcal{V}_j$ we have

$$\left|a\left(R_i^T v_i, R_j^T v_j\right)\right| \leq \epsilon_{i,j} a\left(R_i^T v_i, R_i^T v_i\right)^{\frac{1}{2}} a\left(R_j^T v_j, R_j^T v_j\right)^{\frac{1}{2}}. \tag{3.19}$$

  We will denote the spectral radius of the $N \times N$ matrix $\varepsilon = [\epsilon_{i,j}]$ with $\rho(\varepsilon)$.

- **Assumption 3: Local Stability.** There exists constant $\omega > 0$ such that for all $i = 0,...,N$ we have

$$a\left(R_j^T v_j, R_j^T v_j\right) \leq \omega \tilde{a}_j(v_j, v_j), \qquad \forall v_j \in \text{range}\left(\tilde{P}_j\right) \subset \mathcal{V}_j. \tag{3.20}$$

Theorem 3.2. With Assumptions 1,2,3, we have the following bound for the condition number of the 2-level Additive Schwarz Operator:

$$\text{cond}\left(P_{ASO}^{(2)}\right) \leq C_0^2 \omega \left(\rho(\varepsilon) + 1\right). \tag{3.21}$$

For a full proof of Theorem 3.2 we refer to [33], Chapter 2.

3.2 Decomposition strategies

As usual in Isogeometric Analysis, the decomposition of the domain Ω in subdomains $\{\Omega_j\}_{j=0}^N$ is performed dimension-wise, hence it will be explained for univariate NURBS spaces and then extended via tensor-product. In this Section, we will denote with $\mathcal{V}$ a generic B-spline or NURBS function space where the polynomial degree p is kept fixed.
Let $\Xi = \{\xi_1 \leq ... \leq \xi_{n+p+1}\}$ be an open knot vector. We select a subset of the knot vector by choosing N non-repeated knots called *interface knots*:

$$\{0 = \xi_{i_0} < ... < \xi_{i_N} = 1\} \subset \Xi. \tag{3.22}$$

The partition $\left\{\hat{I}_j = (\xi_{i_j}, \xi_{i_{j+1}})\right\}_{j=1}^N$ will constitute the subdomains.
Normally B-spline basis functions have a support that spans more than one element, therefore there are different strategies for constructing the local spaces $\mathcal{V}_j$ associated to the subdomains Ω_j. All of the strategies presented here rely on the choice of the basis functions across the interface knots, and leads to different local spaces. Strategy 1 is the original strategy from [9], Strategy 2 is the strategy that has been implemented in the code, while Strategy 3 is the algebraic strategy implemented by PETSc in order to apply Overlapping Domain Decomposition techniques in their full generality.

3.2.1 Strategy 1

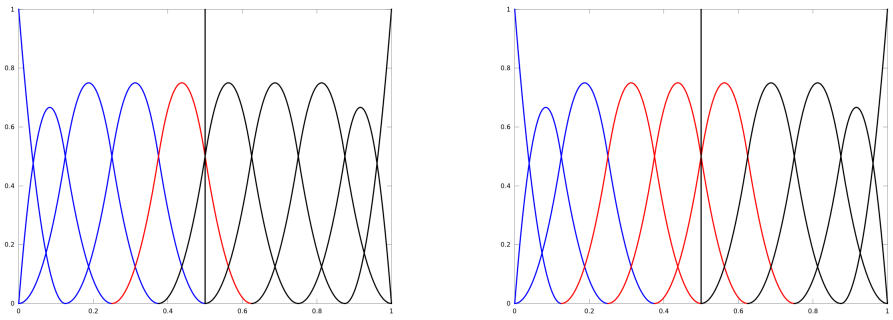
An overlap index $r \in \mathbb{N}$ is chosen to represent the number of overlapping functions among two adjacent subdomains. We then select two basis functions $\hat{B}^p_{i'}$, $\hat{B}^p_{i''}$ such that we have

$$\hat{I}_{j-1} \cap \text{supp}(\hat{B}^p_{i'}) \cap \hat{I}_j \neq \emptyset, \qquad \hat{I}_j \cap \text{supp}(\hat{B}^p_{i''}) \cap I_{j+1} \neq \emptyset.$$

Any choice of $\hat{B}^p_{i'}$ and $\hat{B}^p_{i''}$ is valid. Then we define the local subspace in the following way:

$$\mathcal{V}_j = \text{span}\left\{\hat{B}^p_i \in \mathcal{V} | i' - r \leq i \leq i'' + r\right\}.$$

We notice that there are at most $2r + 1$ overlapping functions among two adjacent subdomains.



(a) $r = 0$ (b) $r = 1$

Figure 3.1: Knot vector decomposition by Strategy 1.

Figure 3.1 shows B-spline quadratic basis functions associated with the uniform open knot vector of 8 elements, for a total of 10 basis functions. We decompose in two subdomains, hence only one interface knot has to be chosen. The chosen one is $\xi = 1/2$ so that the two subdomains are $\hat{I}_1 = (0, 1/2)$ and $\hat{I}_2 = (1/2, 1)$. In Figure 3.1a, we have $r = 0$ and we choose basis function B^2_5. We notice that B^2_6 could have been a valid choice. The two local spaces are

$$\mathcal{V}_1 = \text{span}\left\{\hat{B}^2_i \in \mathcal{V} \,|\, i = 1, ..., 5\right\}, \qquad \mathcal{V}_2 = \text{span}\left\{\hat{B}^2_i \in \mathcal{V} \,|\, i = 5, ..., 10\right\}.$$

In Figure 3.1b the overlap parameter is $r = 1$, hence there are more overlapping functions. The local spaces are

$$\mathcal{V}_1 = \text{span}\left\{\hat{B}^2_i \in \mathcal{V} \,|\, i = 1, ..., 6\right\}, \qquad \mathcal{V}_2 = \text{span}\left\{\hat{B}^2_i \in \mathcal{V} \,|\, i = 4, ..., 10\right\}.$$

Internal basis function of $\mathcal{V}_1$ and $\mathcal{V}_2$ are depicted respectively in blue and black. Overlapping basis functions are depicted in red.

3.2.2 Strategy 2

The following subspaces can be chosen:

$$\mathcal{V}_j = \text{span}\left\{\hat{B}_i^p \in \mathcal{V} \,|\, \text{supp}\left(\hat{B}_i^p\right) \cap \hat{I}_j \neq \emptyset\right\}, \qquad j = 1, ..., N.$$

If the multiplicity of the interface knot is 1, the number of overlapping functions is exactly p. The idea is selecting as overlapping functions those whose support contains the interface knot. This strategy depends on the polynomial degree, and can be referred as *generous overlap*.

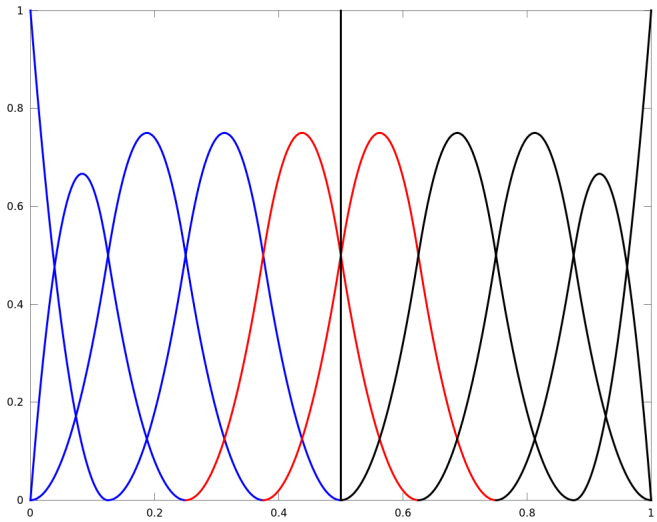


Figure 3.2: Knot vector decomposition by Strategy 2.

Figure 3.2 shows the same space $\mathcal{V}$ of Figure 3.1, with the same coloring scheme: 10 quadratic B-splines defined over a uniform open knot vector of 8 element. Again, the knot vector $(0, 1)$ is decomposed in two local subdomains, $\hat{I}_1 = (0, 1/2)$ and $\hat{I}_2 = (1/2, 1)$. Accordingly to this Strategy, the overlapping basis functions are B_5^2 and B_6^2, i.e those whose support include the interface knot $\xi = 1/2$. The two local spaces are

$$\mathcal{V}_1 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 1, ..., 6\right\}, \qquad \mathcal{V}_2 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 5, ..., 10\right\}.$$

3.2.3 Strategy 3

The last way of selecting the function subspaces is algebraic: basis functions of $\mathcal{V}$ are divided in N disjoint subsets, where in each subset basis functions are consecutive accordingly with the index order:

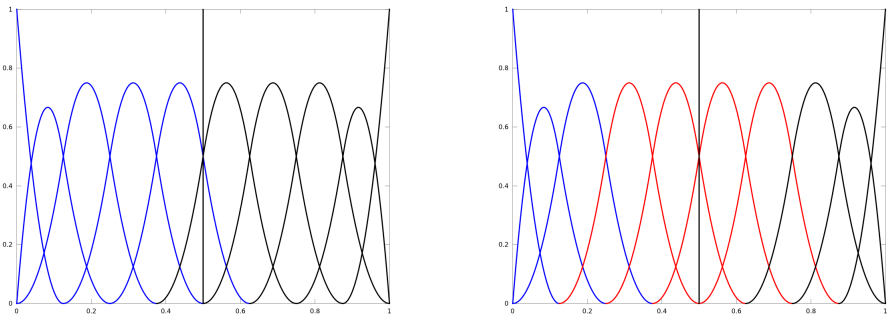
$$\left\{\hat{B}_i^p\right\}_{i=1}^{n} = \bigcup_{j=1}^{N} \left\{\hat{B}_i^p\right\}_{i=i_{j-1}}^{i_j-1}, \qquad 1 = i_0 < i_1 < \ldots < i_{N-1} < i_N = n,$$

$$\tilde{\mathcal{V}}_j = \text{span}\left\{\hat{B}_i^p | i_{j-1} \leq i < i_j\right\}.$$

This preliminary non-overlapping decomposition is the typical partitioning of degrees of freedom of the system matrix in a parallel computational environment. The j-th local subspace is defined as span of all the basis functions coupled with those of $\tilde{\mathcal{V}}_j$:

$$\mathcal{V}_j = \text{span}\left\{\hat{B}_i^p \,|\, \exists k \text{ with } i_{j-1} \leq k < i_j \text{ such that } a_{ik} \neq 0\right\}.$$

where a_{ik} is the i,j-th entry of the matrix $A = [a_{ij}]$. Due to the definition of a matrix A arising from the Petrov-Galerkin discretization of a PDE, coupled basis functions (i.e. degrees of freedom) are those whose supports have non-void intersection. This construction can be performed directly from the system matrix, and is typically used for algebraic Additive Schwarz Preconditioner. No knowledge of the PDE setting underlying the system matrix is required. Moreover, the coupling procedure can be applied recursively on $\mathcal{V}_j$ in order to obtain a more generous overlapping subspace. This is the default procedure implemented in PETSc.



(a) Non overlapping subspaces.

(b) Overlapping subspaces.

Figure 3.3: Knot vector decomposition by Strategy 3.

Figure 3.3a shows the non-overlapping subdomains $\tilde{\mathcal{V}}_1$ and $\tilde{\mathcal{V}}_2$ constructed from the usual space of quadratic B-splines on an 8-element knot vector:

$$\tilde{\mathcal{V}}_1 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 1, ..., 5\right\}, \qquad \tilde{\mathcal{V}}_2 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 6, ..., 10\right\}.$$

We add to $\tilde{\mathcal{V}}_1$ all basis functions whose support intersects the blue B-splines in order to obtain the local space:

$$\mathcal{V}_1 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 1, ..., 7\right\} = \tilde{\mathcal{V}}_1 \cup \left\{\hat{B}_6^2, \hat{B}_7^2\right\}.$$

Analogously, by adding the overlapping functions to the black ones we obtain the second local space:

$$\mathcal{V}_2 = \text{span}\left\{\hat{B}_i^2 \in \mathcal{V} \,|\, i = 4, ..., 10\right\} = \tilde{\mathcal{V}}_2 \cup \left\{\hat{B}_4^2, \hat{B}_5^2\right\}.$$

3.3 Isogeometric Decomposition setting

In this work, Strategy 2 is chosen for both implementation and theoretical background. The following definitions are aimed to fix once for all the required machinery for the adaptation of the abstract theory of Additive Schwarz Preconditioning to the Isogeometric case.
Let Ξ be a knot vector of degree p and mesh size h, $\{\xi_{i_j}\}_{j=1}^N \subset \Xi$ be interface knots and $\hat{\mathcal{V}}$ the associated B-spline space. We define the following objects:

- *Subdomains* are the open intervals $\hat{I}_j = (\xi_{j_i}, \xi_{i_{j+1}})$, for which the *mesh size* is $H \approx H_j = \text{diam}(\hat{I}_j)$.

- *Local spaces* are the spaces

$$\hat{\mathcal{V}}_j = \text{span}\left\{\hat{B}_i^p \in \mathcal{V} \,|\, \text{supp}\left(\hat{B}_i^p\right) \cap \hat{I}_j \neq \emptyset\right\}, \qquad j = 1, ..., N. \quad (3.23)$$

  For the sake of simplicity we restrict our analysis to uniform open knot vectors with the interior knots having multiplicity 1. This means that in each interior knot there are exactly p overlapping functions. If basis functions $\hat{B}_i^p, ..., \hat{B}_{i+p-1}^p$ are all and the only functions belonging to both $\hat{\mathcal{V}}_j$ and $\hat{\mathcal{V}}_{j+1}$, we define an index

$$s_j = \lfloor \frac{2i + p - 1}{2} \rfloor, \quad (3.24)$$

so that local spaces can be written as

$$\hat{\mathcal{V}}_j = \begin{cases} \text{span}\left\{\hat{B}_i^p \in \hat{\mathcal{V}} | 1 < i \leq s_2 + r\right\}, & j = 1, \\ \text{span}\left\{\hat{B}_i^p \in \hat{\mathcal{V}} | s_j - l \leq i \leq s_{j+1} + r\right\}, & 2 \leq j \leq N-1, \\ \text{span}\left\{\hat{B}_i^p \in \hat{\mathcal{V}} | s_N - l \leq i < s_{N+1}\right\}, & j = N, \end{cases} \tag{3.25}$$

with $l = \lfloor \frac{p-1}{2} \rfloor, r = \lfloor \frac{p}{2} \rfloor$, $l + r + 1 = p$. One can notices that in space $\hat{\mathcal{V}}_1$ and $\hat{\mathcal{V}}_N$ the first and last basis functions are respectively removed. This is due to the fact that those functions (or degrees of freedom) are associated with Dirichlet boundary conditions, hence they are not directly involved in the computation.

- *Extended subdomains* are the intervals

$$\hat{I}'_j = \bigcup_{\hat{B}_i^p \in \hat{\mathcal{V}}_j} \text{supp}\hat{B}_i^p = \left(\xi_{s_j - l}, \xi_{s_{j+1}+r+p+1}\right), \quad j = 1, ..., N. \tag{3.26}$$

- *Further extended subdomains* are the intervals

$$\hat{I}''_j = \bigcup_{\text{supp}\hat{B}_j^p \cap \hat{I}'_j \neq \emptyset} \text{supp}B_i^p, \quad j = 1, ..., N. \tag{3.27}$$

The decomposition of a knot vector defines not only the local spaces but the coarse space as well. The interface knots as in 3.22 are used to construct a coarse open knot vector with the same polynomial degree of the starting knot vector Ξ:

$$\Xi_0 = \left\{\xi_1 = ... = \xi_{p+1} \leq \xi_{i_1} \leq ... \leq \xi_{i_{N-1}} \leq \xi_{n_0+1} = ... = \xi_{n_0+p+1}\right\} \subset \Xi. \tag{3.28}$$

where n_0 is deduced from the construction. Given the set of control weights w, we denote in the following way the coarse space associated to Ξ_0:

$$\hat{\mathcal{V}}_0 = \hat{\mathcal{N}}\left(\Xi_0, p, w\right).$$

We notice that the coarse space $\hat{\mathcal{V}}_0$ is a proper linear subspace of $\hat{\mathcal{V}}$.
In the practice (and as implemented in the code), a very coarse mesh defines the geometry $\mathbf{F}$, a slight h-refinement defines the subdomains and the coarse space $\hat{\mathcal{V}}_0$ and finally a heavy h-refinement defines the local spaces $\hat{\mathcal{V}}_j$, so that there is no actual coarsening process.
As stated in Section 3.1, the interpolation operators R_j^T with $j = 1, ..., N$ are simple matrices of 0 and 1 that activate or deactivate the basis functions that don't belong to $\hat{\mathcal{V}}_j$. During the activation process, functions of $\hat{\mathcal{V}}_j$ are extended to 0 in $\hat{\mathcal{V}}$.

In the practice matrices R_j^T are never explicitly computed: their action is simply implemented as computing on a certain portion of the set of degrees of freedom. The interpolation operator R_0^T, instead, has to be computed explicitly in order to project from $\hat{\mathcal{V}}_0$ to $\hat{\mathcal{V}}$ (back and forth) the residual vector during the iteration of the linear solver. The coarse-to-global projection operator is computed from Algorithm 2.13:

$$R_0^T = R(\Xi_0, \Xi).$$

Finally, all concepts and mathematical entities are extended to the multivariate case via tensor product and to the physical domain via push-forward. For example, subdomains $\hat{I}_{\mathbf{j}}$ in parametric space are tensor-product of the relative intervals:

$$\hat{\Omega}_{\mathbf{j}} = \hat{I}_{\mathbf{j}} = \bigoplus_{d=1}^{D} \hat{I}_{j_d}, \qquad \mathbf{j} = 1, ..., \mathbf{N}, \tag{3.29}$$

while subdomains in physical space are the image via the geometry $\mathbf{F}$:

$$\Omega_{\mathbf{j}} = I_{\mathbf{j}} = \mathbf{F}\left(\hat{I}_{\mathbf{j}}\right), \qquad \mathbf{j} = 1, ..., \mathbf{N}.$$

We are now ready for a formal definition of the Additive Schwarz Operators and Preconditioners for the Isogeometric Analysis of the scalar elliptic equation, along with the main theoretical result of this work.

Definition 3.3. Let $\mathcal{V} = \mathcal{N}(\boldsymbol{\Xi}, \mathbf{p}, \mathbf{w})$ be a NURBS space defined over $\Omega \subset \mathbb{R}^D$ with decomposition $\{\mathcal{V}_{\mathbf{j}}\}_{\mathbf{j}=0}^{\mathbf{N}}$. Given the bilinear continuous and coercive form

$$a : \mathcal{V} \times \mathcal{V} \longrightarrow \mathbb{R}, \qquad a(v, w) = \int_\Omega \nabla v \cdot \nabla w \, dx,$$

the associated local and coarse bilinear forms

$$a_{\mathbf{j}} : \mathcal{V}_{\mathbf{j}} \times \mathcal{V}_{\mathbf{j}} \longrightarrow \mathbb{R}, \qquad a_{\mathbf{j}}(v_{\mathbf{j}}, w_{\mathbf{j}}) = \int_\Omega \nabla v_{\mathbf{j}} \cdot \nabla w_{\mathbf{j}} \, dx,$$

and relative stiffness matrices A, $\{A_{\mathbf{j}}\}_{\mathbf{j}=0}^{\mathbf{N}}$ and the interpolation operators $\{R_{\mathbf{j}}\}_{\mathbf{j}=0}^{\mathbf{N}}$, we define the *1-level Additive Schwarz Operator* the operator

$$P_{ASO}^{(1)} = \sum_{\mathbf{j}=1}^{\mathbf{N}} P_{\mathbf{j}} = \sum_{\mathbf{j}=1}^{\mathbf{N}} R_{\mathbf{j}}^T A_{\mathbf{j}}^{-1} R_{\mathbf{j}} A. \tag{3.30}$$

We define *1-level Additive Schwarz Preconditioner* the operator

$$\left(A_{ASO}^{(1)}\right)^{-1} = \sum_{\mathbf{j}=1}^{\mathbf{N}} R_{\mathbf{j}}^T A_{\mathbf{j}}^{-1} R_{\mathbf{j}}. \tag{3.31}$$

Analogously we define the *2-level Additive Schwarz Operator* the operator

$$P_{ASO}^{(2)} = P_0 + \sum_{\mathbf{j}=1}^{\mathbf{N}} P_{\mathbf{j}} = R_0^T A_0^{-1} R_0 A + \sum_{\mathbf{j}=1}^{\mathbf{N}} R_{\mathbf{j}}^T A_{\mathbf{j}}^{-1} R_{\mathbf{j}} A. \tag{3.32}$$

and the *2-level Additive Schwarz Preconditioner* the operator

$$\left(A_{ASO}^{(2)}\right)^{-1} = R_0^T A_0^{-1} R_0 + \sum_{\mathbf{j}=1}^{\mathbf{N}} R_{\mathbf{j}}^T A_{\mathbf{j}}^{-1} R_{\mathbf{j}}. \tag{3.33}$$

Theorem 3.4. The condition number of the 2-level Additive Schwarz Operator 3.32 for the Isogeometric Analysis of the scalar elliptic equation is bounded by

$$\operatorname{cond}\left(P_{ASO}^{(2)}\right) \leq C\left(1 + \frac{H}{h}\right), \tag{3.34}$$

where constant $C > 0$ is independent of $H, h, \mathbf{N}$, but not of $\mathbf{p}$.

Proof. See Section 3.4. □

3.4 Stable splitting

In order to prove the bound of Theorem 3.4, the three Assumptions in Section 3.1 must be satisfied.
Assumption 2 ensures an upper bound for the maximum eigenvalue $\lambda_{max}(P_{ASO})$. With a standard coloring argument, the spectral radius $\rho(\varepsilon)$ is bounded by the number of colors. In case of tensor product Cartesian grid, the number of colors is 2^D, hence $\rho(\varepsilon) \leq 2^D$.
Assumption 3 is a generic requirement that is trivially satisfied with use of exact solvers. One can verify that the definition of exact solver 3.7 satisfies 3.20 with $\omega = 1$.
Assumption 1 has to be specifically proven for IGA, and it is a nontrivial task. It relies on the definition of interpolation operators between $\hat{\mathcal{V}}$ and $\hat{\mathcal{V}}_j$. These operators will be firstly presented only for the univariate case and for a decomposition of two subdomains. Multivariate tensor product structure and the fact that the difficulties arise on the interface of the two subdomains guarantee an almost trivial extension to more general cases.
As introduced in Chapter 1, the scalar elliptic equation treated in this work is defined as follow:

$$-\nabla \cdot (\mathbf{K}\nabla u) = f \quad \text{in } \Omega.$$

We assume that the diffusivity tensor $\mathbf{K} : \Omega \longrightarrow \mathbb{R}^D \times \mathbb{R}^D$ is of the form $\mathbf{K} = \kappa \mathbf{Id}$, with $\kappa = \kappa(x) \in L^\infty(\Omega)$. Thus the coefficient κ is bounded from both above and below:

$$\kappa_{min} \leq \kappa(x) \leq \kappa_{max}, \qquad \forall x \in \Omega,\ \kappa_{min}, \kappa_{max} \in \mathbb{R}.$$

For the proof of Assumption 1 we assume $\kappa \equiv 1$ in Ω, so that the scalar elliptic equation can be rewritten in the following way:

$$-\Delta u = f \ \text{ in } \Omega. \tag{3.35}$$

The more general case can be proved straightforwardly with the addition of the term $\kappa_{max}/\kappa_{min}$. As a standard procedure in calculus, equation 3.35 is written in variational form and the Laplace operator is seen as an elliptic bilinear form:

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx = F(v).$$

Noticing that $a(v, v) = \int_\Omega |\nabla v|^2 \, dx = \|\nabla v\|^2_{L^2(\Omega)}$, Assumption 1 in Section 3.1 will be proved as a bound on the H^1-seminorms of the splitting $v = v_0 + \sum_{j=1}^N v_j$:

$$\|\nabla v_0\|^2_{L^2(\Omega)} + \sum_{j=1}^N \|\nabla v_j\|^2_{L^2(\Omega)} \leq C \left(1 + \frac{H}{h}\right) \|\nabla v\|^2_{L^2(\Omega)}.$$

Once the latter is established, Theorem 3.4 is proved as a particular case of the abstract Theorem 3.2.
In what follows, we denote with $a \lesssim b$ any inequality of the type $a \leq Cb$ with C being a generic constant eventually independent from underlying knot vectors, thus independent from mesh sizes h, H and the number of subdomains.

3.4.1 Univariate stable splitting

Definition 3.5. Given an univariate B-spline space $\hat{\mathcal{V}} = \hat{\mathcal{B}}(\Xi, p)$, the decomposition $\hat{\Omega} = \overline{\hat{I}_1 \cap \hat{I}_2} = \overline{(0, \xi_{i_2}) \cap (\xi_{i_2}, 1)}$ and the subsequent local spaces

$$\hat{\mathcal{V}}_1 = \text{span}\left\{\hat{B}^p_i | 2 \leq i \leq s_2 + r\right\}, \qquad \hat{\mathcal{V}}_2 = \text{span}\left\{\hat{B}^p_i | s_2 - l \leq i \leq n - 1\right\}.$$

as in Strategy 2 in Section 3.2.2, then for $v = \sum c_i \hat{B}^p_i \in \hat{\mathcal{V}}$ we define the two interpolation operators $\hat{\Pi}_j : \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}}_j$ in the following way:

$$\hat{\Pi}_1 v = \sum_{i=2}^{s_2-l-1} c_i \hat{B}^p_i + \sum_{i=s_2-l}^{s_2+r} \frac{s_2 + r + 1 - i}{p+1} c_i \hat{B}^p_i, \tag{3.36}$$

$$\hat{\Pi}_2 v = \sum_{i=s_2-l}^{s_2+r} \frac{l - s_2 + 1 + i}{p+1} c_i \hat{B}^p_i + \sum_{i=s_2+r+1}^{n-1} c_i \hat{B}^p_i. \tag{3.37}$$

The action of the interpolation operators is the restriction of a function in $\hat{\mathcal{V}}$ to $\hat{\mathcal{V}}_j$, with coefficients of the overlapping functions linearly decreasing from 1 to 0 outward the subdomain. It is easy to see that

$$v = \hat{\Pi}_1 v + \hat{\Pi}_2 v.$$

For brevity, we rewrite the operators with $\hat{\Pi}_1 v = \sum_{i=2}^{s_2+r} \bar{c}_i B_i^p$, where

$$\bar{c}_i = c_i d_i, \qquad d_i = \begin{cases} 1 & 2 \le i \le s_2 - l - 1 \\ \frac{s_2+r+1-i}{p+1} & s_2 - l \le i \le n-1. \end{cases} \tag{3.38}$$

Lemma 3.6. For all $z \in \hat{\mathcal{V}}$ the operators $\hat{\Pi}_j$, with $j = 1, 2$, satisfy

$$\left\| \frac{d}{d\xi} \hat{\Pi}_j z \right\|^2_{L^2(\hat{I}'_j)} \lesssim \left(1 + \frac{H}{h}\right) \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}''_j)} + \frac{1}{hH} \|z\|^2_{L^2(\hat{I}''_j)}, \tag{3.39}$$

$$\left\| \hat{\Pi}_j z \right\|^2_{L^2(\hat{I}'_j)} \lesssim \|z\|^2_{L^2(\hat{I}''_j)}, \tag{3.40}$$

where $\hat{I}'_j$ and $\hat{I}''_j$ are as in 3.26 and 3.27 respectively.

Proof. We will prove the result for $\hat{\Pi}_1$, the case of $\hat{\Pi}_2$ being analogous. First, we recall the B-spline derivative formula 2.4:

$$\frac{d}{d\xi} \hat{B}_i^p(\xi) = p \left(\frac{\hat{B}_i^{p-1}(\xi)}{\Delta_i^p} - \frac{\hat{B}_{i+1}^{p-1}(\xi)}{\Delta_{i+1}^p} \right).$$

A telescoping sum follows immediately:

$$\begin{aligned} \frac{d}{d\xi} \hat{\Pi}_1 z = \sum_{i=2}^{s_2+r} \bar{c}_i \frac{d}{d\xi} \hat{B}_i^p = \sum_{i=2}^{s_2+r} \bar{c}_i p \left(\frac{\hat{B}_i^{p-1}}{\Delta_i^p} - \frac{\hat{B}_{i+1}^{p-1}}{\Delta_{i+1}^p} \right) = \\ = p \sum_{i=2}^{s_2+r+1} (\bar{c}_i - \bar{c}_{i-1}) \frac{\hat{B}_i^{p-1}}{\Delta_i^p}. \end{aligned} \tag{3.41}$$

with the convention that $\bar{c}_1 = \bar{c}_{s_2+r+1} = 0$. We now consider that

$$\bar{c}_i - \bar{c}_{i-1} = c_i d_i - c_{i-1} d_{i-1} = c_i (d_i - d_{i-1}) + d_{i-1} (c_i - c_{i-1}),$$

and substituting in 3.41 we obtain

$$\begin{aligned} \frac{d}{d\xi} \hat{\Pi}_1 z = \left(p \sum_{i=2}^{s_2+r+1} (c_i (d_i - d_{i-1})) \frac{\hat{B}_i^{p-1}}{\Delta_i^p} \right) + \\ + \left(p \sum_{i=2}^{s_2+r+1} (d_{i-1} (c_i - c_{i-1})) \frac{\hat{B}_i^{p-1}}{\Delta_i^p} \right) = T_1 + T_2. \end{aligned} \tag{3.42}$$

Terms T_1 and T_2 will be estimated separately. Starting with T_1, we now notice that

$$d_i - d_{i-1} = \begin{cases} 0 & 2 \le i \le s_2 - l - 1, \\ -\theta & s_2 - l \le i \le s_2 + r, \end{cases} \qquad \theta = \frac{1}{p+1}.$$

Due to the mesh regularity, we have that $\Delta_i^p \ge hp$ for any i, hence

$$|T_1| = \left| p \sum_{i=s_2-l}^{s_2+r+1} (-c_i)\theta \frac{\hat{B}_i^{p-1}}{\Delta_i^p} \right| \le \frac{1}{h} \sum_{i=s_2-l}^{s_2+r+1} \theta |c_i| \hat{B}_i^{p-1}.$$

Dual basis functionals as in 2.5 are such that $\lambda_i^p[z] = c_i$, therefore with $q = +\infty$ we get

$$|c_i| = |\lambda_i^p[z]| \lesssim \|z\|_{L^\infty(\xi_i, \xi_{i+p+1})} \lesssim \|z\|_{L^\infty(\mathrm{supp} T_1)}.$$

We hence obtain

$$|T_1| \le \frac{1}{h} \sum_{i=s_2-l}^{s_2+r+1} \theta |c_i| \hat{B}_i^{p-1} \le \frac{\theta}{h} \max_{i=s_2-l}^{s_2+r+1} |c_i| \sum_{i=s_2-l}^{s_2+r+1} \hat{B}_i^{p-1} \lesssim \frac{1}{h} \|z\|_{L^\infty(\mathrm{supp} T_1)}.$$

We now notice that

$$\mathrm{supp}(T_1) = \bigcup_{i=s_2-l}^{s_2+r+1} \mathrm{supp}(\hat{B}_i^{p-1}) \quad \Longrightarrow \quad \mathrm{supp}(T_1) = (\xi_{s_2-l}, \xi_{s_2+r+p+1}) \subset \hat{I}_1'.$$

thus, squaring both sides and integrating over $\hat{I}_1'$ we get

$$\int_{\hat{I}_1'} |T_1(\xi)|^2 \, d\xi = \int_{\mathrm{supp}(T_1)} |T_1(\xi)|^2 \, dx \lesssim \frac{1}{h^2} |\xi_{s_2+r+p+1} - \xi_{s_2-l}| \|z\|^2_{L^\infty(\mathrm{supp} T_1)}. \tag{3.43}$$

We notice that the extrema of the interval $(\xi_{s_2-l}, \xi_{s_2+r+p+1})$ are knot whose indexes differ by the following quantity:

$$s_2 + r + p + 1 - (s_2 - l) = l + r + 1 + p = 2p, \tag{3.44}$$

thus $|\xi_{s_2+r+p+1} - \xi_{s_2-l}| \lesssim h$ due to mesh regularity assumptions. Plugging it in 3.43 we have

$$\int_{\hat{I}_1'} |T_1(\xi)| \, d\xi \lesssim \frac{1}{h} \|z\|^2_{L^\infty(\mathrm{supp} T_1)} \lesssim \frac{1}{h} \|z\|^2_{L^\infty(\hat{I}_1'')}.$$

With a standard scaling argument from the one-dimensional $H^1 \subset L^\infty$ Sobolev embedding, we get

$$\int_{\hat{I}_1'} |T_1(\xi)| \, d\xi \lesssim \frac{H}{h} \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1'')} + \frac{1}{Hh} \|z\|^2_{L^2(\hat{I}_1'')}. \tag{3.45}$$

We now prove a bound for the second term T_2. Again, using the dual basis function bound, for every $\phi = \sum_{i=2}^{n-1} \beta_i \hat{B}_i^{p-1}$ and for $q = 2$ we have

$$|\beta_i| = |\lambda_i^{p-1}[\phi]| \lesssim h^{-\frac{1}{2}} \|z\|_{L^2(\xi_i, \xi_{i+p})}.$$

For $z = \sum_{i=2}^{n-1} c_i \hat{B}_i^p$ we recall the derivative telescoping formula:

$$\frac{d}{d\xi} z = \sum_{i=2}^{n} p \frac{c_i - c_{i-1}}{\Delta_i^p} \hat{B}_i^{p-1},$$

and combining the two we get that for any i

$$p \frac{|c_i - c_{i-1}|}{\Delta_i^p} \lesssim |\xi_{i+p} - \xi_i|^{-\frac{1}{2}} \left\| \frac{d}{d\xi} z \right\|_{L^2(\xi_i, \xi_{i+p})} \lesssim \frac{1}{\sqrt{h}} \left\| \frac{d}{d\xi} z \right\|_{L^2(\xi_i, \xi_{i+p})},$$

with the usual convention $c_1 = c_n = 0$. We now rewrite T_2 using the definition of d_i as in 3.38:

$$T_2 = p \sum_{i=2}^{s_2 - l} (c_i - c_{i-1}) \frac{\hat{B}_i^{p-1}}{\Delta_i^p} + p \sum_{i=s_2-l+1}^{s_2+r+1} d_i (c_i - c_{i-1})) \frac{\hat{B}_i^{p-1}}{\Delta_i^p}.$$

We notice that if we restrict T_2 to the interval $(0, \xi_{s_2-l+1})$, then T_2 coincides with $\frac{d}{d\xi} z$, hence

$$\begin{aligned} \int_{\hat{I}_1'} |T_2(\xi)|^2 \, d\xi &= \int_0^{\xi_{s_2-l+1}} |T_2(\xi)|^2 \, d\xi + \int_{\xi_{s_2-l+1}}^{\xi_{s_2+r+p+1}} |T_2(\xi)|^2 \, d\xi = \\ &= \int_0^{\xi_{s_2-l+1}} \left| \frac{d}{d\xi} z(\xi) \right|^2 \, d\xi + \int_{\xi_{s_2-l+1}}^{\xi_{s_2+r+p+1}} |T_2(\xi)|^2 \, d\xi. \end{aligned}$$

Moreover, using the definition of T_2, the B-spline partition of unity property, the bound of the B-spline functions coefficients, and the fact that $d_i \leq 1$, we get for $\xi \in \text{supp} T_2$

$$\begin{aligned} |T_2(\xi)| &\leq \sum_{i=2}^{s_2+r+1} p \frac{|c_i - c_{i-1}|}{\Delta_i^p} \hat{B}_i^{p-1} \leq \\ &\leq \max_{i=2}^{s_2+r+1} \left\{ p \frac{|c_i - c_{i-1}|}{\Delta_i^p} \right\} \sum_{i=2}^{s_2+r+1} B_i^{p-1} \lesssim \\ &\lesssim \frac{1}{\sqrt{h}} \left\| \frac{d}{d\xi} z \right\|_{L^2(\xi_i, \xi_{i+p})}. \end{aligned}$$

We are now ready to conclude the proof of the bound for T_2:

$$\int_{\hat{I}_1'} |T_2(\xi)|^2 \, d\xi = \int_0^{\xi_{s_2-l+1}} \left| \frac{d}{d\xi} z(\xi) \right|^2 \, d\xi + \int_{\xi_{s_2-l+1}}^{\xi_{s_2+r+p+1}} |T_2(\xi)|^2 \, d\xi \lesssim$$
$$\lesssim \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1')} + \frac{1}{h} \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\xi_i, \xi_{i+p})} \int_{\xi_{s_2-l+1}}^{\xi_{s_2+r+p+1}} d\xi \lesssim$$
$$\lesssim \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1'')}.$$

using the fact that $|\xi_{s_2+r+p+1} - \xi_{s_2-l+1}| \lesssim h$ as stated in 3.44.
Combining the bounds on T_1 and T_2, we finally obtain the first part of the Lemma:

$$\left\| \frac{d}{d\xi} \hat{\Pi}^1 z \right\|^2_{L^2(\hat{I}_1')} = \|T_1 + T_2\|^2_{L^2(\hat{I}_1')} = \int_{\hat{I}_1'} |T_1(\xi) + T_2(\xi)|^2 \, d\xi \leq$$
$$\leq \int_{\hat{I}_1'} 2(|T_1(\xi)|^2 + |T_2(\xi)|^2) \, d\xi = 2\|T_1\|^2_{L^2(\hat{I}_1')} + 2\|T_2\|^2_{L^2(\hat{I}_1')} \lesssim$$
$$\lesssim \frac{H}{h} \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1'')} + \frac{1}{hH} \|z\|^2_{L^2(\hat{I}_1'')} + \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1'')} \lesssim$$
$$\lesssim \left(1 + \frac{H}{h} \right) \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}_1'')} + \frac{1}{hH} \|z\|^2_{L^2(\hat{I}_1'')}.$$

For the second part of the Lemma, using the bound on the dual basis functionals with $q = 2$ and noticing that $|\xi_{m+p+1} - \xi_{m-p}| \geq 2hp$, we get for $\xi \in E = (\xi_m, \xi_{m+1}) \subset \hat{I}_1'$

$$\left| \hat{\Pi}_1 z(\xi) \right| = \left| \sum_{i=m-p}^{m} \bar{c}_i \hat{B}_i^p(\xi) \right| \leq \max_{i=m-p}^{m} |c_i| \lesssim \frac{1}{\sqrt{h}} \|z\|_{L^2(\xi_{m-p}, \xi_{m+p+1})},$$

so that, finally

$$\left\| \hat{\Pi}_1 z \right\|^2_{L^2(\hat{I}_1')} = \sum_{E \subset \hat{I}_1'} \int_e \left| \hat{\Pi}_1 z(\xi) \right|^2 \, d\xi \lesssim \frac{1}{h} \sum_{E \subset \hat{I}_1'} \|z\|^2_{L^2(\xi_{m-p}, \xi_{m+p+1})} \int_E d\xi \lesssim$$
$$\lesssim \frac{1}{h} h \|z\|^2_{L^2(\hat{I}_1'')} = \|z\|^2_{L^2(\hat{I}_1'')}.$$

□

This Lemma is easily extended to the case of $N > 2$ subdomains. The interpolation operators associated to inner subdomains will present linearly scaled coefficients in both extrema of the subdomain, while the leftmost and the rightmost

ones are similar to those in Definition 3.5. If $z = \sum_{i=2}^{n-1} c_i \hat{B}_i^p$ we have:

$$\hat{\Pi}_j : \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}}_j, \qquad j = 1, ..., N,$$

$$\begin{aligned}
\hat{\Pi}_1 z &= \sum_{i=2}^{s_2-l-1} c_i \hat{B}_i^p + \sum_{i=s_2-l}^{s_2+r} \frac{s_2 + r + 1 - i}{p+1} c_i \hat{B}_i^p, \\
\hat{\Pi}_j z &= \sum_{i=s_j-l}^{s_j+r} \frac{s_j + r + 1 - i}{p+1} c_i \hat{B}_i^p + \sum_{i=s_j+r+1}^{s_{j+1}-l-1} c_i \hat{B}_i^p + \\
&\qquad + \sum_{i=s_{j+1}-l}^{s_{j+1}+r} \frac{l - s_{j+1} + 1 + i}{p+1} c_i \hat{B}_i^p, \\
\hat{\Pi}_N z &= \sum_{i=s_N-l}^{s_N+r} \frac{s_N + r + 1 - i}{p+1} c_i \hat{B}_i^p + \sum_{i=s_N+r+1}^{n-1} c_i \hat{B}_i^p.
\end{aligned} \tag{3.46}$$

Once again, we have that

$$z = \sum_{j=1}^{N} \hat{\Pi}_j z. \tag{3.47}$$

We thus have the following result:

Lemma 3.7. The operators $\hat{\Pi}_j$ with $j = 1, ..., N$ defined as in 3.46 satisfy for all $z \in \hat{\mathcal{V}}$ the following bounds:

$$\left\| \frac{d}{d\xi} \hat{\Pi}_j z \right\|^2_{L^2(\hat{I}'_j)} \lesssim \left(1 + \frac{H}{h}\right) \left\| \frac{d}{d\xi} z \right\|^2_{L^2(\hat{I}''_j)} + \frac{1}{hH} \|z\|^2_{L^2(\hat{I}''_j)}, \tag{3.48}$$

$$\left\| \hat{\Pi}_j z \right\|^2_{L^2(\hat{I}'_j)} \lesssim \|z\|^2_{L^2(\hat{I}''_j)}. \tag{3.49}$$

3.4.2 Multivariate stable splitting

We now extend Lemma 3.7 to the multivariate case. Let $\boldsymbol{\Xi}$ be a set of D knot vectors, $\mathbf{p}$ the polynomial degrees and $\hat{\mathcal{V}} = \hat{\mathcal{B}}(\boldsymbol{\Xi}, \mathbf{p})$ the associated multivariate B-spline space. We denote the univariate restriction of a function $v = v(\xi_1, ..., \xi_D) \in \hat{\mathcal{B}}$ in the following way:

$$v_{\bar{\xi},d} : (0,1) \longrightarrow \mathbb{R}, \qquad \bar{\xi} = (\xi_1, ..., \xi_{d-1}, \xi_{d+1}, ..., \xi_D) \in \mathbb{R}^{D-1},$$

$$v_{\bar{\xi},d}(\xi) = v(\bar{\xi}_1, ..., \bar{\xi}_{d-1}, \xi, \bar{\xi}_{d+1}, ..., \bar{\xi}_D).$$

If the domain Ω is decomposed in $\mathbf{N} = (N_1, ..., N_D)$ subdomains $\Omega_{\mathbf{j}}$ and thus $\hat{\mathcal{V}}$ is decomposed in local spaces $\hat{\mathcal{V}}_{\mathbf{j}}$, we introduce the D univariate interpolation operators:

$$\hat{\Pi}^d_{j_d} : \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}}, \qquad j_d = 1, ..., N_d, \ d = 1, .., D,$$
$$\hat{\Pi}^d_{j_d} v(\xi_1, ..., \xi_D) = \hat{\Pi}_{j_d} v_{\bar{\xi},d}(\xi_{j_d}), \tag{3.50}$$

where $\hat{\Pi}_{j_d}$ is the interpolation operator defined in 3.46, acting on the univariate function $v_{\bar{\xi},d}$. It is easy to check that these operators commute with respect to composition and differentiation: for all dimensions $d_1, d_2 = 1, ..., D$ and indexes j_{d_1} and j_{d_2} we have

$$\hat{\Pi}^{d_1}_{j_{d_1}} \circ \hat{\Pi}^{d_2}_{j_{d_2}} = \hat{\Pi}^{d_2}_{j_{d_2}} \circ \hat{\Pi}^{d_1}_{j_{d_1}}, \qquad \frac{\partial}{\partial \xi_{d_1}} \hat{\Pi}^{d_2}_{j_{d_2}} = \hat{\Pi}^{d_2}_{j_{d_2}} \frac{\partial}{\partial \xi_{d_1}}.$$

We can now define the multivariate interpolation operators:

Definition 3.8. Given a multivariate B-spline space $\hat{\mathcal{V}}$ and a decomposition in local spaces $\hat{\mathcal{V}}_{\mathbf{j}}$ with $\mathbf{j} = 1, ..., \mathbf{N}$, we define *multivariate interpolation operators* the operators

$$\hat{\Pi}_{\mathbf{j}} = \hat{\Pi}_{j_1,...,j_D} : \hat{\mathcal{V}} \longrightarrow \hat{\mathcal{V}}_{\mathbf{j}}, \qquad \hat{\Pi}_{\mathbf{j}} = \hat{\Pi}^1_{j_1} \circ ... \circ \hat{\Pi}^D_{j_D}.$$

where $\hat{\Pi}^d_{j_d}$ are defined as in 3.50.

Let $\hat{\Pi}_0$ be the standard spline quasi-interpolant into the coarse global space $\hat{\mathcal{V}}_0 = \hat{\mathcal{B}}(\Xi_0, p)$ (see [31], Theorem 12.6 and 12.7). Given a function $u \in \hat{\mathcal{V}}$ we denote

$$u_0 = \hat{\Pi}_0 u \in \hat{\mathcal{V}}_0, \qquad z = u - u_0 \in \hat{\mathcal{V}},$$

for which the following bound holds:

$$\|z\|^2_{L^2(\hat{\Omega})} + H^2 \|\nabla u_0\|^2_{L^2(\hat{\Omega})} \lesssim H^2 \|\nabla u\|^2_{L^2(\hat{\Omega})}. \tag{3.51}$$

Using the linearity of the interpolant operators 3.46 and the splitting property 3.47, the generic function $u \in \hat{\mathcal{V}}$ admits the following splitting:

$$u = u_0 + \sum_{\mathbf{j}=1}^{\mathbf{N}} u_{\mathbf{j}} = u_0 + \sum_{j_1=1}^{N_1} ... \sum_{j_D=1}^{N_D} u_{j_1,...,j_D}, \tag{3.52}$$

where

$$u_0 = \hat{\Pi}_0 u \in \hat{\mathcal{V}}_0, \qquad u_{\mathbf{j}} = \hat{\Pi}_{\mathbf{j}} u \in \hat{\mathcal{V}}_{\mathbf{j}}. \tag{3.53}$$

Theorem 3.9. For all $v \in \hat{\mathcal{V}}$ it holds

$$\|\nabla u_0\|^2_{L^2(\hat{\Omega})} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\nabla u_{\mathbf{j}}\|^2_{L^2(\hat{\Omega})} \lesssim \left(1 + \frac{H}{h}\right) \|\nabla u\|^2_{L^2(\hat{\Omega})}. \tag{3.54}$$

Proof. We will prove the bound for $D = 2$, the generic case $D > 2$ being analogous. From property 3.51 it immediately follows that

$$\|\nabla u_0\|^2_{L^2(\hat{\Omega})} \lesssim \|\nabla u\|_{L^2(\hat{\Omega})}. \tag{3.55}$$

For the second term, without loss of generality we treat derivatives in the first dimension, i.e. with respect to ξ_1. We fix a multiindex $\mathbf{j} = (j_1, j_2)$. By definition we have that

$$\mathrm{supp} u_{\mathbf{j}} = \hat{\Omega}'_j = \bigotimes_{\mathbf{j}=1}^{2} \hat{I}'_{\mathbf{j}} = \hat{I}'_{j_1} \times \hat{I}'_{j_2},$$

so that with the definition of the interpolation operators 3.46 and the commutativity properties we obtain for $u - u_0 = z$

$$\left\| \frac{\partial}{\partial \xi_1} u_{j_1 j_2} \right\|^2_{L^2(\hat{\Omega})} = \left\| \frac{\partial}{\partial \xi_1} \hat{\Pi}^1_{j_1} \hat{\Pi}^2_{j_2} z \right\|^2_{L^2(\hat{\Omega}_{j_1 j_2})} = \left\| \hat{\Pi}^2_{j_2} \frac{\partial}{\partial \xi_1} \hat{\Pi}^1_{j_1} z \right\|^2_{L^2(\hat{\Omega}_{j_1 j_2})}, \tag{3.56}$$

where $\hat{\Omega}_{j_1 j_2}$ are defined as in 3.29. From the bound 3.49 we obtain for $v \in \hat{\mathcal{V}}$

$$\begin{aligned}
\left\| \hat{\Pi}^2_{j_2} v \right\|^2_{L^2(\hat{\Omega}_{j_1 j_2})} &= \int_{\hat{I}'_{j_1}} \int_{\hat{I}'_{j_2}} \left| \hat{\Pi}^2_{j_2} v(\xi_1, \xi_2) \right|^2 d\xi_2 d\xi_1 = \\
&= \int_{\hat{I}'_{j_1}} \left(\int_{\hat{I}'_{j_2}} \left| \hat{\Pi}_{j_2} v_{\bar{\xi},2}(\xi_2) \right|^2 d\xi_2 \right) d\xi_1 \lesssim \\
&\lesssim \int_{\hat{I}'_{j_1}} \left(\int_{\hat{I}''_{j_2}} \left| v_{\bar{\xi},2} v(\xi_2) \right|^2 d\xi_2 \right) d\xi_1 = \\
&= \int_{\hat{I}'_{j_1}} \int_{\hat{I}''_{j_2}} |v(\xi_1, \xi_2)|^2 \, d\xi_2 d\xi_1.
\end{aligned} \tag{3.57}$$

Combining 3.56, 3.57 and bound 3.48 from Lemma 3.7 we obtain

$$\begin{aligned}
\left\| \frac{\partial}{\partial \xi_1} u_{j_1 j_2} \right\|^2_{L^2(\hat{\Omega})} &\lesssim \int_{\hat{I}'_{j_i}} \int_{\hat{I}''_{j_2}} \left| \frac{\partial}{\partial \xi_1} u_{j_1 j_2}(\xi_1, \xi_2) \right|^2 d\xi_2 d\xi_1 = \\
&= \int_{\hat{I}'_{j_i}} \int_{\hat{I}''_{j_2}} \left| \frac{\partial}{\partial \xi_1} \hat{\Pi}^1_{j_1} z(\xi_1, \xi_2) \right|^2 d\xi_2 d\xi_1 = \\
&= \int_{\hat{I}''_{j_2}} \int_{\hat{I}'_{j_1}} \left| \frac{\partial}{\partial \xi_1} \hat{\Pi}_{j_1} z_{\bar{\xi},1}(\xi_1) \right|^2 d\xi_1 d\xi_2 \lesssim \\
&\lesssim \left(1 + \frac{H}{h}\right) \int_{\hat{I}''_{j_2}} \int_{\hat{I}'_{j_1}} \left| \frac{\partial}{\partial \xi_1} z_{\bar{\xi},1}(\xi_1) \right|^2 d\xi_1 d\xi_2 + \\
&\qquad + \frac{1}{hH} \int_{\hat{I}''_{j_2}} \int_{\hat{I}'_{j_1}} \left| z_{\bar{\xi},1}(\xi_1) \right|^2 d\xi_1 d\xi_2 = \\
&= \left(1 + \frac{H}{h}\right) \left\| \frac{\partial}{\partial \xi_1} z \right\|^2_{L^2(\hat{\Omega}''_{j_1 j_2})} + \frac{1}{hH} \|z\|^2_{L^2(\hat{\Omega}''_{j_1 j_2})},
\end{aligned} \tag{3.58}$$

where, with the usual notational consistency, $\hat{\Omega}''_{j_1 j_2} = \hat{I}''_{j_1} \times \hat{I}''_{j_2}$. Due to the tensor structure, the number of overlaps of further extended subdomains is uniformly bounded by 2^D, hence by a standard coloring argument a global summation on inequality 3.58 gives

$$\sum_{j_1=1}^{N_1} \sum_{j_2=1}^{N_2} \left\| \frac{\partial}{\partial \xi_1} u_{j_1 j_2} \right\|^2_{L^2(\hat{\Omega})} \lesssim \left(1 + \frac{H}{h}\right) \left\| \frac{\partial}{\partial \xi_1} z \right\|^2_{L^2(\hat{\Omega})} + \frac{1}{hH} \|z\|^2_{L^2(\hat{\Omega})}. \tag{3.59}$$

Using property 3.51 and some simple algebra, from 3.59 we finally get

$$\sum_{j_1=1}^{N_1} \sum_{j_2=1}^{N_2} \left\| \frac{\partial}{\partial \xi_1} u_{j_1 j_2} \right\|^2_{L^2(\hat{\Omega})} \lesssim \left(1 + \frac{H}{h}\right) \|\nabla u\|^2_{L^2(\hat{\Omega})}.$$

Combining the latter with 3.55 we obtain the thesis. □

3.4.3 Stable splitting in physical domain

The stable splitting is proved for B-spline functions in parametric domain. It will now be presented in its full generality, i.e. for NURBS functions in parametric domain. Let $\mathbf{F}$ be the geometric map that defines the domain Ω and W the weight function as in 2.16. It is easy to check that the NURBS space in physical domain $\mathcal{V} = \mathcal{N}(\boldsymbol{\Xi}, \mathbf{p}, \mathbf{w})$ can be seen in the following way:

$$\mathcal{V} = \left\{ \frac{\hat{v}}{W} \circ \mathbf{F}^{-1} \,\middle|\, \hat{v} \in \hat{\mathcal{V}} = \hat{\mathcal{B}}(\boldsymbol{\Xi}, \mathbf{p}) \right\}.$$

Thus, there exists a bijective correspondence between $\mathcal{V}$ and $\hat{\mathcal{V}}$ such that for a function $u \in \mathcal{V}$

$$\frac{\hat{u}}{W} \circ \mathbf{F}^{-1} = u \longleftrightarrow \hat{u} = W(u \circ \mathbf{F}).$$

The bijection permits to transport the splitting 3.52 from the parametric space to the physical space:

$$u_{\mathbf{j}} = \frac{\hat{u}_{\mathbf{j}}}{W} \circ \mathbf{F}^{-1} \in \mathcal{V}_{\mathbf{j}}, \qquad u_0 = \frac{\hat{u}_0}{W} \circ \mathbf{F}^{-1} \in \mathcal{V}_0.$$

so that the function u in physical space $\mathcal{V}$ can be written in the following way:

$$u = u_0 + \sum_{\mathbf{j}=1}^{\mathbf{N}} u_{\mathbf{j}}. \tag{3.60}$$

Theorem 3.10. For all $u \in \mathcal{V}$ it holds

$$\|\nabla u_0\|^2_{L^2(\Omega)} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\nabla u_{\mathbf{j}}\|^2_{L^2(\Omega)} \lesssim \left(1 + \frac{H}{h}\right) \|\nabla u\|^2_{L^2(\Omega)}. \tag{3.61}$$

Proof. As a standard assumption is Isogeometric Analysis (see [7]), functions $\mathbf{F}$ and w are continuous, piecewise regular and fixed at the coarsest level of mesh discretization. Furthermore the inverse function $\mathbf{F}^{-1}$ is well defined and with bounded derivatives. Hence, the three functions w, $\mathbf{F}$ and $\mathbf{F}^{-1}$ belongs (component-wise) to the Sobolev space $W^{1,\infty}(\Omega)$. With these assumption, a change of variable from $\hat{\Omega}$ to Ω and a Poincaré inequality on $\hat{\Omega}$ give

$$\begin{aligned} \|\nabla u_0\|^2_{L^2(\Omega)} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\nabla u_{\mathbf{j}}\|^2_{L^2(\Omega)} &\lesssim \|\nabla \hat{u}_0\|^2_{H^1(\hat{\Omega})} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\hat{u}_{\mathbf{j}}\|^2_{H^1(\hat{\Omega})} \lesssim \\ &\lesssim \|\nabla \hat{u}_0\|^2_{L^2(\hat{\Omega})} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\nabla \hat{u}_{\mathbf{j}}\|^2_{L^2(\hat{\Omega})}. \end{aligned} \tag{3.62}$$

We now apply Theorem 3.9 to the rightmost end of inequality 3.62, map back into Ω and finally apply the Poincaré inequality to obtain the thesis.

$$\begin{aligned} \|\nabla u_0\|^2_{L^2(\Omega)} + \sum_{\mathbf{j}=1}^{\mathbf{N}} \|\nabla u_{\mathbf{j}}\|^2_{L^2(\Omega)} &\lesssim \left(1 + \frac{H}{h}\right) \|\nabla \hat{u}\|^2_{L^2(\hat{\Omega})} \lesssim \\ &\lesssim \left(1 + \frac{H}{h}\right) \|u\|^2_{H^1(\Omega)} \lesssim \\ &\lesssim \left(1 + \frac{H}{h}\right) \|\nabla u\|^2_{L^2(\Omega)}. \end{aligned}$$

□

3.5 On the advection-diffusion equation

As already presented in Section 2.6.3, the advection-diffusion equations with the therein assumptions on κ, b and f, are:

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) + \mathbf{b} \cdot \nabla u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

In variational form we have

$$a(v, w) = F(v), \qquad \forall v \in H_0^1(\Omega),$$

where the bilinear form a and the functional F are defined as follow:

$$\begin{aligned} a(v, w) &= a^{diff}(v, w) + a^{adv}(v, w) = \\ &= \int_\Omega \kappa \nabla v \cdot \nabla w \, dx + \int_\Omega w \nabla v \cdot \mathbf{b} \, dx, \end{aligned} \tag{3.63}$$

$$F(v) = \int_\Omega f v \, dx. \tag{3.64}$$

Classical results of Overlapping Domain Decomposition methods ([33]) show that a similar bound to the one in Theorem 3.4 can be proved with the introduction of projection-like operators as 3.9: given a decomposition $\{\mathcal{V}_j\}_{j=0}^N$, the associated extension operators R_j^T, the exact local bilinear forms

$$a_j(v_j, w_j) = a\left(R_j^T v_j, R_j^T w_j\right), \qquad \forall v_j, w_j \in \mathcal{V}_j,$$

$$a_j^{diff}(v_j, w_j) = a^{diff}\left(R_j^T v_j, R_j^T w_j\right), \qquad \forall v_j, w_j \in \mathcal{V}_j,$$

with matrix local representation $A_j = R_j A R_j^T$ and $A_j^{diff} = R_j A^{diff} R_j^T$, we define

$$\tilde{P}_j : \mathcal{V} \longrightarrow \mathcal{V}_j, \qquad a_j(\tilde{P}_j v, w) = a(v, R_j^T w_j), \qquad \forall v_j \in \mathcal{V}_j, \tag{3.65}$$

$$\tilde{Q}_j : \mathcal{V} \longrightarrow \mathcal{V}_j, \qquad a_j^{diff}(\tilde{Q}_j v, w) = a(v, R_j^T w_j), \qquad \forall v_j \in \mathcal{V}_j, \tag{3.66}$$

In a similar fashion of Section 3.1, the final operators that constitute the building blocks of the preconditioner are:

$$P_j = R_j^T \tilde{P}_j = R_j^T A_j^{-1} R_j A,$$

$$Q_j = R_j^T \tilde{Q}_j = R_j^T (A_j^{diff})^{-1} R_j A^{diff},$$

We finally have two similar Additive Schwarz Operators for the advection-diffusion equation:

$$P^{(1)} = \sum_{j=0}^{N} P_j, \tag{3.67}$$

$$P^{(2)} = P_0 + \sum_{j=1}^{N} Q_j. \tag{3.68}$$

The difference between the two operators relies on the choice of the local projection-like operators, the coarse one being the same.
It can be shown (see [33], Theorem 11.1) for a classical FEM discretization scheme (piecewise linear functions on a triangular mesh) that there exist strictly positive constants H_0, $c(H_0)$ and $C(H_0)$ such that if $H \leq H_0$, then for $i = 1, 2$ and $u \in \mathcal{V}$

$$c(H_0)C_0^{-2}a(u,u) \leq a(u, P^{(i)}u), \qquad a(P^{(i)}u, P^{(i)}u) \leq C(H_0)a(u,u). \tag{3.69}$$

where $C_0 = (1 + \frac{H}{\delta})$ with H the coarse mesh size and δ an overlap parameter. This result ensures the invertibility of the operators 3.67 and 3.68, whenever the coarse mesh size H is sufficiently small: $H \leq H_0$.
To our knowledge, there are still no adaptation of this result for Isogeometric Analysis, i.e. for NURBS-based discrete functional spaces. Numerical results in Section 4.2 will show somehow a behavior that, accordingly with the theory in a FEM context, requires a minimum coarse mesh size H_0 in order to be effective.

Chapter 4

Numerical Results

In this Chapter numerical results for the scalar elliptic equation and for the advection-diffusion equation are presented. For both 2D and 3D cases are considered. Each case is tested with two geometries and each geometry is represented with a single element at the coarsest level, so that maximum global regularity $\mathcal{C}^{p-1}$ can be achieved with proper refinement (see Section 2.3.3).
We list here the 4 considered geometries with their knot vectors, polynomial degrees, control points and weights, for which in tables we adopted the format $\left((C_{\mathbf{i}})_1, ..., (C_{\mathbf{i}})_D\right), w_{\mathbf{i}}$. See Figures 4.1 and 4.2 for graphical representations of interest.

- **Unit square**:

$$\Omega = [0,1]^2$$

Knot vectors and polynomial degrees:

$$\Xi_1 = \Xi_2 = \{0,0,1,1\}, \qquad p_1 = p_2 = 1,$$

Control points and weights:

	$i_1 = 1$	$i_1 = 2$
$i_2 = 1$	$(0,0),\ 1$	$(1,0),\ 1$
$i_2 = 2$	$(1,0),\ 1$	$(1,1),\ 1$

- **Quarter annulus**:

$$\Omega = \left\{(x,y) \in \mathbb{R}^2 |\, x \geq 0,\ y \geq 0,\ 1 \leq x^2 + y^2 \leq 4\right\}$$

Knot vectors and polynomial degrees:

$$\Xi_1 = \{0, 0, 1, 1\}, \qquad p_1 = 1,$$
$$\Xi_2 = \{0, 0, 0, 1, 1, 1\}, \qquad p_2 = 2,$$

Control points and weights:

	$i_1 = 1$	$i_1 = 2$
$i_2 = 1$	$(1, 0),\ 1$	$(2, 0),\ 1$
$i_2 = 2$	$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right),\ \frac{\sqrt{2}}{2}$	$\left(\sqrt{2}, \sqrt{2}\right),\ \frac{\sqrt{2}}{2}$
$i_2 = 2$	$(0, 1),\ 1$	$(0, 2),\ 1$

- **Unit cube**:

$$\Omega = [0, 1]^3$$

Knot vectors and polynomial degrees:

$$\Xi_1 = \Xi_2 = \Xi_3 = \{0, 0, 1, 1\}, \qquad p_1 = p_2 = p_3 = 1,$$

Control points and weights:

	$i_1 = 1$	$i_1 = 2$
$i_2 = 1,\ i_3 = 1$	$(0, 0, 0),\ 1$	$(1, 0, 0),\ 1$
$i_2 = 2,\ i_3 = 1$	$(1, 0, 0),\ 1$	$(1, 1, 0),\ 1$
$i_2 = 1,\ i_3 = 2$	$(0, 0, 1),\ 1$	$(1, 0, 1),\ 1$
$i_2 = 2,\ i_3 = 2$	$(1, 0, 1),\ 1$	$(1, 1, 1),\ 1$

- **Quarter hose**:

$$\Omega = \left\{(x, y, z) \in \mathbb{R}^3 |\, 1 \leq \left(3 - \sqrt{x^2 + y^2}\right)^2 + z^2 \leq 4,\ x^2 + y^2 \geq 9\right\} \cup$$
$$\cup \left\{(x, y, z) \in \mathbb{R}^3 |\, x \geq 0,\ y \geq 0,\ z \geq 0\right\}$$

Knot vectors and polynomial degrees:

$$\Xi_1 = \Xi_2 = \{0, 0, 0, 1, 1, 1\}, \qquad p_1 = p_2 = 2,$$
$$\Xi_3 = \{0, 0, 1, 1\}, \qquad p_3 = 1,$$

Control points and weights:

	$i_1 = 1$	$i_1 = 2$	$i_1 = 3$
$i_2 = 1,\ i_3 = 1$	$(1, 0, 0),\ 1$	$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right),\ \frac{\sqrt{2}}{2}$	$(0, 1, 0),\ 1$
$i_2 = 2,\ i_3 = 1$	$\left(\frac{\sqrt{2}}{2}, 0, \sqrt{2}\right),\ \frac{\sqrt{2}}{2}$	$\left(\frac{1}{2}, \frac{1}{2}, 1\right),\ \frac{1}{2}$	$\left(0, \frac{\sqrt{2}}{2}, \sqrt{2}\right),\ \frac{\sqrt{2}}{2}$
$i_2 = 3,\ i_3 = 1$	$(3, 0, 2),\ 1$	$\left(\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{2}, \sqrt{2}\right),\ \frac{\sqrt{2}}{2}$	$(0, 3, 2), 1, 1$
$i_2 = 1,\ i_3 = 2$	$(2, 0, 0),\ 1$	$\left(\sqrt{2}, \sqrt{2}, 0\right),\ \frac{\sqrt{2}}{2}$	$(0, 2, 0),\ 1$
$i_2 = 2,\ i_3 = 2$	$\left(\sqrt{2}, 0, \frac{\sqrt{2}}{2}\right),\ \frac{\sqrt{2}}{2}$	$\left(1, 0, \frac{1}{2}\right),\ \frac{1}{2}$	$\left(0, \frac{\sqrt{2}}{,}\sqrt{2}2\right),\ \frac{\sqrt{2}}{2}$
$i_2 = 3,\ i_3 = 2$	$(3, 0, 1),\ 1$	$\left(\frac{3\sqrt{2}}{2}, \frac{3\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right),\ \frac{\sqrt{2}}{2}$	$(0, 3, 1),\ 1$

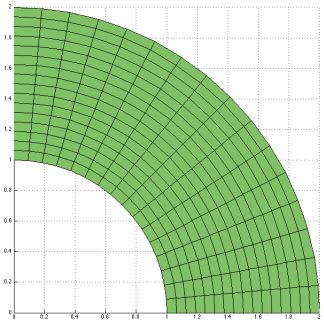


Figure 4.1: Quarter annulus mesh, refined in 16×16 uniform elements.

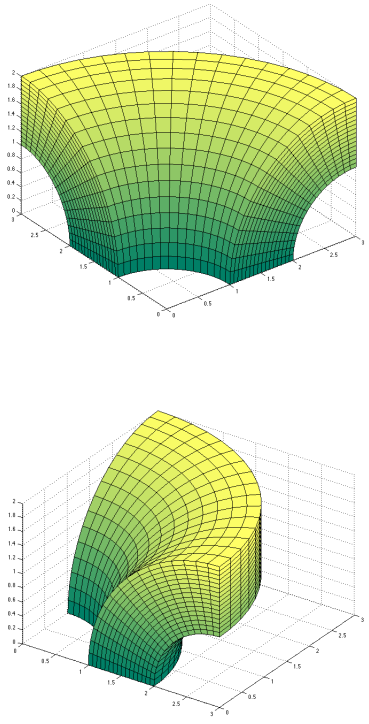


Figure 4.2: Quarter hose mesh, refined in $16 \times 16 \times 16$ uniform elements and depicted from different points of view.

In every test we adopted the *isoparametric concept*, i.e. the same NURBS basis functions used for geometry representation are also used as basis (eventually via push-forward as described in Section 2.5) for discrete function spaces for Petrov-Galerkin discretizations.
As one can notice, the square and the cube geometries are listed with control points and weights[1], even though the geometric map that describes them is the identity map. This means that for these two geometries the analysis is performed directly on the parametric domain with the usage of B-splines, the weights being identically 1. The other two geometries, the quarter annulus and the quarter hose, have to be described in a full NURBS fashion. The analysis on these geometries is hence performed with non-trivial NURBS basis functions in non-trivial physical domains, resulting in being examples of full Isogeometric Analysis capabilities.

[1] Just for the sake of completeness.

We can summarize the discrete function spaces associated to each geometry:

$$\begin{aligned} \text{square} &\longleftrightarrow \hat{\mathcal{B}}(\Xi_1, \Xi_2, p_1, p_2), \\ \text{quarter annulus} &\longleftrightarrow \mathcal{N}(\Xi_1, \Xi_2, p_1, p_2, w), \\ \text{cube} &\longleftrightarrow \hat{\mathcal{B}}(\Xi_1, \Xi_2, \Xi_3, p_1, p_2, p_3), \\ \text{quarter hose} &\longleftrightarrow \mathcal{N}(\Xi_1, \Xi_2, p_1, p_2, p_3, w), \end{aligned} \tag{4.1}$$

where knot vectors, polynomial degrees, control points and weights vary depending on the chosen mesh size or number of elements, accordingly with the refinement methods of Section 2.3.
The isoparametric concept also implies that discrete functional spaces must have at least the minimum polynomial degree required for an exact geometry representation. For the square and the cube, degree $p = 1$ is allowed along each direction although the resulting discrete space coincides with a standard FEM-like linear basis functions space, hence it is not of interest for this work. Test on the quarter annulus and the quarter hose require at least $p = 2$. Mixing different polynomial degrees is beyond the scope of this work: in each test the polynomial degree will be declared as an unidimensional quantity valid for each dimension. The regularity will always be the highest available.
Finally, we use the following notation for the 2D tests, the 3D being analogous.

- $N_{dom} = n_{dom} \times n_{dom}$: number of subdomains and hence number of processors involved in the computation.
- $N_{el} = n_{el} \times n_{el}$: number of elements per subdomain; the global number of elements is $N_{dom} N_{el}$.
- N_{dof}: the number of degrees of freedom and hence the dimension of the global discrete space $\mathcal{V}$.

4.1 The scalar elliptic equation

We recall the scalar elliptic equation in its full generality. Given $\Omega \subset \mathbb{R}^D$, the problem is finding a function $u : \Omega \longrightarrow \mathbb{R}$ such that

$$\begin{cases} -\nabla \cdot (\mathbf{K}\nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma_D, \end{cases} \tag{4.2}$$

where $f \in L^\infty(\Omega)$ and $\mathbf{K} = \kappa(x)\mathbf{Id}$. For the diffusivity coefficient we consider five different functions $\kappa = \hat{\kappa} \circ \mathbf{F}^{-1} : \Omega \longrightarrow \mathbb{R}$, where $\mathbf{F}$ is as usual the geometric function that defines Ω while

$$\hat{\kappa} : [0,1]^D \longrightarrow \mathbb{R},$$

$$\hat{\kappa}(\xi) = \begin{cases} k & \xi \in \left[\frac{1}{4}, \frac{3}{4}\right]^D, \\ 1 & \text{otherwise.} \end{cases} \qquad k \in \left\{1, 10^{-4}, 10^{-2}, 10^2, 10^4\right\}. \tag{4.3}$$

The diffusivity coefficient $k = 1$ is trivially a globally constant coefficient. The other four present a highly discontinuous central jump in the two middle quarters of the knot vectors, and are aimed to show the preconditioner performance with respect to discontinuous diffusivity coefficients.
The variational form of 4.2 leads to the variational problem: find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \kappa \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \qquad \forall v \in H_0^1(\Omega).$$

As last step, the Petrov-Galerkin discretization leads to the discrete problem: find $u_h \in \mathcal{V}$ such that

$$\int_\Omega \kappa \nabla u_h \cdot \nabla v_h \, dx = \int_\Omega f v_h \, dx, \qquad \forall v_h \in \mathcal{V},$$

where the discrete space $\mathcal{V} = \text{span}\{\phi_i | i = 1, ..., N_{dof}\}$ depends on the chosen geometry Ω, as in 4.1. With a small abuse of notation, the associated linear system $Au_h = f$ is finally defined as follow:

$$\begin{aligned} u &= [u_i]_{i=1}^{N_{dof}}, & \sum_{i=1}^{N_{dof}} u_i \phi_i = u_h, \\ A &= [a_{ij}]_{i,j=1}^{N_{dof}}, & a_{ij} = \int_\Omega \kappa \nabla \phi_i \cdot \nabla \phi_j \, dx, \\ f &= [f_i]_{i=1}^{N_{dof}}, & f_i = \int_\Omega f \phi_i \, dx. \end{aligned} \tag{4.4}$$

where integrals are computed with the $(p+1)$-points Gauss quadrature rule.

It is well known that the stiffness matrix A is symmetric and positive definite. As such, in order to solve the system the chosen iterative solver is the Preconditioned Conjugate Gradient method (PCG). We refer to [30] for a full explanation of the method.
The PCG is sensitive to the condition number of the system matrix. As general rule, the higher the condition number, the more the iteration required to solve the system up to a certain tolerance. The clustering of the eigenvalues is also important, so that PCG can perform differently with two matrices with identical condition numbers.
The bound of Theorem 3.4 states that as long as the ratio H/h is kept fixed, the condition number of the preconditioned stiffness matrix is bounded by a constant, thus it does not depend from the number of subdomains (i.e. the number of processors involved in the computation). As we will see, this behavior is confirmed by numerical results.
For the following tests, the PCG is set with two stopping criteria: the maximum number of iterations is 10000 as a divergence criterion, while the Euclidean norm of the preconditioned residual has to be less than 10^{-7} as a convergence criterion. Local matrices A_{j}^{-1} (see Definition 3.3) are not computed explicitly. The PCG (as any other iterative solver) requires only their action on a vector v as a matrix-vector product. Thus, the product $A_{\mathrm{j}}^{-1}v$ is computed as solution of a local linear system: solve $A_{\mathrm{j}}z = v$ in order to obtain $z = A_{\mathrm{j}}^{-1}v$. These local systems are set to be solved exactly with LU factorization. Their solving dominates the global solving in terms of time. The solver of the coarse problem $A_0^{-1}z$ is again the LU factorization.
In each 2D tests we refine each subdomain in $64 \times 64 = 4096$ elements, so that the ratio $H/h = 1/64$ is fixed. The number of overlapping subdomains ranges from $4 \times 4 = 16$ to $64 \times 64 = 4096$. The biggest tests involves about 16.7 million elements and a similar number of degrees of freedom. Polynomial degrees range from 2 to 4.
Lastly, computation of the minimum and maximum eigenvalues is performed with a Lanczos or Arnoldi iteration during the solving process. The entire computation is a PETSc black-box algorithm, for which we refer to [5] and [6].
Every test in this Section has been run on Fermi supercomputer ([1]), located at Cineca, Bologna, Italy. It is a BlueGene/Q machine based on IBM PowerA2 processors running at 1.6 GHz.
In all following Tables and Figures in Sections 4.1.1, 4.1.2, 4.1.3 and 4.1.4 we denote with red fonts the numerical results for $p = 2$, with blue for $p = 3$ and black for $p = 4$. In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

4.1.1 2D results: the square

The following tests show the scalability of the 2-level Additive Schwarz preconditioner for the B-spline-discretized scalar elliptic equation. Scalability is achieved with highly discontinuous central jumps, even though the condition number as a higher asymptotic constant trend. Unsurprisingly, the wider the jump, the higher the condition number.
In comparison, the 1-level Additive Schwarz Preconditioner does not scale, resulting in being an useful preconditioner only when the decomposition involves less then a hundred of subdomains (or processors).
It is worth to notice that the maximum eigenvalue is approximately $4 \approx 2^D$ in any case, so that it is somehow dimension-dependent. This is a consequence of Assumption 2, which involves the domain decomposition and not the coarse space, hence it is valid for both preconditioners. The maximum eigenvalue of the 2-level preconditioner is slightly greater due to the dimension dependency: the addition of a coarse problem add a "fractional dimension" flavor to the estimate.
In each test the fine mesh is refined in $4096 = 64 \times 64$ elements per subdomain, while polynomial degrees range from 2 to 4. In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

Diffusivity coefficient $k = 1$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	28	$\frac{4.000}{1.587e-02} = 251.99$	18	$\frac{4.000}{1.291e-01} = 30.98$
	$64 = 8 \times 8$	42	$\frac{4.000}{4.144e-03} = 965.25$	14	$\frac{4.000}{2.286e-01} = 17.50$
	$256 = 16 \times 16$	67	$\frac{4.000}{1.047e-03} = 3.82e+03$	12	$\frac{4.000}{2.321e-01} = 17.24$
	$1024 = 32 \times 32$	116	$\frac{4.000}{2.625e-04} = 1.52e+04$	10	$\frac{4.000}{2.330e-01} = 17.17$
	$4096 = 64 \times 64$	210	$\frac{4.000}{6.568e-05} = 6.09e+04$	7	$\frac{4.000}{2.342e-01} = 17.08$
$p = 3$	$16 = 4 \times 4$	25	$\frac{4.000}{2.173e-02} = 184.09$	17	$\frac{4.000}{2.826e-01} = 14.15$
	$64 = 8 \times 8$	37	$\frac{4.000}{5.682e-03} = 703.94$	16	$\frac{4.000}{2.478e-01} = 16.14$
	$256 = 16 \times 16$	59	$\frac{4.000}{1.437e-03} = 2.78e+03$	12	$\frac{4.000}{3.074e-01} = 13.01$
	$1024 = 32 \times 32$	102	$\frac{4.000}{3.602e-04} = 1.11e+04$	9	$\frac{4.000}{3.089e-01} = 12.95$
	$4096 = 64 \times 64$	187	$\frac{4.000}{9.011e-05} = 4.44e+04$	7	$\frac{4.000}{3.097e-01} = 12.92$
$p = 4$	$16 = 4 \times 4$	24	$\frac{4.000}{2.782e-02} = 143.79$	17	$\frac{4.000}{3.151e-01} = 12.69$
	$64 = 8 \times 8$	34	$\frac{4.000}{7.287e-03} = 548.93$	17	$\frac{4.000}{1.987e-01} = 20.13$
	$256 = 16 \times 16$	54	$\frac{4.000}{1.843e-03} = 2.17e+03$	14	$\frac{4.000}{1.871e-01} = 21.38$
	$1024 = 32 \times 32$	92	$\frac{4.000}{4.621e-04} = 8.66e+03$	8	$\frac{4.000}{3.837e-01} = 10.43$
	$4096 = 64 \times 64$	168	$\frac{4.000}{1.156e-04} = 3.46e+04$	7	$\frac{4.000}{3.843e-01} = 10.41$

Table 4.1: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 1$

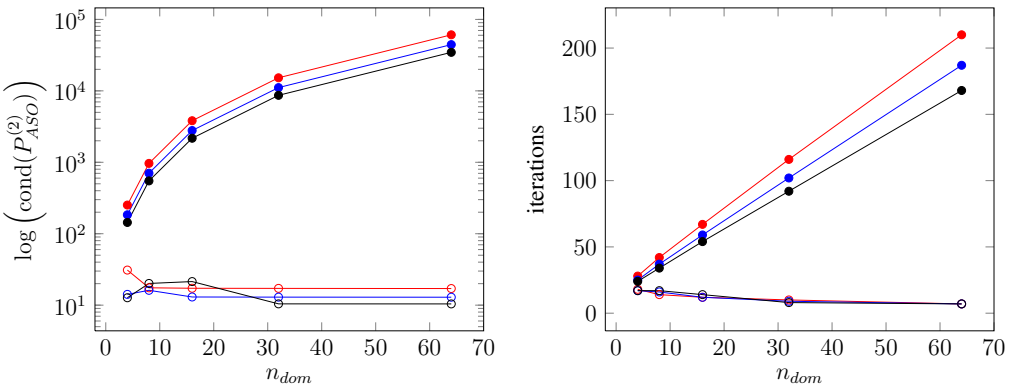


Figure 4.3: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 1$

Diffusivity coefficient $k = 10^{-4}$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	27	$\frac{4.000}{5.349e-02} = 74.79$	24	$\frac{4.003}{5.379e-02} = 74.42$
	$64 = 8 \times 8$	52	$\frac{4.000}{1.176e-02} = 340.12$	30	$\frac{4.003}{5.140e-02} = 77.89$
	$256 = 16 \times 16$	106	$\frac{4.000}{2.706e-03} = 1.48e+03$	38	$\frac{4.003}{5.646e-02} = 70.90$
	$1024 = 32 \times 32$	213	$\frac{4.000}{6.448e-04} = 6.20e+03$	39	$\frac{4.003}{5.802e-02} = 68.99$
	$4096 = 64 \times 64$	432	$\frac{4.000}{1.571e-04} = 2.55e+04$	40	$\frac{4.003}{5.789e-02} = 69.15$
$p = 3$	$16 = 4 \times 4$	24	$\frac{4.000}{7.272e-02} = 55.00$	20	$\frac{4.018}{7.447e-02} = 53.95$
	$64 = 8 \times 8$	47	$\frac{4.000}{1.608e-02} = 248.69$	30	$\frac{4.016}{4.429e-02} = 90.66$
	$256 = 16 \times 16$	93	$\frac{4.000}{3.709e-03} = 1.08e+03$	38	$\frac{4.018}{3.553e-02} = 113.08$
	$1024 = 32 \times 32$	186	$\frac{4.000}{8.843e-04} = 4.52e+03$	45	$\frac{4.019}{3.406e-02} = 118.00$
	$4096 = 64 \times 64$	375	$\frac{4.000}{2.156e-04} = 1.86e+04$	47	$\frac{4.020}{3.362e-02} = 119.58$
$p = 4$	$16 = 4 \times 4$	22	$\frac{4.000}{9.214e-02} = 43.41$	18	$\frac{4.015}{9.447e-02} = 42.50$
	$64 = 8 \times 8$	43	$\frac{4.000}{2.057e-02} = 194.46$	31	$\frac{4.029}{4.859e-02} = 82.92$
	$256 = 16 \times 16$	83	$\frac{4.000}{4.754e-03} = 841.43$	40	$\frac{4.028}{3.715e-02} = 108.41$
	$1024 = 32 \times 32$	167	$\frac{4.000}{1.134e-03} = 3.53e+03$	46	$\frac{4.028}{3.544e-02} = 113.64$
	$4096 = 64 \times 64$	336	$\frac{4.000}{2.765e-04} = 1.45e+04$	52	$\frac{4.028}{3.485e-02} = 115.58$

Table 4.2: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-4}$

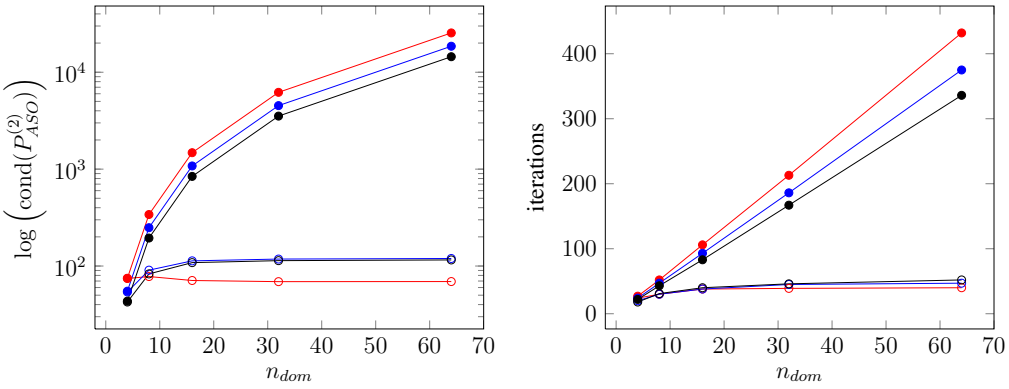


Figure 4.4: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-4}$

Diffusivity coefficient $k = 10^{-2}$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	30	$\frac{4.000}{4.990e-02} = 80.16$	26	$\frac{4.003}{5.939e-02} = 67.40$
	$64 = 8 \times 8$	59	$\frac{4.000}{1.141e-02} = 350.63$	32	$\frac{4.003}{5.555e-02} = 72.06$
	$256 = 16 \times 16$	111	$\frac{4.000}{2.643e-03} = 1.51e+03$	39	$\frac{4.003}{5.943e-02} = 67.35$
	$1024 = 32 \times 32$	227	$\frac{4.000}{6.319e-04} = 6.33e+03$	36	$\frac{4.003}{6.097e-02} = 65.65$
	$4096 = 64 \times 64$	464	$\frac{4.000}{1.542e-04} = 2.59e+04$	39	$\frac{4.003}{6.091e-02} = 65.72$
$p = 3$	$16 = 4 \times 4$	27	$\frac{4.000}{6.771e-02} = 59.07$	21	$\frac{4.014}{1.483e-01} = 27.07$
	$64 = 8 \times 8$	52	$\frac{4.000}{1.559e-02} = 256.53$	29	$\frac{4.011}{1.030e-01} = 38.96$
	$256 = 16 \times 16$	98	$\frac{4.000}{3.622e-03} = 1.10e+03$	30	$\frac{4.010}{9.080e-02} = 44.17$
	$1024 = 32 \times 32$	196	$\frac{4.000}{8.665e-04} = 4.62e+03$	34	$\frac{4.010}{8.884e-02} = 45.14$
	$4096 = 64 \times 64$	404	$\frac{4.000}{2.116e-04} = 1.89e+04$	32	$\frac{4.010}{8.821e-02} = 45.46$
$p = 4$	$16 = 4 \times 4$	25	$\frac{4.000}{8.577e-02} = 46.64$	21	$\frac{4.014}{1.156e-01} = 34.72$
	$64 = 8 \times 8$	48	$\frac{4.000}{1.995e-02} = 200.55$	27	$\frac{4.010}{9.693e-02} = 41.37$
	$256 = 16 \times 16$	88	$\frac{4.000}{4.643e-03} = 861.48$	30	$\frac{4.010}{1.046e-01} = 38.34$
	$1024 = 32 \times 32$	179	$\frac{4.000}{1.111e-03} = 3.60e+03$	30	$\frac{4.010}{1.102e-01} = 36.39$
	$4096 = 64 \times 64$	363	$\frac{4.000}{2.714e-04} = 1.47e+04$	31	$\frac{4.010}{1.118e-01} = 35.87$

Table 4.3: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-2}$

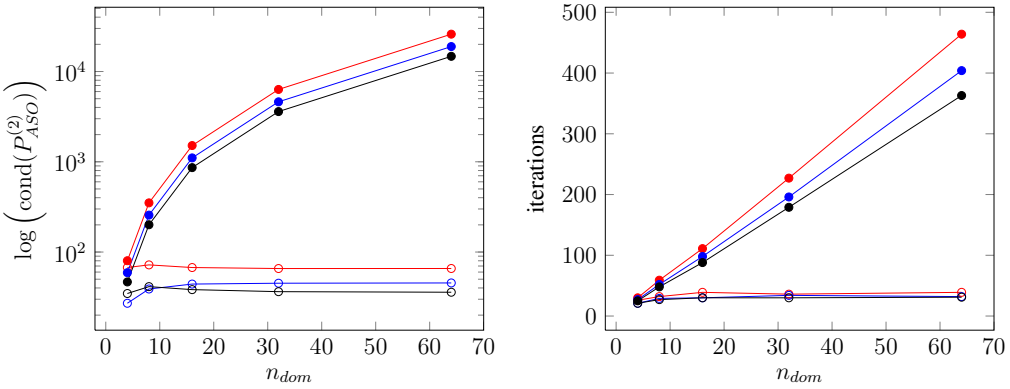


Figure 4.5: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-2}$

Diffusivity coefficient $k = 10^2$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	35	$\frac{4.000}{6.539e-04} = 6.12e+03$	24	$\frac{4.010}{1.215e-01} = 33.01$
	$64 = 8 \times 8$	73	$\frac{4.000}{1.128e-04} = 3.55e+04$	31	$\frac{4.011}{8.653e-02} = 46.35$
	$256 = 16 \times 16$	136	$\frac{4.000}{2.442e-05} = 1.64e+05$	37	$\frac{4.011}{6.621e-02} = 60.57$
	$1024 = 32 \times 32$	272	$\frac{4.000}{5.721e-06} = 6.99e+05$	39	$\frac{4.011}{6.158e-02} = 65.14$
	$4096 = 64 \times 64$	548	$\frac{4.000}{1.386e-06} = 2.89e+06$	37	$\frac{4.011}{6.116e-02} = 65.57$
$p = 3$	$16 = 4 \times 4$	32	$\frac{4.000}{8.972e-04} = 4.46e+03$	21	$\frac{4.023}{1.601e-01} = 25.14$
	$64 = 8 \times 8$	63	$\frac{4.000}{1.549e-04} = 2.58e+04$	27	$\frac{4.015}{1.061e-01} = 37.84$
	$256 = 16 \times 16$	119	$\frac{4.000}{3.352e-05} = 1.19e+05$	29	$\frac{4.013}{9.172e-02} = 43.75$
	$1024 = 32 \times 32$	236	$\frac{4.000}{7.851e-06} = 5.10e+05$	32	$\frac{4.013}{8.943e-02} = 44.87$
	$4096 = 64 \times 64$	478	$\frac{4.000}{1.903e-06} = 2.10e+06$	32	$\frac{4.013}{8.857e-02} = 45.31$
$p = 4$	$16 = 4 \times 4$	29	$\frac{4.000}{1.146e-03} = 3.49e+03$	20	$\frac{4.029}{2.119e-01} = 19.01$
	$64 = 8 \times 8$	57	$\frac{4.000}{1.982e-04} = 2.02e+04$	25	$\frac{4.021}{1.825e-01} = 22.03$
	$256 = 16 \times 16$	106	$\frac{4.000}{4.295e-05} = 9.31e+04$	27	$\frac{4.021}{1.410e-01} = 28.53$
	$1024 = 32 \times 32$	211	$\frac{4.000}{1.007e-05} = 3.97e+05$	30	$\frac{4.021}{1.166e-01} = 34.48$
	$4096 = 64 \times 64$	426	$\frac{4.000}{2.440e-06} = 1.64e+06$	30	$\frac{4.021}{1.124e-01} = 35.78$

Table 4.4: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^2$

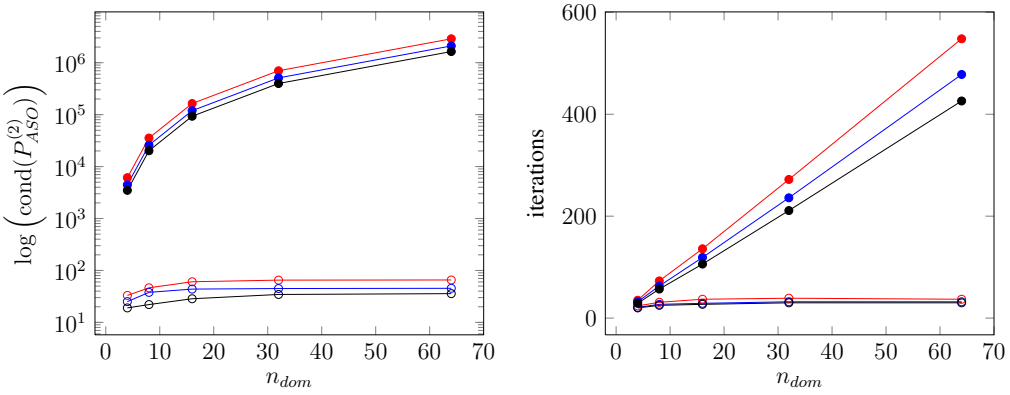


Figure 4.6: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^2$

Diffusivity coefficient $k = 10^4$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	38	$\frac{4.000}{6.971e-06} = 5.74e+05$	22	$\frac{4.014}{8.743e-02} = 45.91$
	$64 = 8 \times 8$	81	$\frac{4.000}{1.162e-06} = 3.44e+06$	31	$\frac{4.017}{6.759e-02} = 59.43$
	$256 = 16 \times 16$	152	$\frac{4.000}{2.491e-07} = 1.61e+07$	38	$\frac{4.017}{5.634e-02} = 71.30$
	$1024 = 32 \times 32$	298	$\frac{4.000}{5.812e-08} = 6.88e+07$	39	$\frac{4.017}{5.496e-02} = 73.10$
	$4096 = 64 \times 64$	606	$\frac{4.000}{1.406e-08} = 2.84e+08$	34	$\frac{4.017}{5.550e-02} = 72.38$
$p = 3$	$16 = 4 \times 4$	33	$\frac{4.000}{9.620e-06} = 4.16e+05$	23	$\frac{4.098}{9.268e-02} = 44.21$
	$64 = 8 \times 8$	71	$\frac{4.000}{1.599e-06} = 2.50e+06$	33	$\frac{4.062}{4.559e-02} = 89.09$
	$256 = 16 \times 16$	133	$\frac{4.000}{3.422e-07} = 1.17e+07$	43	$\frac{4.054}{3.296e-02} = 123.00$
	$1024 = 32 \times 32$	259	$\frac{4.000}{7.980e-08} = 5.01e+07$	45	$\frac{4.054}{3.289e-02} = 123.26$
	$4096 = 64 \times 64$	524	$\frac{4.000}{1.930e-08} = 2.07e+08$	43	$\frac{4.053}{3.312e-02} = 122.40$
$p = 4$	$16 = 4 \times 4$	31	$\frac{4.000}{1.240e-05} = 3.23e+05$	23	$\frac{4.113}{9.740e-02} = 42.22$
	$64 = 8 \times 8$	64	$\frac{4.000}{2.057e-06} = 1.95e+06$	32	$\frac{4.100}{4.262e-02} = 96.19$
	$256 = 16 \times 16$	119	$\frac{4.000}{4.396e-07} = 9.10e+06$	40	$\frac{4.084}{3.601e-02} = 113.43$
	$1024 = 32 \times 32$	232	$\frac{4.000}{1.024e-07} = 3.90e+07$	47	$\frac{4.081}{3.488e-02} = 116.99$
	$4096 = 64 \times 64$	467	$\frac{4.000}{2.477e-08} = 1.61e+08$	50	$\frac{4.081}{3.465e-02} = 117.79$

Table 4.5: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^4$

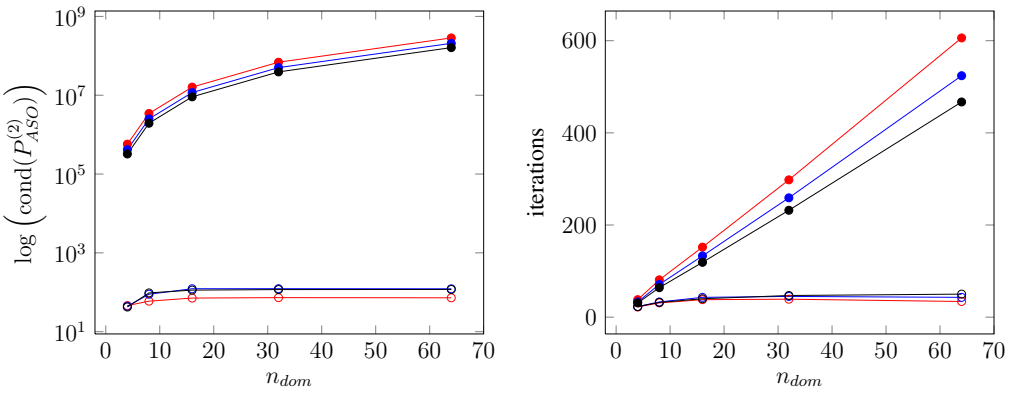
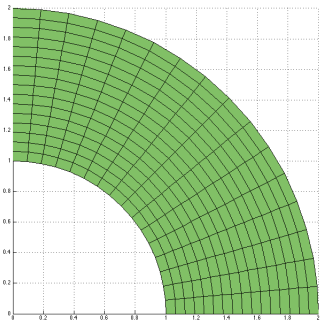


Figure 4.7: Square domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^4$

4.1.2 2D results: the quarter annulus



In the following tests the considered geometry is the quarter annulus, with the same setting of the square domain. Each subdomain is refined in $64 \times 64 = 4096$ elements thus $H/h = 1/64$. The number of overlapping subdomains ranges from $4 \times 4 = 16$ to $64 \times 64 = 4096$, while polynomial degrees from 2 to 4.
As one can see, scalability is achieved by the 2-level preconditioner, while the performance of the 1-level preconditioner deteriorates with an increasing number of subdomains.
Again, the introduction of a highly discontinuous diffusivity coefficient results in higher condition numbers.
In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

Diffusivity coefficient $k = 1$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	57	$\frac{4.000}{1.588e-2} = 251.91$	37	$\frac{4.000}{5.776e-2} = 69.25$
	$64 = 8 \times 8$	116	$\frac{4.000}{4.145e-03} = 964.92$	45	$\frac{4.000}{3.681e-02} = 108.66$
	$256 = 16 \times 16$	234	$\frac{4.000}{1.048e-03} = 3.82e+03$	33	$\frac{4.000}{6.635e-02} = 60.29$
	$1024 = 32 \times 32$	463	$\frac{4.000}{2.626e-04} = 1.52e+04$	28	$\frac{4.000}{7.505e-02} = 53.30$
	$4096 = 64 \times 64$	796	$\frac{4.000}{6.570e-05} = 6.09e+04$	25	$\frac{4.000}{7.568e-02} = 52.86$
$p = 3$	$16 = 4 \times 4$	49	$\frac{4.000}{2.174e-02} = 184.01$	28	$\frac{4.000}{1.192e-01} = 33.55$
	$64 = 8 \times 8$	101	$\frac{4.000}{5.685e-03} = 703.56$	31	$\frac{4.000}{7.232e-02} = 55.31$
	$256 = 16 \times 16$	202	$\frac{4.000}{1.438e-03} = 2.78e+03$	26	$\frac{4.000}{9.343e-02} = 42.81$
	$1024 = 32 \times 32$	404	$\frac{4.000}{3.604e-04} = 1.11e+04$	23	$\frac{4.000}{1.090e-01} = 36.68$
	$4096 = 64 \times 64$	688	$\frac{4.000}{9.016e-05} = 4.44e+04$	23	$\frac{4.000}{1.021e-01} = 39.19$
$p = 4$	$16 = 4 \times 4$	45	$\frac{4.000}{2.784e-02} = 143.70$	27	$\frac{4.000}{1.376e-01} = 29.07$
	$64 = 8 \times 8$	90	$\frac{4.000}{7.293e-03} = 548.49$	31	$\frac{4.000}{7.611e-02} = 52.55$
	$256 = 16 \times 16$	179	$\frac{4.000}{1.845e-03} = 2.17e+03$	26	$\frac{4.000}{1.050e-01} = 38.10$
	$1024 = 32 \times 32$	359	$\frac{4.000}{4.625e-04} = 8.65e+03$	18	$\frac{4.000}{1.456e-01} = 27.47$
	$4096 = 64 \times 64$	618	$\frac{4.000}{1.157e-04} = 3.46e+04$	21	$\frac{4.000}{1.306e-01} = 30.64$

Table 4.6: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 1$

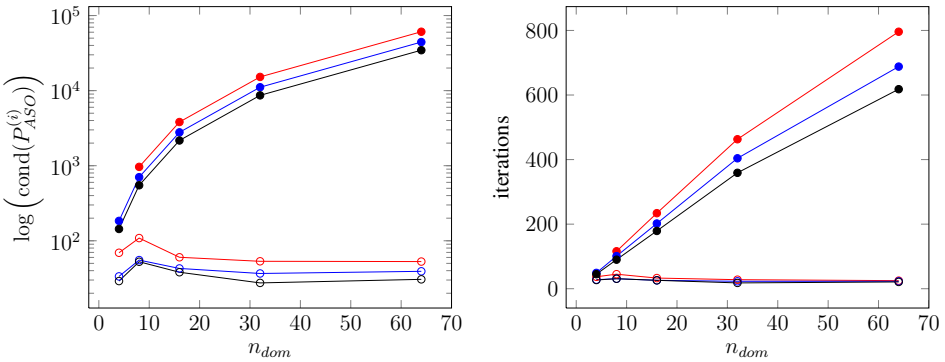


Figure 4.8: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 1$

Diffusivity coefficient $k = 10^{-4}$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	42	$\frac{4.000}{2.987e-02} = 133.92$	38	$\frac{4.004}{5.378e-02} = 74.46$
	$64 = 8 \times 8$	94	$\frac{4.000}{7.409e-03} = 539.86$	52	$\frac{4.005}{3.057e-02} = 130.99$
	$256 = 16 \times 16$	196	$\frac{4.000}{1.821e-03} = 2.20e+03$	70	$\frac{4.005}{2.356e-02} = 169.96$
	$1024 = 32 \times 32$	413	$\frac{4.000}{4.491e-04} = 8.91e+03$	76	$\frac{4.005}{2.016e-02} = 198.66$
	$4096 = 64 \times 64$	825	$\frac{4.000}{1.114e-04} = 3.59e+04$	73	$\frac{4.005}{1.817e-02} = 220.46$
$p = 3$	$16 = 4 \times 4$	38	$\frac{4.000}{4.061e-02} = 98.50$	31	$\frac{4.025}{7.446e-02} = 54.06$
	$64 = 8 \times 8$	82	$\frac{4.000}{1.014e-02} = 394.44$	53	$\frac{4.024}{3.696e-02} = 108.87$
	$256 = 16 \times 16$	169	$\frac{4.000}{2.496e-03} = 1.60e+03$	67	$\frac{4.031}{2.076e-02} = 194.14$
	$1024 = 32 \times 32$	356	$\frac{4.000}{6.160e-04} = 6.49e+03$	78	$\frac{4.035}{1.532e-02} = 263.43$
	$4096 = 64 \times 64$	711	$\frac{4.000}{1.528e-04} = 2.62e+04$	81	$\frac{4.037}{1.298e-02} = 311.09$
$p = 4$	$16 = 4 \times 4$	35	$\frac{4.000}{5.163e-02} = 77.47$	29	$\frac{4.022}{9.443e-02} = 42.59$
	$64 = 8 \times 8$	73	$\frac{4.000}{1.298e-02} = 308.19$	48	$\frac{4.044}{4.761e-02} = 84.92$
	$256 = 16 \times 16$	150	$\frac{4.000}{3.200e-03} = 1.25e+03$	64	$\frac{4.043}{2.922e-02} = 138.35$
	$1024 = 32 \times 32$	317	$\frac{4.000}{7.903e-04} = 5.06e+03$	74	$\frac{4.043}{2.048e-02} = 197.40$
	$4096 = 64 \times 64$	633	$\frac{4.000}{1.961e-04} = 2.04e+04$	76	$\frac{4.043}{1.669e-02} = 242.32$

Table 4.7: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-4}$

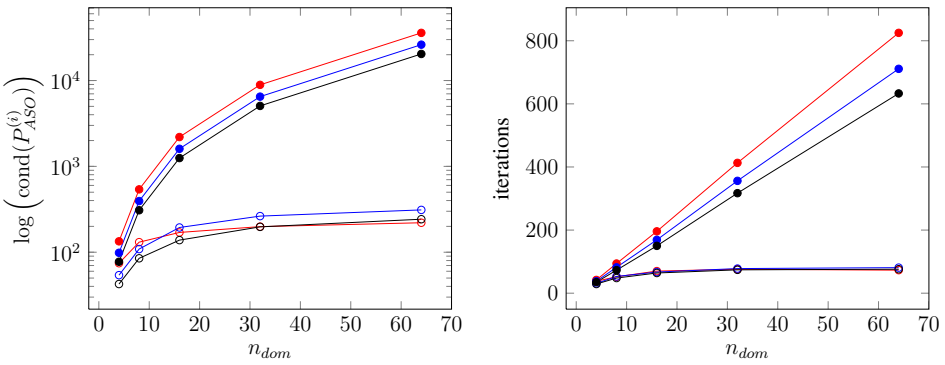


Figure 4.9: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-4}$

Diffusivity coefficient $k = 10^{-2}$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	47	$\frac{4.000}{2.976e-02} = 134.42$	42	$\frac{4.004}{5.838e-02} = 68.59$
	$64 = 8 \times 8$	100	$\frac{4.000}{7.399e-03} = 540.62$	58	$\frac{4.005}{3.284e-02} = 121.96$
	$256 = 16 \times 16$	215	$\frac{4.000}{1.820e-03} = 2.20e+03$	69	$\frac{4.005}{2.514e-02} = 159.27$
	$1024 = 32 \times 32$	451	$\frac{4.000}{4.491e-04} = 8.91e+03$	73	$\frac{4.005}{2.169e-02} = 184.61$
	$4096 = 64 \times 64$	907	$\frac{4.000}{1.114e-04} = 3.59e+04$	68	$\frac{4.005}{1.963e-02} = 203.98$
$p = 3$	$16 = 4 \times 4$	42	$\frac{4.000}{4.045e-02} = 98.88$	32	$\frac{4.020}{1.313e-01} = 30.62$
	$64 = 8 \times 8$	88	$\frac{4.000}{1.013e-02} = 395.04$	44	$\frac{4.018}{8.174e-02} = 49.15$
	$256 = 16 \times 16$	186	$\frac{4.000}{2.495e-03} = 1.60e+03$	55	$\frac{4.022}{5.507e-02} = 73.04$
	$1024 = 32 \times 32$	389	$\frac{4.000}{6.160e-04} = 6.49e+03$	56	$\frac{4.025}{4.393e-02} = 91.63$
	$4096 = 64 \times 64$	785	$\frac{4.000}{1.529e-04} = 2.62e+04$	51	$\frac{4.025}{3.854e-02} = 104.44$
$p = 4$	$16 = 4 \times 4$	39	$\frac{4.000}{5.144e-02} = 77.76$	30	$\frac{4.020}{1.164e-01} = 34.53$
	$64 = 8 \times 8$	78	$\frac{4.000}{1.296e-02} = 308.63$	44	$\frac{4.016}{6.069e-02} = 66.18$
	$256 = 16 \times 16$	166	$\frac{4.000}{3.199e-03} = 1.25e+03$	53	$\frac{4.016}{4.612e-02} = 87.09$
	$1024 = 32 \times 32$	346	$\frac{4.000}{7.903e-04} = 5.06e+03$	56	$\frac{4.016}{4.166e-02} = 96.42$
	$4096 = 64 \times 64$	701	$\frac{4.000}{1.961e-04} = 2.04e+04$	56	$\frac{4.017}{3.843e-02} = 104.53$

Table 4.8: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-2}$

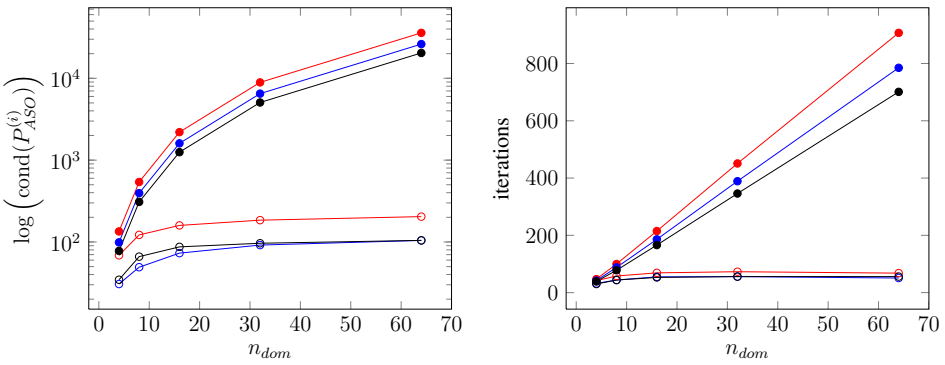


Figure 4.10: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^{-2}$

Diffusivity coefficient $k = 10^2$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	58	$\frac{4.000}{6.387e-04} = 6.26e+03$	37	$\frac{4.014}{5.429e-02} = 73.94$
	$64 = 8 \times 8$	125	$\frac{4.000}{1.099e-04} = 3.64e+04$	52	$\frac{4.012}{3.369e-02} = 119.08$
	$256 = 16 \times 16$	264	$\frac{4.000}{2.377e-05} = 1.68e+05$	61	$\frac{4.012}{2.421e-02} = 165.74$
	$1024 = 32 \times 32$	548	$\frac{4.000}{5.564e-06} = 7.19e+05$	63	$\frac{4.012}{2.086e-02} = 192.35$
	$4096 = 64 \times 64$	1096	$\frac{4.000}{1.348e-06} = 2.97e+06$	71	$\frac{4.012}{1.911e-02} = 209.94$
$p = 3$	$16 = 4 \times 4$	51	$\frac{4.000}{8.772e-04} = 4.56e+03$	34	$\frac{4.030}{1.045e-01} = 38.56$
	$64 = 8 \times 8$	109	$\frac{4.000}{1.510e-04} = 2.65e+04$	43	$\frac{4.026}{6.157e-02} = 65.38$
	$256 = 16 \times 16$	228	$\frac{4.000}{3.263e-05} = 1.23e+05$	48	$\frac{4.025}{4.544e-02} = 88.58$
	$1024 = 32 \times 32$	472	$\frac{4.000}{7.638e-06} = 5.24e+05$	48	$\frac{4.026}{3.929e-02} = 102.47$
	$4096 = 64 \times 64$	944	$\frac{4.000}{1.851e-06} = 2.16e+06$	51	$\frac{4.026}{3.601e-02} = 111.80$
$p = 4$	$16 = 4 \times 4$	46	$\frac{4.000}{1.121e-03} = 3.57e+03$	31	$\frac{4.032}{1.453e-01} = 27.76$
	$64 = 8 \times 8$	97	$\frac{4.000}{1.933e-04} = 2.07e+04$	37	$\frac{4.025}{8.505e-02} = 47.33$
	$256 = 16 \times 16$	202	$\frac{4.000}{4.183e-05} = 9.56e+04$	46	$\frac{4.025}{5.234e-02} = 76.90$
	$1024 = 32 \times 32$	419	$\frac{4.000}{9.797e-06} = 4.08e+05$	48	$\frac{4.025}{4.142e-02} = 97.18$
	$4096 = 64 \times 64$	837	$\frac{4.000}{2.374e-06} = 1.68e+06$	51	$\frac{4.025}{3.802e-02} = 105.85$

Table 4.9: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^2$

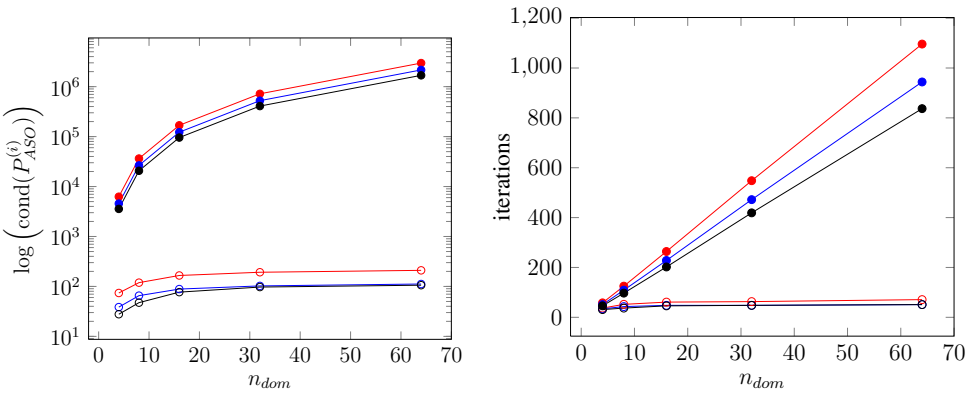


Figure 4.11: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^2$

Diffusivity coefficient $k = 10^4$

p	$N_{dom} = n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$16 = 4 \times 4$	65	$\frac{4.000}{6.799e-06} = 5.88e+05$	40	$\frac{4.019}{4.394e-02} = 91.45$
	$64 = 8 \times 8$	142	$\frac{4.000}{1.131e-06} = 3.54e+06$	53	$\frac{4.018}{2.863e-02} = 140.36$
	$256 = 16 \times 16$	297	$\frac{4.000}{2.422e-07} = 1.65e+07$	66	$\frac{4.018}{2.127e-02} = 188.94$
	$1024 = 32 \times 32$	613	$\frac{4.000}{5.649e-08} = 7.08e+07$	63	$\frac{4.018}{1.872e-02} = 214.63$
	$4096 = 64 \times 64$	1245	$\frac{4.000}{1.366e-08} = 2.93e+08$	71	$\frac{4.018}{1.742e-02} = 230.64$
$p = 3$	$16 = 4 \times 4$	57	$\frac{4.000}{9.395e-06} = 4.26e+05$	38	$\frac{4.103}{5.400e-02} = 75.98$
	$64 = 8 \times 8$	124	$\frac{4.000}{1.558e-06} = 2.57e+06$	54	$\frac{4.067}{2.646e-02} = 153.72$
	$256 = 16 \times 16$	257	$\frac{4.000}{3.330e-07} = 1.20e+07$	68	$\frac{4.059}{1.626e-02} = 249.67$
	$1024 = 32 \times 32$	528	$\frac{4.000}{7.760e-08} = 5.15e+07$	76	$\frac{4.059}{1.347e-02} = 301.25$
	$4096 = 64 \times 64$	1071	$\frac{4.000}{1.876e-08} = 2.13e+08$	87	$\frac{4.059}{1.204e-02} = 337.17$
$p = 4$	$16 = 4 \times 4$	52	$\frac{4.000}{1.212e-05} = 3.30e+05$	37	$\frac{4.139}{6.434e-02} = 64.33$
	$64 = 8 \times 8$	110	$\frac{4.000}{2.005e-06} = 1.99e+06$	50	$\frac{4.111}{3.006e-02} = 136.76$
	$256 = 16 \times 16$	228	$\frac{4.000}{4.280e-07} = 9.35e+06$	63	$\frac{4.095}{2.076e-02} = 197.31$
	$1024 = 32 \times 32$	468	$\frac{4.000}{9.966e-08} = 4.01e+07$	69	$\frac{4.093}{1.709e-02} = 239.52$
	$4096 = 64 \times 64$	948	$\frac{4.000}{2.409e-08} = 1.66e+08$	86	$\frac{4.093}{1.509e-02} = 271.22$

Table 4.10: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^4$

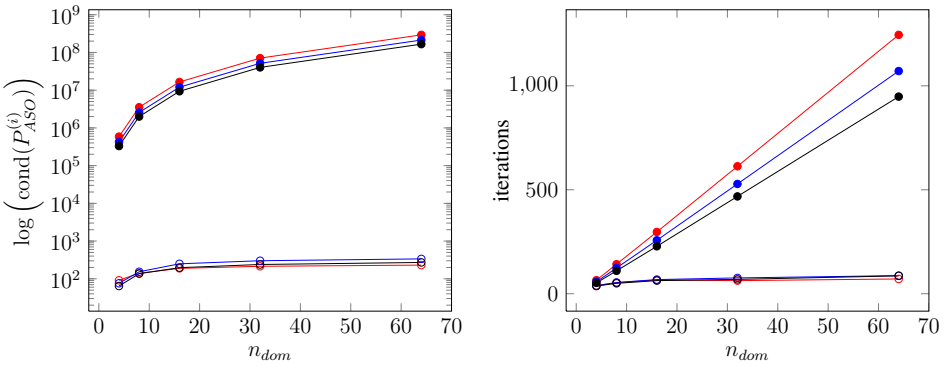


Figure 4.12: Quarter annulus domain, $N_{el} = 4096 = 64 \times 64$, $k = 10^4$

4.1.3 3D results: the cube

For the cube domain we present relatively small set of tests due to limited availability of the supercomputers at our disposal.
In these tests, the number of subdomains ranges from $64 = 4 \times 4 \times 4$ to $512 = 8 \times 8 \times 8$, while the number of elements per subdomain is kept fixed at $4096 = 16 \times 16 \times 16$. Quadratic and cubic functions are considered only, the case $p = 4$ being too much expensive in terms of computational time at disposal.
The results show how the 1-level Preconditioner is outperformed by the 2-level one, with analogous behaviors of the 2D case.
In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

Diffusivity coefficient $k = 1$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner		2-level preconditioner	
		it.	cond	it.	cond
$p = 2$	$64 = 4 \times 4 \times 4$	26	$\frac{8.000}{7.095e-02} = 112.75$	19	$\frac{8.000}{3.706e-01} = 21.59$
	$512 = 8 \times 8 \times 8$	36	$\frac{8.000}{1.876e-02} = 426.42$	16	$\frac{8.000}{5.813e-01} = 13.76$
$p = 3$	$64 = 4 \times 4 \times 4$	25	$\frac{8.000}{1.015e-01} = 78.85$	18	$\frac{8.007}{7.370e-01} = 10.86$
	$512 = 8 \times 8 \times 8$	34	$\frac{8.000}{2.706e-02} = 295.59$	18	$\frac{8.002}{6.083e-01} = 13.15$

Table 4.11: Cube domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 1$

Diffusivity coefficient $k = 10^{-4}$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner		2-level preconditioner	
		it.	cond	it.	cond
$p = 2$	$64 = 4 \times 4 \times 4$	26	$\frac{8.000}{1.450e-01} = 55.18$	21	$\frac{8.004}{2.344e-01} = 34.15$
	$512 = 8 \times 8 \times 8$	46	$\frac{8.000}{3.672e-02} = 217.88$	29	$\frac{8.004}{1.697e-01} = 47.18$
$p = 3$	$64 = 4 \times 4 \times 4$	25	$\frac{8.000}{2.010e-01} = 39.80$	22	$\frac{8.035}{3.336e-01} = 24.09$
	$512 = 8 \times 8 \times 8$	43	$\frac{8.000}{5.247e-02} = 152.46$	29	$\frac{8.041}{1.616e-01} = 49.75$

Table 4.12: Cube domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-4}$

Diffusivity coefficient $k = 10^{-2}$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	28	$\frac{8.000}{1.427e-01} = 56.04$	24	$\frac{8.004}{2.424e-01} = 33.02$
	$512 = 8 \times 8 \times 8$	51	$\frac{8.000}{3.628e-02} = 220.49$	29	$\frac{8.004}{1.800e-01} = 44.47$
$p = 3$	$64 = 4 \times 4 \times 4$	26	$\frac{8.000}{1.980e-01} = 40.40$	22	$\frac{8.028}{4.282e-01} = 18.75$
	$512 = 8 \times 8 \times 8$	46	$\frac{8.000}{5.186e-02} = 154.27$	26	$\frac{8.029}{2.570e-01} = 31.24$

Table 4.13: Cube domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-2}$

Diffusivity coefficient $k = 10^2$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	36	$\frac{8.000}{3.598e-03} = 2.22e+03$	25	$\frac{8.016}{2.318e-01} = 34.57$
	$512 = 8 \times 8 \times 8$	62	$\frac{8.000}{6.279e-04} = 1.27e+04$	30	$\frac{8.013}{1.857e-01} = 43.14$
$p = 3$	$64 = 4 \times 4 \times 4$	33	$\frac{8.000}{5.208e-03} = 1.54e+03$	23	$\frac{8.054}{2.959e-01} = 27.22$
	$512 = 8 \times 8 \times 8$	55	$\frac{8.000}{9.101e-04} = 8.79e+03$	26	$\frac{8.038}{2.243e-01} = 35.83$

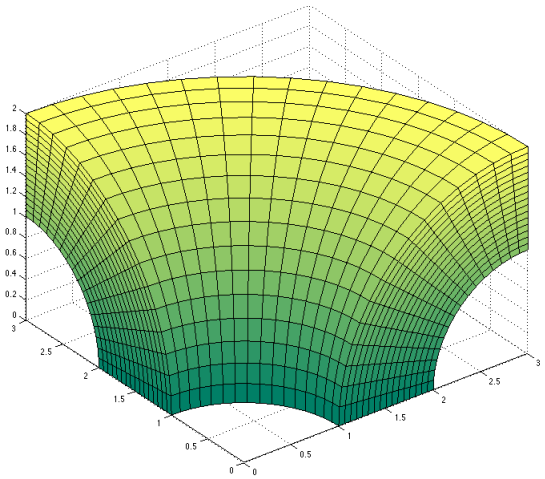
Table 4.14: Cube domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^2$

Diffusivity coefficient $k = 10^4$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	39	$\frac{8.000}{3.889e-05} = 2.06e+05$	25	$\frac{8.060}{1.494e-01} = 53.94$
	$512 = 8 \times 8 \times 8$	70	$\frac{8.000}{6.530e-06} = 1.23e+06$	31	$\frac{8.070}{1.287e-01} = 62.71$
$p = 3$	$64 = 4 \times 4 \times 4$	37	$\frac{8.000}{5.736e-05} = 1.39e+05$	25	$\frac{8.233}{1.573e-01} = 52.34$
	$512 = 8 \times 8 \times 8$	62	$\frac{8.000}{9.548e-06} = 8.38e+05$	34	$\frac{8.144}{1.089e-01} = 74.75$

Table 4.15: Cube domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^4$

4.1.4 3D results: the quarter hose



The hose domain is now considered with a complete suite of tests. The number of subdomains ranges from $64 = 4 \times 4 \times 4$ to $4096 = 16 \times 16 \times 16$, while the number of elements per subdomain is kept fixed at $4096 = 16 \times 16 \times 16$. Quadratic and cubic functions are considered.
Again, scalability with respect to number of subdomains and jumps on the diffusivity coefficient is achieved.
In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

Diffusivity coefficient $k = 1$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	50	$\frac{8.000}{7.098e-02} = 112.70$	31	$\frac{8.000}{2.626e-01} = 30.47$
	$512 = 8 \times 8 \times 8$	92	$\frac{8.000}{1.882e-02} = 425.12$	34	$\frac{8.000}{2.090e-01} = 38.29$
	$1728 = 12 \times 12 \times 12$	137	$\frac{8.000}{8.455e-03} = 946.24$	33	$\frac{8.000}{1.913e-01} = 41.82$
	$4096 = 16 \times 16 \times 16$	184	$\frac{8.000}{4.774e-03} = 1.68e+03$	32	$\frac{8.000}{1.829e-01} = 43.75$
$p = 3$	$64 = 4 \times 4 \times 4$	50	$\frac{8.000}{1.010e-01} = 79.23$	29	$\frac{8.020}{3.501e-01} = 22.91$
	$512 = 8 \times 8 \times 8$	100	$\frac{8.000}{2.699e-02} = 296.37$	35	$\frac{8.010}{2.782e-01} = 28.78$
	$1728 = 12 \times 12 \times 12$	117	$\frac{8.000}{1.224e-02} = 653.46$	32	$\frac{8.005}{2.538e-01} = 31.54$
	$4096 = 16 \times 16 \times 16$	157	$\frac{8.000}{6.917e-03} = 1.16e+03$	32	$\frac{8.004}{2.442e-01} = 32.78$

Table 4.16: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 1$

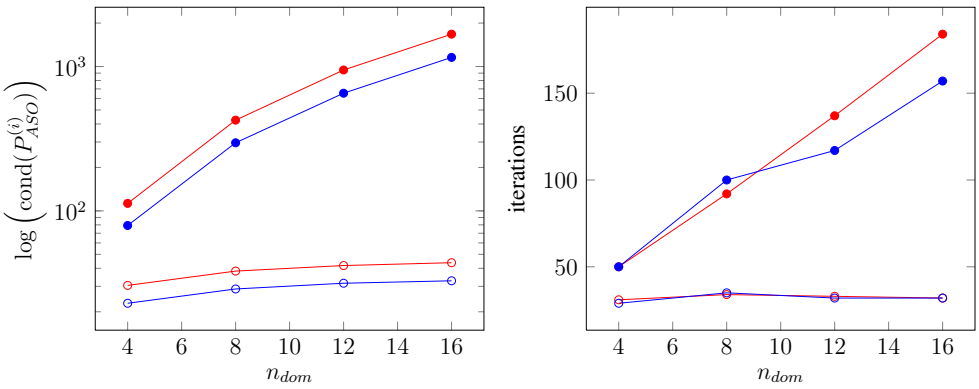


Figure 4.13: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 1$

Diffusivity coefficient $k = 10^{-4}$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	44	$\frac{8.000}{1.003e-01} = 79.80$	38	$\frac{8.006}{1.956e-01} = 40.93$
	$512 = 8 \times 8 \times 8$	88	$\frac{8.000}{2.666e-02} = 300.05$	54	$\frac{8.008}{1.081e-01} = 74.07$
	$1728 = 12 \times 12 \times 12$	129	$\frac{8.000}{1.194e-02} = 669.99$	59	$\frac{8.009}{7.823e-02} = 102.38$
	$4096 = 16 \times 16 \times 16$	177	$\frac{8.000}{6.724e-03} = 1.19e+03$	64	$\frac{8.009}{6.519e-02} = 122.86$
$p = 3$	$64 = 4 \times 4 \times 4$	40	$\frac{8.000}{1.410e-01} = 56.75$	43	$\frac{8.071}{1.177e-01} = 45.49$
	$512 = 8 \times 8 \times 8$	77	$\frac{8.000}{3.828e-02} = 209.01$	57	$\frac{8.091}{9.827e-02} = 82.32$
	$1728 = 12 \times 12 \times 12$	112	$\frac{8.000}{1.724e-02} = 464.13$	61	$\frac{8.083}{9.188e-02} = 87.98$
	$4096 = 16 \times 16 \times 16$	151	$\frac{8.000}{9.722e-03} = 822.90$	66	$\frac{8.102}{6.984e-02} = 116.00$

Table 4.17: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-4}$

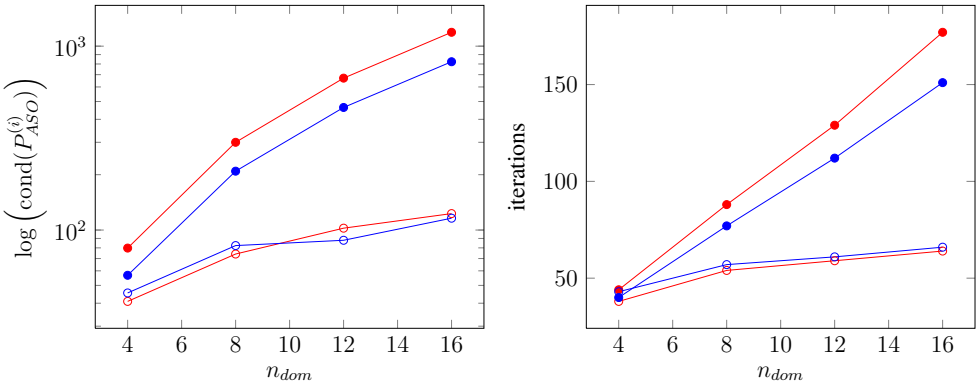


Figure 4.14: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-4}$

Diffusivity coefficient $k = 10^{-2}$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	47	$\frac{8.000}{1.002e-01} = 79.85$	41	$\frac{8.006}{1.896e-01} = 42.23$
	$512 = 8 \times 8 \times 8$	96	$\frac{8.000}{2.666e-02} = 300.09$	56	$\frac{8.008}{9.300e-02} = 86.11$
	$1728 = 12 \times 12 \times 12$	146	$\frac{8.000}{1.194e-02} = 670.05$	64	$\frac{8.008}{7.184e-02} = 111.48$
	$4096 = 16 \times 16 \times 16$	196	$\frac{8.000}{6.724e-03} = 1.19e+03$	65	$\frac{8.008}{6.917e-02} = 115.78$
$p = 3$	$64 = 4 \times 4 \times 4$	42	$\frac{8.000}{1.409e-01} = 56.78$	44	$\frac{8.042}{3.196e-01} = 25.15$
	$512 = 8 \times 8 \times 8$	89	$\frac{8.000}{3.732e-02} = 214.34$	47	$\frac{8.052}{1.569e-01} = 51.29$
	$1728 = 12 \times 12 \times 12$	124	$\frac{8.000}{1.724e-02} = 464.06$	50	$\frac{8.060}{1.296e-01} = 62.21$
	$4096 = 16 \times 16 \times 16$	166	$\frac{8.000}{9.724e-03} = 822.73$	53	$\frac{8.072}{1.091e-01} = 73.96$

Table 4.18: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-2}$

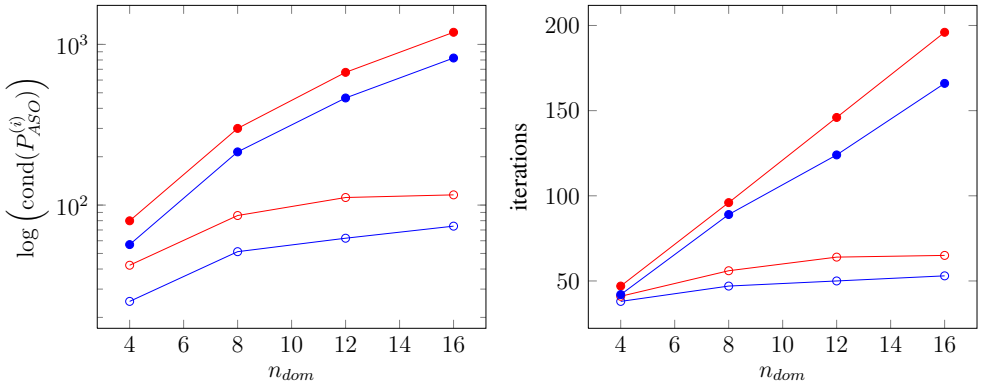


Figure 4.15: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^{-2}$

Diffusivity coefficient $k = 10^2$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	63	$\frac{8.000}{3.419e-03} = 2.34e+03$	43	$\frac{8.022}{1.298e-01} = 61.79$
	$512 = 8 \times 8 \times 8$	134	$\frac{8.000}{5.908e-04} = 1.35e+04$	56	$\frac{8.018}{8.199e-02} = 97.80$
	$1728 = 12 \times 12 \times 12$	205	$\frac{8.000}{2.381e-04} = 3.36e+04$	62	$\frac{8.018}{6.513e-02} = 123.11$
	$4096 = 16 \times 16 \times 16$	277	$\frac{8.000}{1.279e-04} = 6.25e+04$	67	$\frac{8.018}{5.748e-02} = 139.50$
$p = 3$	$64 = 4 \times 4 \times 4$	61	$\frac{8.000}{5.220e-03} = 1.53e+03$	46	$\frac{8.083}{2.253e-01} = 35.88$
	$512 = 8 \times 8 \times 8$	144	$\frac{8.000}{8.988e-04} = 8.90e+03$	48	$\frac{8.093}{1.603e-01} = 50.46$
	$1728 = 12 \times 12 \times 12$	173	$\frac{8.000}{3.469e-04} = 2.31e+04$	51	$\frac{8.090}{1.271e-01} = 63.64$
	$4096 = 16 \times 16 \times 16$	233	$\frac{8.000}{1.863e-04} = 4.30e+04$	52	$\frac{8.092}{1.126e-01} = 71.86$

Table 4.19: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^2$

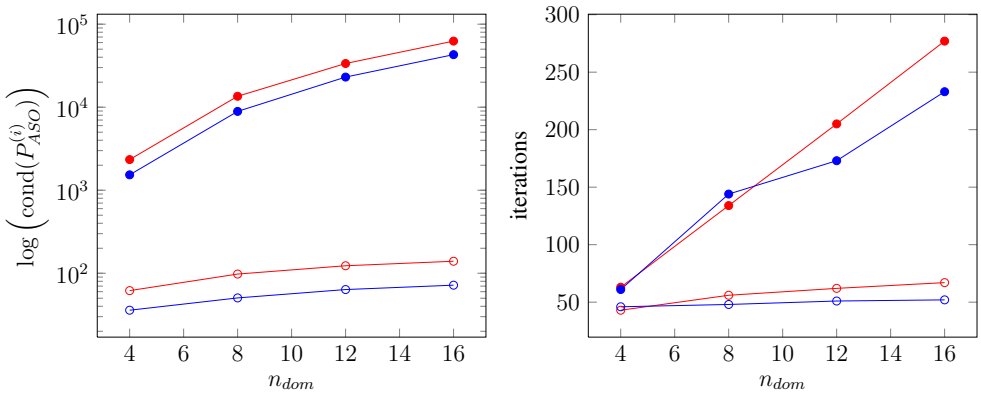


Figure 4.16: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^2$

Diffusivity coefficient $k = 10^4$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner it.	1-level preconditioner cond	2-level preconditioner it.	2-level preconditioner cond
$p = 2$	$64 = 4 \times 4 \times 4$	74	$\frac{8.000}{3.685e-05} = 2.17e+05$	46	$\frac{8.089}{9.326e-02} = 86.74$
	$512 = 8 \times 8 \times 8$	156	$\frac{8.000}{6.132e-06} = 1.30e+06$	62	$\frac{8.077}{6.349e-02} = 127.21$
	$1728 = 12 \times 12 \times 12$	238	$\frac{8.000}{2.451e-06} = 3.26e+06$	69	$\frac{8.077}{5.254e-02} = 153.72$
	$4096 = 16 \times 16 \times 16$	319	$\frac{8.000}{1.313e-06} = 6.09e+06$	72	$\frac{8.077}{4.757e-02} = 169.78$
$p = 3$	$64 = 4 \times 4 \times 4$	71	$\frac{8.000}{5.854e-05} = 1.37e+05$	50	$\frac{8.215}{9.565e-02} = 75.88$
	$512 = 8 \times 8 \times 8$	170	$\frac{8.000}{9.540e-06} = 8.39e+05$	65	$\frac{8.199}{6.509e-02} = 125.95$
	$1728 = 12 \times 12 \times 12$	200	$\frac{8.000}{3.596e-06} = 2.22e+06$	67	$\frac{8.178}{5.389e-02} = 151.75$
	$4096 = 16 \times 16 \times 16$	269	$\frac{8.000}{1.921e-06} = 4.17e+06$	75	$\frac{8.176}{4.658e-02} = 175.52$

Table 4.20: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^4$

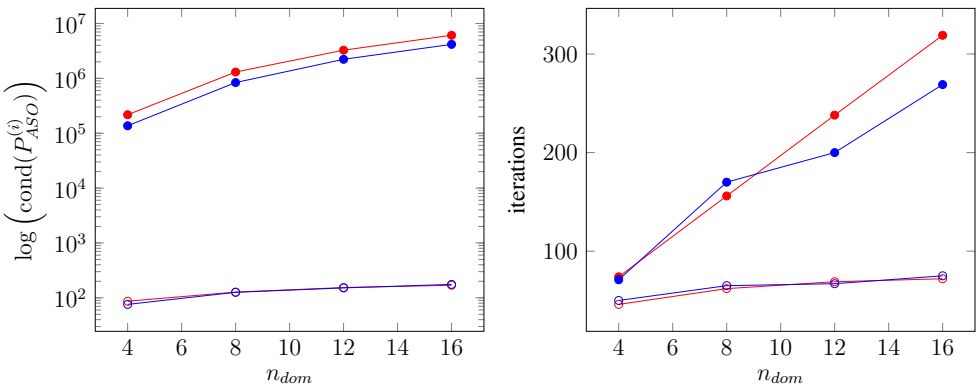


Figure 4.17: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 10^4$

4.2 The advection-diffusion equation

We recall the advection diffusion equations as presented in Section 2.6.3. The model problem is finding a function $u : \Omega \longrightarrow \mathbb{R}$ such that

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) + \mathbf{b} \cdot \nabla u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

where $\Omega \subset \mathbb{R}^D$ is an open domain with boundary $\partial\Omega$, $f : \Omega \longrightarrow \mathbb{R}$ is the body force, κ is the diffusivity coefficient and $\mathbf{b} : \Omega \longrightarrow \mathbb{R}^D$ is the velocity field, assumed to be divergence-free, i.e. $\nabla \cdot \mathbf{b} = 0$. We recall also the SUPG stabilization:

$$a_{SUPG}(v, w) = a(u, v) + (\tau \mathbf{b} \cdot \nabla v, \mathcal{L} w)_{L^2(\tilde{\Omega})},$$

$$F_{SUPG}(v) = F(v) + (\tau \mathbf{b} \cdot \nabla v, f)_{L^2(\tilde{\Omega})}.$$

where $\mathcal{L} v = -\kappa \Delta v + \mathbf{b} \cdot \nabla v$. The stabilization parameter τ depends on the mesh sizes h and H, respectively for the local problems and the coarse problem. It is chosen to be

$$\tau = \tau(\kappa, h, p, \mathbf{b}) = \frac{h_K}{2p|\mathbf{b}|} \phi\left(\mathbb{P}\mathrm{e}_K\right), \tag{4.5}$$

where h_K is the length of the mesh size along the direction of the velocity field $\mathbf{b}$, $\mathbb{P}\mathrm{e}_K = \frac{|\mathbf{b}| h_K}{2p\kappa}$ is the so called *local Péclet number* and $\phi(\alpha) = \coth(\alpha) - \frac{1}{\alpha}$. We refer to [34] and references therein for a deeper analysis.
For these tests we consider the square and cube domains. The annulus and hose domains are addressed in some minor preliminar tests. For each hypercubic domain, we consider both the unstabilized and the SUPG-stabilized B-spline discretization of the model problem, with a given constant, divergence-free velocity field

$$\mathbf{b}(x_1, ..., x_D) = [D, ..., 1], \qquad D = 2, 3. \tag{4.6}$$

The diffusivity parameter κ ranges in the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$. As the numerical results show, the higher the diffusivity coefficient, the more diffusion-dominated is the model problem. In order to test the robustness of the preconditioner with respect to boundary layers, we select exact solutions with gradient depending on κ, so that lower values for κ give exact solutions with higher boundary layers. The solutions are

$$u(x_1, ..., x_D) = \prod_{d=1}^{D} x_d \left(1 - e^{\frac{x_d - 1}{2\kappa}}\right).$$

and the body forces are explicitly computed as $f = \mathcal{L} u$.

The system matrix A arising from the Petrov-Galerkin discretization is positive definite but not symmetric. This implies the use of Generalized Minimal Residual Method (GMRES) as iterative solver (see [30] for a deeper analysis). In each test, GMRES is set with a maximum number of 10000 iterations for the 2D tests and 1000 iterations for the 3D tests. The tolerance of the Euclidean norm of the preconditioned residual is again 10^{-7}. Due to PETSc limitations, there are no explicit estimates on the minimum and maximum eigenvalues of the preconditioned system matrix, hence we report the number of iterations required to achieved the convergence of the solver up to the said tolerance.
The considered preconditioners are 3.67 and the 1-level version, i.e. the sum of the same local projection operators without the coarse grid correction given by P_0. The 2-level Isogeometric Additive Schwarz Preconditioner is hence compared with the 1-level Additive Schwarz black-box preconditioner implemented in PETSc library.
Finally, the machine used[2] for these tests is the Mira supercomputer, located at the Argonne National Laboratory, Illinois, USA. The supercomputer is a BluGene/Q based on PowerPC A2 processors running at 1.6Ghz. We refer to [2] for technical details.

4.2.1 2D results: the square

For the square domain, we fix the number of elements per subdomain at $4096 = 16 \times 16$. Polynomial degree ranges from 2 to 4, with the same color scheme from Section 4.1: red for $p = 2$, blue for $p = 3$ and black for $p = 4$. Regularity of basis functions is $\mathcal{C}^{p-1}$. The number of subdomains ranges from 8×8 subdomains or processors, up to $8100 = 90 \times 90$ for the $\kappa = 10^{-2}$ case, achieving successfully an Isogeometric discretization of 33 millions elements, or approximately 37 millions degrees of freedom.
Due to supercomputer utilization limitations, we tested the $\kappa = 10^{-1}$ and $\kappa = 10^{-3}$ cases with up to respectively $2304 = 48 \times 48$ and $3136 = 56 \times 56$ subdomains.
Tables 4.22 shows that a high diffusivity coefficient $\kappa = 10^{-1}$, and hence a diffusion-dominated problem, brings no difficulties to the 2-level Preconditioner, while the 1-level one quickly fails to converge with an increasing number of subdomains. The SUPG stabilization has little to none effect. Similar remarks can be done for the $\kappa = 10^{-2}$ case.
In the advection-dominated case $\kappa = 10^{-3}$ (Table 4.24), the 1-level preconditioner performs better but still shows an increasing trend with respect to the number of subdomain. The 2-level preconditioner as well shows an increasing trend from

[2]Supercomputer utilization has been supported and supervised by Professor L. F. Pavarino.

8×8 to 24×24 subdomains. With more subdomains, it recovers the asymptotic constant trend seen in previous tests.
An explanation of this behavior can be found in a trade-off between the effectiveness of the 1-level preconditioner and the quality of the coarse grid solution. With a small number of subdomains, the 1-level preconditioned solver converges rapidly due to quick share of boundary data among the few processors, and the coarse problem has little effect. When the number of subdomains raises, the 1-level preconditioner starts to be less effective. The coarse problem, on the other hand, may be not refined enough to ensure a good global correction, as it is known that the advection-diffusion problem presents difficulties with insufficiently low mesh sizes. This behavior agrees with theoretical remarks in Section 3.5: in order to be effective, the 2-level preconditioner requires a maximum mesh size H_0, hence a minimum number of subdomains. As one can see in Table 4.21, the case $k = 10^{-1}$ shows that local Péclet numbers for the coarse grid are well below 1, hence the optimal results, meanwhile in the advection-dominated case $k = 10^{-3}$ the local Péclet numbers are above the unitary threshold in the 8×8 to 32×32 subdomains range.
The SUPG stabilization enhances the stability of the discretization scheme in general, so that the trade-off between coarse problem and 1-level preconditioning is less evident.

$N_{dom} =$	$k = 10^{-1}$			$k = 10^{-2}$			$k = 10^{-3}$		
$n_{dom} \times n_{dom}$	$p = 2$	$p = 3$	$p = 4$	$p = 2$	$p = 3$	$p = 4$	$p = 2$	$p = 3$	$p = 4$
$64 = 8 \times 8$	0.70	0.47	0.35	6.99	4.66	3.49	69.88	46.58	34.94
$256 = 16 \times 16$	0.35	0.23	0.17	3.49	2.33	1.75	34.94	23.29	17.47
$576 = 24 \times 24$	0.23	0.16	0.12	2.33	1.55	1.16	23.29	15.53	11.65
$1024 = 32 \times 32$	0.17	0.12	0.09	1.75	1.16	0.87	17.47	11.65	8.73
$1600 = 40 \times 40$	0.14	0.09	0.07	1.40	0.93	0.70	13.98	9.32	6.99
$2304 = 48 \times 48$	0.12	0.08	0.06	1.16	0.78	0.58	11.65	7.76	5.82
$3136 = 56 \times 56$	0.10	0.07	0.05	1.00	0.67	0.50	9.98	6.65	4.99
$4096 = 64 \times 64$	0.09	0.06	0.04	0.87	0.58	0.44	8.73	5.82	4.37

Table 4.21: Local Péclet number as in 4.5 for the 2D coarse grid.

Lastly, it is worth to notice in Table 4.24 how the case $p = 3$ achieves scalability but performs worse than the other two cases. It starts to be known in the IGA community that there are theoretical and practical differences between odd and even polynomial degrees discretizations, with the $p = 3$ case being particularly anomalous. Further investigation is required and it is beyond the scope of this work.
In the Figures, plots depicted with full circles represent the 1-level Schwarz Preconditioner, while empty circles the 2-level Preconditioner.

Diffusivity coefficient $\kappa = 10^{-1}$

$N_{dom} =$	No stabilization						SUPG stabilization					
	$p = 2$		$p = 3$		$p = 4$		$p = 2$		$p = 3$		$p = 4$	
$n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 8 \times 8$	161	26	147	21	112	21	160	26	147	21	112	21
$256 = 16 \times 16$	553	23	319	17	207	18	621	23	320	17	207	18
$576 = 24 \times 24$	10000	20	680	15	439	16	10000	20	715	15	438	16
$1024 = 32 \times 32$	10000	19	10000	14	651	15	10000	19	10000	15	651	15
$1600 = 40 \times 40$	10000	17	10000	13	1281	14	10000	17	10000	14	1228	14
$2304 = 48 \times 48$	10000	15	10000	13	2236	13	10000	16	10000	13	2310	13

Table 4.22: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-1}$.

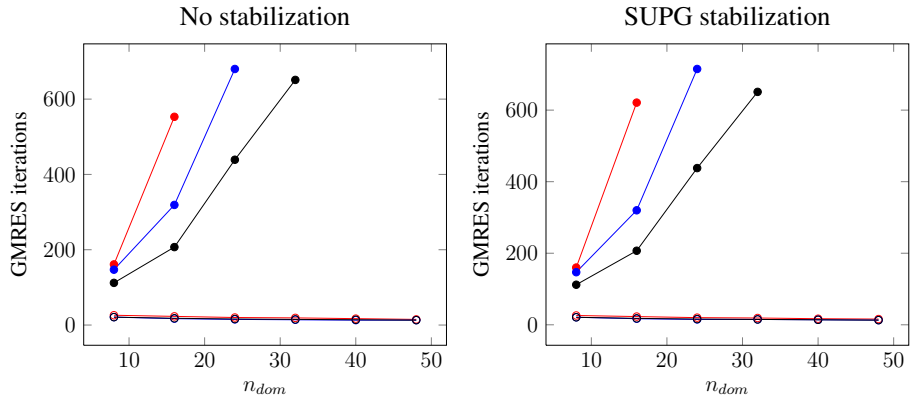


Figure 4.18: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-1}$.

Diffusivity coefficient $\kappa = 10^{-2}$

$N_{dom} =$	No stabilization						SUPG stabilization					
	$p = 2$		$p = 3$		$p = 4$		$p = 2$		$p = 3$		$p = 4$	
$n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 8 \times 8$	37	43	29	42	27	29	37	38	29	37	27	29
$256 = 16 \times 16$	146	34	125	39	114	26	150	35	125	35	114	26
$576 = 24 \times 24$	228	30	217	34	192	24	225	34	217	31	192	24
$1024 = 32 \times 32$	313	26	258	30	207	22	322	29	260	29	206	23
$1600 = 40 \times 40$	395	25	365	27	295	20	389	27	364	27	296	21
$2304 = 48 \times 48$	497	24	435	26	394	19	482	25	436	25	395	19
$3136 = 56 \times 56$	665	23	525	23	466	18	664	24	520	23	449	18
$4096 = 64 \times 64$	820	22	541	21	480	18	829	22	543	21	507	18
$5184 = 72 \times 72$	1471	22	652	20	578	17	1203	22	666	20	583	17
$6400 = 80 \times 80$	10000	21	748	19	627	17	10000	21	747	18	630	17
$7744 = 88 \times 88$	10000	21	844	17	702	16	10000	21	839	18	703	16
$8100 = 90 \times 90$	10000	21	882	17	743	16	10000	21	867	17	738	16

Table 4.23: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-2}$.

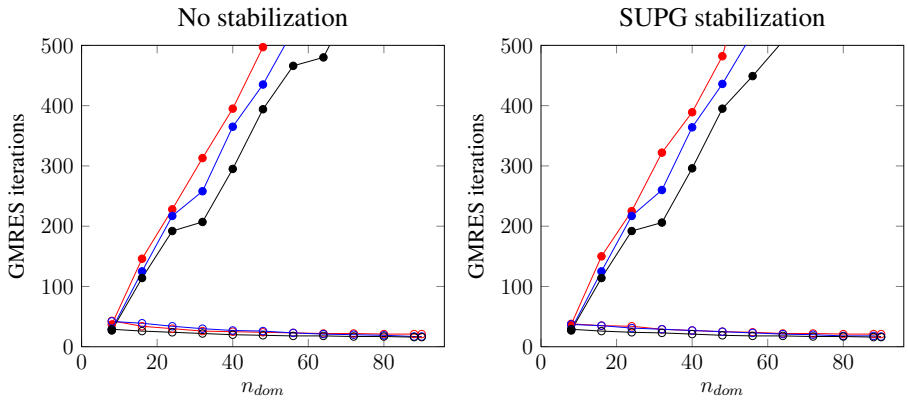


Figure 4.19: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-2}$.

Diffusivity coefficient $\kappa = 10^{-3}$

	No stabilization						SUPG stabilization					
$N_{dom} =$	$p = 2$		$p = 3$		$p = 4$		$p = 2$		$p = 3$		$p = 4$	
$n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 8 \times 8$	26	83	29	84	29	60	23	43	26	48	27	41
$256 = 16 \times 16$	50	146	40	168	47	92	44	52	40	66	45	58
$576 = 24 \times 24$	120	157	119	186	119	113	120	58	120	82	118	72
$1024 = 32 \times 32$	161	152	174	182	173	113	153	58	175	80	174	71
$1600 = 40 \times 40$	238	115	247	157	192	96	234	53	244	72	192	57
$2304 = 48 \times 48$	188	80	267	130	264	75	216	48	267	66	266	46
$3136 = 56 \times 56$	316	62	246	110	247	58	313	42	242	59	248	40

Table 4.24: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-3}$.

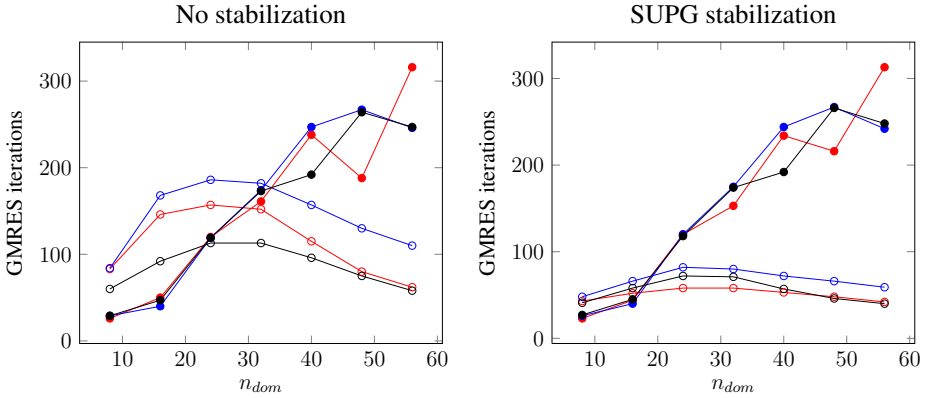


Figure 4.20: Square domain, $N_{el} = 64 \times 64$, $\kappa = 10^{-3}$.

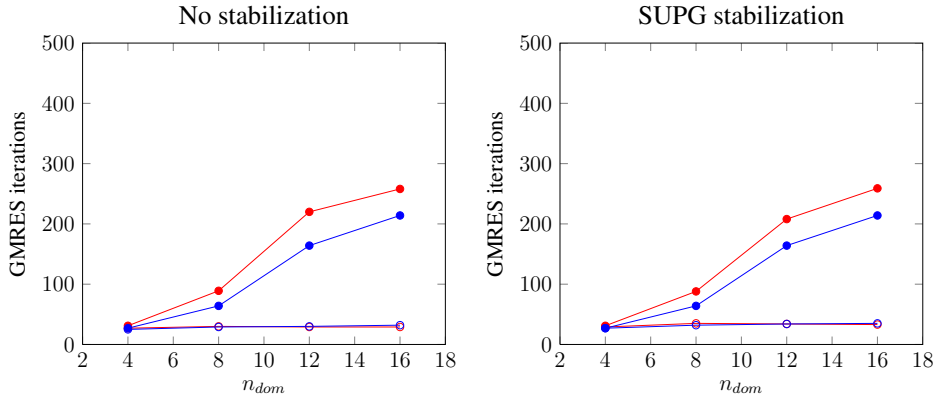
4.2.2 3D results: the cube

For the 3D tests we considered the cube domain, refined in $4096 = 16 \times 16 \times 16$ elements per subdomains. As for the scalar elliptic equation, polynomial degrees are $p = 2$ (red) and $p = 3$ (blue). Tests are a direct comparison between the 1-level and the 2-level preconditioners, with and without SUPG stabilization.
Table 4.25 is one more evidence of the scalability of the 2-level preconditioner for diffusion-dominated problems. As in the 2D case, there is almost no difference between the unstabilized and the SUPG-stabilized problems.
Table 4.26 shows instead that with no stabilization the 2-level preconditioned solver fail to converge for any decomposition and polynomial degree, while the 1-level preconditioner performs surprisingly well, although it still show a growing behavior with respect to the number of processors. The SUPG stabilization recovers the scalability and overall performance of the 2-level preconditioner.
Finally, in the the advection-dominated case $\kappa = 10^{-3}$ (Table 4.27), the unstabilized tests show failure for the 2-level preconditioner, while the SUPG-stabilized ones are meaningful if compared with the $\kappa = 10^{-2}$ tests. Our idea is that the 2-level preconditioner with $p = 3$ has a bump trend as discussed in the 2D case, even though it goes beyond our GMRES settings.

Diffusivity coefficient $k = 10^{-1}$

$N_{dom} =$	No stabilization				SUPG stabilization			
	$p = 2$		$p = 3$		$p = 2$		$p = 3$	
$n_{dom} \times n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 4 \times 4 \times 4$	31	27	27	25	31	29	27	27
$512 = 8 \times 8 \times 8$	89	30	64	29	88	35	64	32
$1728 = 12 \times 12 \times 12$	220	29	164	30	208	34	164	34
$4096 = 16 \times 16 \times 16$	258	29	214	32	259	33	214	35

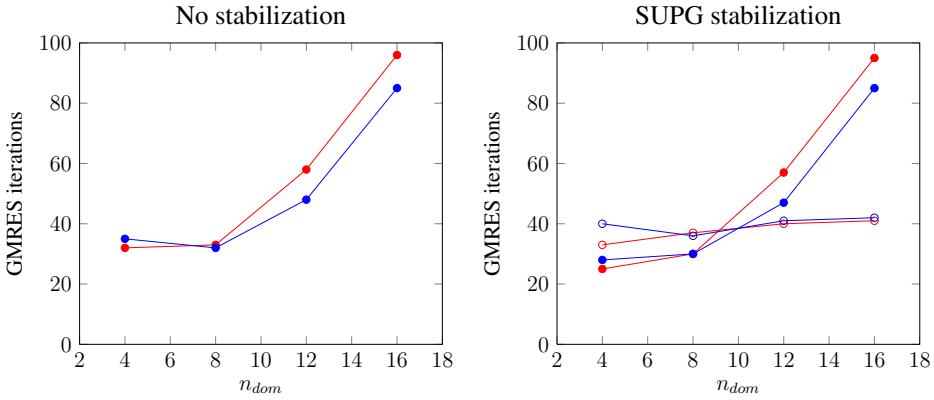
Table 4.25: Cube domain, $N_{el} = 16 \times 16 \times 16$, $\kappa = 10^{-1}$.



Diffusivity coefficient $k = 10^{-2}$

$N_{dom} =$	No stabilization				SUPG stabilization			
	$p = 2$		$p = 3$		$p = 2$		$p = 3$	
$n_{dom} \times n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 4 \times 4 \times 4$	32	1000	35	1000	25	33	28	40
$512 = 8 \times 8 \times 8$	33	1000	32	1000	30	37	30	36
$1728 = 12 \times 12 \times 12$	58	1000	48	1000	57	40	47	41
$4096 = 16 \times 16 \times 16$	96	1000	85	1000	95	41	85	42

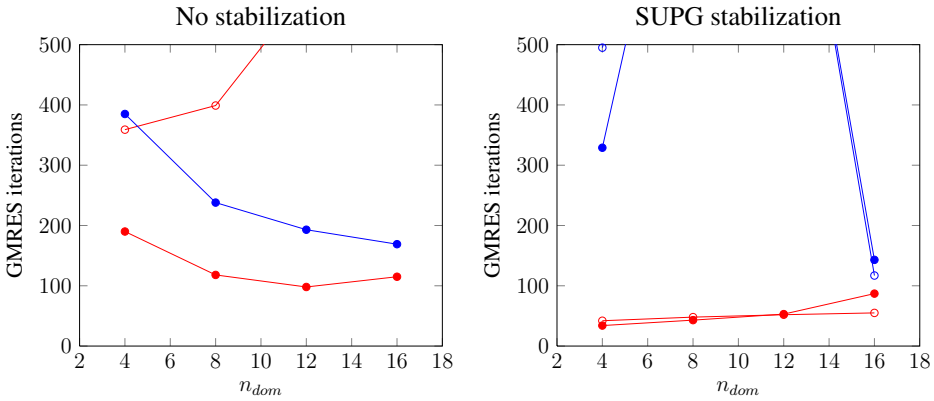
Table 4.26: Cube domain, $N_{el} = 16 \times 16 \times 16$, $\kappa = 10^{-2}$.



Diffusivity coefficient $k = 10^{-3}$

$N_{dom} =$ $n_{dom} \times n_{dom} \times n_{dom}$	No stabilization				SUPG stabilization			
	$p = 2$		$p = 3$		$p = 2$		$p = 3$	
	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 4 \times 4 \times 4$	190	359	385	1000	34	42	329	495
$512 = 8 \times 8 \times 8$	118	399	238	1000	43	48	1000	1000
$1728 = 12 \times 12 \times 12$	98	590	193	1000	53	52	1000	1000
$4096 = 16 \times 16 \times 16$	115	1000	169	1000	87	55	143	117

Table 4.27: Cube domain, $N_{el} = 16 \times 16 \times 16$, $\kappa = 10^{-3}$.



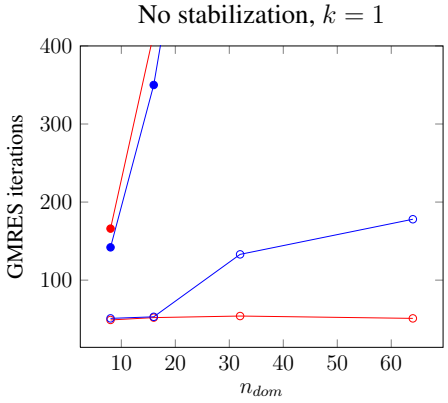
4.2.3 Preliminary results on deformed NURBS domains

We present here two early tests on the two deformed domains treated in this work: the quarter annulus and the hose. The number of subdomains ranges from 8×8 to 64×64 with polynomial degrees $p = 2, 3$ for the 2D case, while the number of subdomains ranges from $4 \times 4 \times 4$ to $16 \times 16 \times 16$ with polynomial degree $p = 2$ for the 3D one. The velocity field is again the constant divergence-free field defined in 4.6.
Due to technical limitations, no SUPG stabilization is considered and the diffusivity coefficient is $k = 1$, resulting in a diffusion dominated case. These preliminar results show the good scalability of the 2-level Preconditioner for quadratic ($p = 2$) NURBS basis functions, easily outperforming the black-box 1-level Preconditioner in both dimensions. With the 2D cubic basis functions the 2-level Preconditioner does not perform well as in the quadratic case, although it seems to show an asymptotically constant trend with respect to the number of subdomains.

Quarter annulus 2D domain

$N_{dom} =$	$p = 2$		$p = 3$	
$n_{dom} \times n_{dom}$	1-lvl	2-lvl	1-lvl	2-lvl
$64 = 8 \times 8$	166	49	142	51
$256 = 16 \times 16$	416	52	350	53
$1024 = 32 \times 32$	1000	54	1000	133
$4096 = 64 \times 64$	1000	51	1000	178

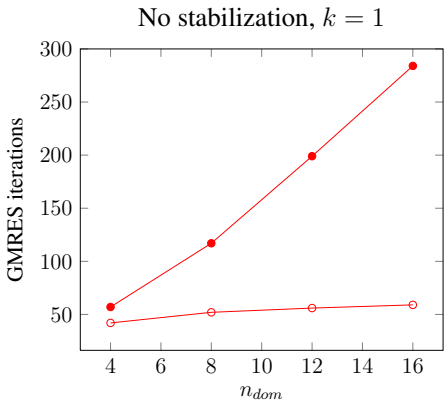
Table 4.28: Quarter annulus domain, $N_{el} = 64 \times 64$, $\kappa = 1$.



Hose 3D domain

$N_{dom} =$ $n_{dom} \times n_{dom} \times n_{dom}$	$p = 2$	
	1-lvl	2-lvl
$64 = 4 \times 4 \times 4$	57	42
$512 = 8 \times 8 \times 8$	117	52
$1728 = 12 \times 12 \times 12$	199	56
$4096 = 16 \times 16 \times 16$	284	59

Table 4.29: Hose domain, $N_{el} = 16 \times 16 \times 16$, $\kappa = 1$.



Chapter 5

Implementation

In this Chapter we overview the implementation of the presented Preconditioner. As stated before, the code is written in C programming language and utilizes the PETSc library for all the purposes beyond an Isogeometric Analysis framework: sparse matrix and vector parallel storage, iterative solvers for Krylov methods with condition number estimates and communication for distributed-memory parallelism.
The PETSc library has its own error checking system based on the integer output variable `ierr`, checked after each function call with `CHKERRQ(ierr);`. This function has been removed from the presented code for the sake of visual clearity, while the output integer `ierr` is substituted with `....` whenever the called function is written by the author. We refer again to the official PETSc website [5] and the documentation [6] for an in-depth explanation of the library.
Core functions such as `Refinement` and `Op_Poisson` and all the functions under the hood are not explained for brevity.

5.1 The main program

The source code presented here is the main code for setting up a 3D scalar elliptic problem over local subdomains refined in $16 \times 16 \times 16$ elements. NURBS cubic basis functions of maximum regularity are computed for the discretization.
The number of subdomains is set with the usual in-line parameter of the command[1] `mpiexec`, provided the grid subdivision with the in-line option `-grid` and the filename of a text file containing the geometry (with the option `-fname`):

```
> mpiexec -np 64 ./main -grid 4,4,4 -fname hose.txt
```

[1] With the bash script language for the Linux environment.

As it will be clear, minor changes are needed in order to change the dimension, the geometry or the differential problem. Here are the first lines of the main program:

```
1  // problem dimension
2  #define DIM 3
3
4  // PETSc header
5  #include"petscksp.h"
6  // headers
7  #include"utils.h"
8  #include"elements.h"
9  #include"refinement.h"
10 #include"functions.h"
11 #include"operators.h"
12 #include"coarse.h"
13
14 // communicator
15 ProcessID id;
```

The very first part of the code is the definition of the dimension `DIM` of the differential problem as it is constant throughout the entire code. Any positive dimension is feasible as any function related to the Isogeometric framework is dimension-free. Some specific part of the code, such as the implementation of the Jacobians inversions and the determinants computations, depends explicitly on the dimension as a true dimension-free implementation is inefficient and beyond the purpose of the code.
The headers are then included, with `petscksp.h` the main header for the entire PETSc library. The other headers contain the author's code.
In line 15 we declare the global variable `id` as an instance of the following self explanatory structure:

```
typedef struct {
  MPI_Comm       comm;             // MPI communicator
  PetscMPIInt    size[DIM], Size, // number of processes
                 rank[DIM], Rank; // process rank
} ProcessID;
```

As a convention throughout the code, lower case integers are arrays representing multiindexes in $\mathbb{N}^D$, while uppercase integers are multiindexes in $\mathbb{N}$, as explained in Section 2.2. The variable `id` is used by every function of the code, as there is only the global communicator. Then, we define the main function and the variables:

```
17 // main
18 int main(int argc, char **argv) {
19
20   PetscErrorCode  ierr;
21   PetscInt        idim;
```

```
22
23      // rite of passage
24    ierr = PetscInitialize(&argc, &argv, PETSC_NULL, PETSC_NULL);
25    id.comm = PETSC_COMM_WORLD;
26    .... = ProcessGrid();
27
28    NURBS         grid,          // starting geometry
29                  fine_grid,     // fine grid geometry
30                  coarse_grid;   // coarse grid geometry
31    LinearSystem  fine_system,   // fine local linear system
32                  coarse_system; // coarse global linear system
33    KSP           ksp,           // KSP main context
34                  *lksp;         // KSP local context
35    PC            pc,            // preconditioners contexts
36                  pc1,           // first level preconditioner
37                  pc2,           // second level preconditioner
38                  lpc;           // local system solver
39    IS            is;            // index set for overlap
40    ShellPC       *shell;        // second level context
41    Mat           P;             // projection operator
42
43    PetscReal     emax, emin;    // max/min eigenvalue
44    PetscInt      it;            // number of iterations
```

Lines 24-26 are the standard initialization of the MPI environment (handled by PETSc) and the computation of all the fields of `id`.
For consistency with PETSc, the primitive types `int`, `double` are replaced with the PETSc-defined ones, `PetscInt`, `PetscScalar`. The types `KSP`, `PC`, `IS`, `ShellPC` and `Mat` are PETSc objects whose purpose will be clear in the next lines of the `main`.
The type `NURBS` is a structure containing all the informations about the NURBS geometry, the grid and the parallel/serial setting. The knot vectors and control points are implemented with the minor structures `KnotVector` and `ControlPoint` that encapsulate appropriate `double` arrays, while the `Stencil` is a structure with information of the overlap of each subdomain along the `DIM` directions. Here is the `NURBS` structure definition:

```
typedef struct {
    // NURBS GEOMETRY INFORMATION
  char          name[PETSC_MAX_PATH_LEN]; // nurbs name
  PetscBool     distributed,    // distributed flag
                refined;        // refined flag
  PetscInt      nel[DIM], Nel,  // local number of elements
               tnel[DIM], tNel, // total number of elements
                deg[DIM],       // degrees
                reg[DIM],       // regularity
                ncp[DIM], Ncp,  // local number of ctrl points
               tncp[DIM], tNcp; // total number of ctrl points
```

```
  // DISCRETE SPACE INFORMATION
  PetscInt      nqn[DIM], Nqn,  // num of quad nodes per el
                nbf[DIM], Nbf;  // num of nonzero fncts per el

  // DOF INDEXING INFORMATIONS
  PetscInt      dof,            // num of dofs per ctrl point
                Ndof,           // local number of dofs
                tNdof,          // total number of dofs
                *ncon[DIM],     // ctrl pts natural ordering
                *pcon,          // dof parallel ordering
                Ngst,           // number of ghosted control pts
                *gst;           // ghosted dof connectivity
  PetscBool     *bdr;           // boundary control points flag
  Stencil       *layout;        // global overlap stencil

  // KNOT VECTORS AND CONTROL POINTS
  KnotVector    kv[DIM];        // knot vectors
  ControlPoint  *cp;            // control points
  PetscInt      span[DIM][2];   // index of span knot
} NURBS;
```

Finally, the structure `LinearSystem` is a simple container of the three objects of a linear system:

```
typedef struct {
  Mat           A;   // system matrix
  Vec           b,   // right hand side
                x;   // solution
} LinearSystem;
```

Once the main variables are declared, we can start with the construction of the grids:

```
46  // loading the starting geometry
47  .... = GeometryReader(&grid);
48
49  // creating the fine grid
50  fine_grid.dof = 1;
51  for (idim=0; idim<DIM; idim++) {
52    fine_grid.nel[idim] = 16;
53    fine_grid.deg[idim] = 3;
54    fine_grid.reg[idim] = 2;
55  }
56  .... = Refinement(&grid, &fine_grid, PETSC_TRUE);
57
58  // creating the coarse grid
59  coarse_grid.dof = 1;
60  for (idim=0; idim<DIM; idim++) {
61    coarse_grid.nel[idim] = id.size[idim];
```

```
62      coarse_grid.deg[idim] = 3;
63      coarse_grid.reg[idim] = 2;
64    }
65    .... = Refinement(&grid, &coarse_grid, PETSC_FALSE);
66    .... = Projection(&coarse_grid, &fine_grid, &P);
```

In line 47 we call the function that loads the text file `hose.txt` specified by the in-line command `-fname`. In lines 49-56 we first set the number of degrees of freedom per basis function (i.e. 1 for scalar problems), then we set the desired amount of refinement of the `fine_grid` along each direction. After the fields are filled, `fine_grid` can be refined from `grid` by a call of the function `Refinement`. The last input value of this function is a flag for specifying that `fine_grid` has to be refined in a parallel fashion, the domain decomposition setting being deduced from the global variable `id`. The refinement process is local, which means that only the local knot vectors and control points associated to local subdomains are computed and stored. Each MPI process owns different data.
The `coarse_grid` is created in the same way. The only difference is that now the flag for `Refinement` is false, and the refinement is not parallel and local but global. Each MPI process owns the same data. Notice that the number of elements is exactly the tensor grid for the processes.
Lastly, from the `coarse_grid` and the `fine_grid` the projection operator `P` is constructed by the function `Projection`. We refer to Section 5.2 for a deeper analysis.
We can now assemble the two linear system, the only difference between a local fine problem and a global coarse problem being the call of the preallocation functions `SystemCreator` and `SystemCreatorSeq`:

```
68    // creating the fine and coarse linear systems
69    .... = SystemCreator(&fine_grid, &fine_system);
70    .... = Op_Poisson(&fine_grid, &fine_system, &f, &k);
71
72    .... = SystemCreatorSeq(&coarse_grid, &coarse_system);
73    .... = Op_Poisson(&coarse_grid, &coarse_system, &f, &k);
```

The function `Op_Poisson` wants a `NURBS`, a preallocated `LinearSystem` and two pointers to user-provided functions for the source term `f` and the diffusivity coefficient `k`, whose declarations are the following:

```
PetscScalar f(PetscScalar *coord);
PetscScalar k(PetscScalar *coord);
```

The function `Op_Poisson` does not explicitly depends on the locality/globality of the grid because it is implemented as a classical element loop. On each element, an element stiffness matrix is computed from the nonzero basis function over the element, and then it is given to the PETSc routines for the global assembling,

along with the local-to-global index connectivity stored in the `pcon` field of the `NURBS` variable. Whether the `NURBS` grid is local or global, the element loop is the same. As we considered only homogeneous Dirichlet boundary conditions, there is no need to specify a boundary function. Degrees of freedom associated to the boundary are set to `0.0` during the assemble of the right hand side. For the element stiffness matrices associated to boundary elements, off-diagonal entries of rows and columns of boundary degrees of freedom are set to `0.0` so that there is no need to a posteriori modifications to the global matrix.
Each MPI process has now available its own local grid in `fine_grid`, its own local problem in `fine_system`, the global grid in `coarse_grid` and the global problem `coarse_system`. It is time to construct the PETSc machinery for the preconditioned iterative solver and the preconditioner itself:

```
75  // creating the KSP context
76  ierr = KSPCreate( id.comm, &ksp);
77  ierr = KSPSetOperators(ksp, fine_system.A, fine_system.A);
78  ierr = KSPSetType(ksp, KSPCG);
79  ierr = KSPSetComputeSingularValues(ksp, PETSC_TRUE);
80  ierr = KSPSetTolerances(ksp, 1.e-7, PETSC_DEFAULT,
        PETSC_DEFAULT, 1000);
81
82  // creating the main preconditioner context
83  ierr = KSPGetPC(ksp, &pc);
84  ierr = PCSetType(pc, PCCOMPOSITE);
85  ierr = PCCompositeSetType(pc, PC_COMPOSITE_ADDITIVE);
```

In lines 76-77 the iterative solver context `ksp` is created and associated to the global communicator stored in `id.comm` and to the `fine_system` matrix. It is then set to use the Preconditioned Conjugate Gradient method with the flag `KSPCG` (line 78), to compute an estimation of the maximum and minimum eigenvalues (line 79) and finally set with a preconditioned residual tolerance of 10^{-7} with 1000 maximum iterations (line 80).
Each `KSP` object contains a preconditioner object `PC`. We extract it in line 83, set it as a `PCCOMPOSITE`, which means that it will be composed of two distinct sub-preconditioner (the 1-level ASO and the coarse grid correction), and set to manage in an additive fashion the sub-preconditioners.
We now need to define the first sub-preconditioner:

```
87  // setting up the first level of preconditioning
88  ierr = PCCompositeAddPC(pc, PCASM);
89  ierr = PCCompositeGetPC(pc, 0, &pc1);
90  ierr = ISCreateGeneral(id.comm, fine_grid.Ndof, fine_grid.pcon
        , PETSC_COPY_VALUES, &is);
91  ierr = PCASMSetLocalSubdomains(pc1, 1, &is, PETSC_NULL);
92  ierr = PCASMSetType(pc1, PC_ASM_BASIC);
93  ierr = PCASMGetSubKSP(pc1, PETSC_NULL, PETSC_NULL, &lksp);
```

```
94    ierr = KSPGetPC(lksp[0],&lpc);
95    ierr = PCSetType(lpc,PCLU);
```

In line 88 we add an Additive Schwarz preconditioner to the main composite preconditioner list. This is the black-box implementation of the 1-level ASO with overlap defined with Strategy 3. Different flavors exist, and we select the basic one in line 92. In order to set `pc1` to handle the overlap with Strategy 2, we need to define a distributed parallel set of indexes `IS`. We do this in line 90 directly with the `pcon` field of the `fine_grid`. Accordingly with the degrees of freedom distribution stored in `is`, the first sub-preconditioner `pc1` is set in line 91. The last three line of code for this part extract the `KSP` context `lksp` for the local solvers of the local problem, extract the local preconditioner `lpc` and set it to LU factorization. These are necessary steps for exact direct solving of the local problems, as PETSc uses incomplete LU factorization as default local solver.
The code so far is the actual implementation of the 1-level ASO, whose results are shown in Chapter 4. The coarse grid correction, or the second level of domain decomposition, is set with these lines of code:

```
97    // setting up the second level of preconditioning
98    ierr = PCCompositeAddPC(pc, PCSHELL);
99    ierr = PCCompositeGetPC(pc, 1, &pc2);
100   .... = ShellPCCreate(&shell);
101   ierr = PCShellSetContext(pc2, shell);
102   .... = ShellPCSetUp(pc2, &coarse_grid, &fine_grid,
          coarse_system.A, P, 1);
103   ierr = PCShellSetApply(pc2, ShellPCApply);
```

Again, being the `pc` of `PCCOMPOSITE` type, we add another sub-preconditioner of type `PCSHELL`. This kind of PETSc preconditioner is only an interface, and it is up to the user to define the action of the matrix it represents. It can be thought as an abstract C++ class that needs a derived implementation. Once added the `PCSHELL` preconditioner and extracted its pointer into `pc2` (lines 98,99), we construct the "derived class" `shell` with a call of the function `ShellPCCreate` (line 100), and then we pass it to `pc2` (line 101). For now we just set the pointers of the infrastructure. The real construction of the `PCSHELL` preconditioner `pc2` is demanded to the function `ShellPCSetUp`. Finally, we pass to the preconditioner the function that implements the action of the coarse grid correction: `ShellPCApply`.
We are now ready to conclude the main program:

```
105   // completing the set up
106   ierr = KSPSetUp(ksp);
107   ierr = PCSetUp(pc1);
108   ierr = PCSetUp(pc2);
109
110   // solve the system
```

```
111   ierr = KSPSolve(ksp,fine_system.b,fine_system.x);
112
113   // collect the results
114   ierr = KSPComputeExtremeSingularValues(ksp, &emax, &emin);
115   ierr = KSPGetIterationNumber(ksp, &it);
116
117   // cleaning up
118   ierr = ISDestroy(&is);
119   ierr = PCDestroy(pc2);
120   ierr = MatDestroy(&P);
121   ierr = KSPDestroy(&ksp);
122   .... = SystemDestroyer(&fine_system);
123   .... = SystemDestroyer(&coarse_system);
124   .... = GeometryDestroyer(&fine_grid);
125   .... = GeometryDestroyer(&coarse_grid);
126   .... = GeometryDestroyer(&grid);
127
128   // rite of passage
129   ierr = PetscFinalize();
130   return 0;
131 }
```

As it is self-explanatory, in lines 105-107 we finalize the setup of the `KSP` contexts along with its sub-preconditioners, in line 111 we solve the system, in lines 113,114 we extract the number of iterations required to reach convergence and the estimated minimum and maximum eigenvalues, and finally in lines 117-125 we deallocate everything for a clean termination of the program. Lines 128-130 don't require a single word.

5.2 The projection operator

In this Section we present an overview of the function `Projection` called in line 66 of the main source code of Section 5.1. For brevity the innermost functions are omitted. As already noticed, the projection operator depends only on the `fine_grid` and the `coarse_grid`: the `NURBS` structure contains all the data required for the computation. The projection operator will be stored as a rectangular dense PETSc serial matrix.
As common practice in implementations of domain decomposition methods and in parallel sparse matrices storage, the system matrix `fine_system.A` is owned by rows. This means that each processor owns a non overlapping subset of all the rows of the matrix, i.e. it owns a non overlapping subset of degrees of freedom. On processor with index `id.Rank`, the projection `P` is the refinement operator described in Section 2.3.1, acting from the $(p+1)^D$ nonzero coarse basis functions of the element number `iel=id.Rank` to the non overlapping fine basis functions

owned by `id.Rank`. As the B-spline (or NURBS) basis functions overlap across subdomains, the degrees of freedom on the interface require the missing data from neighboring subdomains. This communication is handled by `ShellPCApply`, as we will see in the next section.

We can now present the first part of the construction of `P`:

```
1  PetscErrorCode Projection(NURBS *coarse_grid, NURBS *fine_grid,
       Mat *P) {
2
3    PetscErrorCode  ierr;
4    PetscInt    idim,         // index for dimension loop
5                irow, icol,   // indexes for matrix loops
6                nrow[DIM],    // rows of unirefing matrices
7                ncol[DIM],    // columns of unirefining matrices
8                Nrow,         // rows of complete unirefining mat
9                Ncol,         // cols of complete unirefining mat
10               nknt,         // number of knots to insert
11               frow, fcol;   // indexes for submatrix extraction
12   PetscScalar a, b,         // extremas of subdomain interbal
13               *knt,         // knots to insert
14               *r,           // complete unirefining matrix
15               *R[DIM];      // extracted unrefining matrices
16
17   for (idim=0; idim<DIM; idim++) {
18     // searching knots to insert
19     a = coarse_grid->kv[idim].u[ id.rank[idim] ];
20     b = coarse_grid->kv[idim].u[ id.rank[idim]+1 ];
21     .... = ExtractKnots(coarse_grid->kv[idim], fine_grid->kv[
           idim], a, b, &nknt, knt);
22
23     // calcuating complete unirefining matrix r
24     Nrow = coarse_grid->ncp[idim];
25     Ncol = coarse_grid->ncp[idim]+nknt;
26     ierr = PetscMalloc(Nrow*Ncol*sizeof(PetscScalar), &r);
27     .... = UniRefinementMatrix(&coarse_grid->kv[idim], knt, nknt
           , coarse_grid->deg[idim], r);
28
29     // creating local unirefining matrix R:
30     nrow[idim] = coarse_grid->deg[idim]+1;
31     ncol[idim] = fine_grid->layout[id.Rank].stncl[idim][1];
32     ierr = PetscMalloc(nrow[idim]*ncol[idim]*sizeof(PetscScalar)
           ,&R[idim]);
33
34     // searching first index for rows and columns
35     ierr = FindFirstIndexes(coarse_grid, a, b, &frow, &fcol);
36
37     // extracting the local unirefining matrix
38     for (irow = 0; irow<nrow[idim]; irow++) {
39       for (icol = 0; icol<ncol[idim]; icol++) {
```

```
40          R[idim][irow*ncol[idim]+icol] = r[(frow+irow)*Ncol+fcol+
                icol];
41        }
42      }
43
44      // cleaning up
45      ierr = PetscFree(knt);
46      ierr = PetscFree(r);
47    }
```

After the usual declaration of the working variables, we initiate a loop over the dimension `DIM` of the problem. In this first part we compute the `DIM` univariate projection operators acting between the coarse and fine knot vectors. In lines 19-21 we compute from the (global) coarse knot vector the extreme values of the interval $\hat{I}_j$ defined by the interface knots as in 3.22. The field `kv.u` is an array of `doubles` storing the non repeated knots, so that the extraction of the extreme values of the interval can be performed straightforwardly with the processor's multi-index `id.rank[idim]`. We then proceed to compute the knots that needs to be added to the coarse knot vector in order to obtain the fine knot vector. Some more knots before `a` and after `b` are required and automatically computed accordingly with the degree `deg[idim]`.
In lines 24-27 we compute the refining matrix acting from the global coarse knot vector to the coarse knot vector plus the local fine knots in the specific subdomain owned by the processor. We store the result in the previously allocated array `r`.
The refining matrix `r` has a global scope, hence we need to extract the submatrix of interest. The number of rows is the number of nonzero basis function per element in the coarse grid, i.e. $p+1$ (line 30), while the number of columns is the number of non overlapping basis functions owned by the processor (line 31). This information is stored inside the array `layout` of `Stencil`. This is its declaration:

```
typedef struct {
  PetscInt    stncl[DIM][3];     // overlap stencil
                                 // stncl[idim][0]: left ovrlap
                                 // stncl[idim][1]: owned dofs
                                 // stncl[idim][2]: right ovrlap
} Stencil;
```

Each `stncl[idim]` represent the degrees of freedom layout of a particular processor along dimension `idim`. A tensor product treatment of this data gives the multivariate layout of all degrees of freedom of all processor. It is now clear that the non overlapping number of univariate basis functions owned by processor `id.Rank` is `stncl[idim][1]`. We know the size of the submatrix to be extracted but not its position inside `r`, therefore a call of `FindFirstIndexes` is required (line 35). The first row index `frow` and the first column index `fcol`

can be automatically computed with an appropriate exploit of the global variable `id` and the field `layout` containing the `Stencils` of each processor. Once we get this informations, the extraction into the previously allocated dense matrix `R[idim]` follows in lines 38-42.
The following code is the second part of the function `Projection`:

```
49    // calculating tensor product of unirefining matrices
50    ierr = Tensorizer(R, nrow, ncol, P);
51
52    // scaling with control weights
53    Vec lD, rD;          // diagonals of scaling matrices
54    PetscInt    indexes; // insertion indexes
55    PetscScalar weights; // weights
56
57    ierr = MatGetSize(*P,&Nrow,&Ncol);
58    ierr = PetscMalloc(max(Nrow,Ncol)*sizeof(PetscInt),&indexes);
59    ierr = PetscMalloc(max(Nrow,Ncol)*sizeof(PetscScalar),&weights
          );
60    for (irow=0; irow<max(Nrow,Ncol); irow++) {
61      indexes[irow]=irow;
62    }
63
64    // building left diagonal scaling matrix (as vector)
65    .... = ExtractCoarseWeights(coarse_grid, weights);
66    ierr = VecCreateSeq(PETSC_COMM_SELF, Nrow, &lD);
67    ierr = VecSetValues(lD, Nrow, indexes,weights, INSERT_VALUES);
68    ierr = VecAssemblyBegin(lD);
69    ierr = VecAssemblyEnd(lD);
70    // building right diagonal scaling matrix (as vector, again)
71    .... = ExtactFineWeights(fine_grid, weights);
72    ierr = VecCreateSeq(PETSC_COMM_SELF, Ncol, &rD);
73    ierr = VecSetValues(rD, Ncol, indexes,weights, INSERT_VALUES);
74    ierr = VecAssemblyBegin(rD);
75    ierr = VecAssemblyEnd(rD);
76
77      // scaling the matrix
78    ierr = MatDiagonalScale(*P, lD, rD);
79
80      // cleaning up
81    for (idim = 0; idim<DIM; idim++) {
82      ierr = PetscFree(R[idim]);
83    }
84    ierr = VecDestroy(&lD);
85    ierr = VecDestroy(&rD);
86    ierr = PetscFree(weights);
87    ierr = PetscFree(indexes);
88
89    return ierr;
90  }
```

In the line 50 we call the function `Tensorizer`. It applies the Kronecker product the `DIM` univariate refining matrices `R[idim]` into the PETSc serial dense matrix `P`. This procedure would suffice for B-spline basis functions, but for NURBS the projection operator `P` needs a left and right scaling by respectively the coarse and fine weights. The refining matrix operates exactly between B-spline spaces, so that in order to act correctly on NURBS basis functions the fine control weights have to be removed with a right-multiplication by the diagonal matrix of the reciprocal of the fine weights. Then the weights are restored with a left-multiplication by the diagonal matrix of the coarse weights. The weight function is not involved in this scaling procedure as it is a B-spline function and hence it is refinement independent.
Luckily PETSc provides a function that scales both sides of a rectangular matrix given two `Vecs`. In lines 65-69 the coarse weights are extracted and the `Vec` object `lD` is created and assembled (with some minor machinery in lines 57-62). The same procedure is performed in lines 71-75 with the exception that `weights` now stores the reciprocal of the fine weights. We can now scale the projection operator `P`, clean up the memory and terminate the `Projection` function.

5.3 The shell preconditioner

In this Section we present the source code that implements the action of the second level of the 2-level ASO. The function `ShellPCApply` is devoted to project the global residual from the fine space to the coarse space, solve the coarse problem and then reproject it back into the fine space. The preconditioner is implemented on the top of the following structure:

```
typedef struct {

          // 0 |_|_|_|_|X|X|X|X|_|_|_|_|   parallel X
  Vec sX, // 1         |X|X|X|X|           sequential X
      lXc, // 2          |X|X|             local Xc=P*X
      gXc, // 3       |x|x|X|X|x|x|        global Xc
      Xc,  // 4       |X|X|X|X|X|X|        Xc
      Yc,  // 5       |Y|Y|Y|Y|Y|Y|        Yc=Ac^-1 * Xc
      lYc, // 6           |Y|Y|            local Yc
      sY;  // 7         |Y|Y|Y|Y|          sequential Y=P^T*Yc
           // 8 |_|_|_|_|Y|Y|Y|Y|_|_|_|_|   parallel Y

  PetscInt     nrow,   // number of rows of P
               ncol,   // number of columns of P
               ndof;   // size of Ac
  Mat          Ac,     // coarse system matrix
               P;      // projection operator
  KSP          ksp;    // context for coarse problem
```

```
  PC           pc;     // precondtioner for crse prb
  VecScatter   sct;    // all-to-all communication
  PetscInt    *lind,   // local indexes
              *gind,   // global indexes
              *ind;    // working variable
  PetscScalar *val;    // working variable
} ShellPC;
```

The fancy ASCII picture on the right of the `Vec` declarations represents the work-flow required by PETSc in order to apply the coarse grid correction. PETSc makes a determinant distinction between serial (`Seq`-type) objects and parallel (`MPI`-type) objects, so that for example a sequential usage of a parallel object needs a hand-made conversion from one type to another. This motivates the following constructions of all the `Vec` objects and the all-to-all communication context of the instance `s` of the `ShellPC` (called in the function `ShellPCSetUp`)

```
  ierr = VecCreateSeq(PETSC_COMM_SELF, ncol, &s->sX);
  ierr = VecCreateSeq(PETSC_COMM_SELF, nrow, &s->lXc);
  ierr = VecCreateMPI(id.comm, PETSC_DECIDE, ndof, &s->gXc);
  ierr = VecScatterCreateToAll(s->gXc, &s->sct, &s->Xc);
  ierr = VecCreateSeq(PETSC_COMM_SELF, ndof, &s->Yc);
  ierr = VecCreateSeq(PETSC_COMM_SELF, nrow, &s->lYc);
  ierr = VecCreateSeq(PETSC_COMM_SELF, ncol, &s->sY);
```

Indexes arrays `lind` and `gind` also are allocated and computed during a call of `ShellPCSetUp`. We explain the machinery step by step keeping in mind the notation used in the name variables: `X` is the global input vector while `Y` is the global output. They respectively store the residual of the Conjugate Gradient iteration before and after the coarse correction grid. The prefix `l` as in `lXc` stands for local while `g` for global. The suffix `c` as in `gXc` means that the vector lives in the coarse space. Before the work-flow explanation, for the sake of completeness we present the declaration of the function `ShellPCApply`:

```
1  PetscErrorCode ShellPCApply(PC pc,Vec X,Vec Y) {
2
3    // PETSc rite of passage
4    PetscErrorCode ierr;
5    ShellPC  *s;
6    ierr  =  PCShellGetContext(pc,(void**)&s);
7    // working variables
8    PetscInt nrow = s->nrow,  // rows of
9             ncol = s->ncol;  // columns of P
10   PetscScalar *val, *lxc, *yc, *sy; // working variables
11   const PetscScalar *x;             // working variable
12   PetscInt    *ind;                 // working variable
13   PetscInt    idof;                 // working variable
```

We are now ready for the work-flow explanation:

1. From the global parallel vector `X` we extract the local portion owned by the processor and we store it into the sequential vector `sX`.

```
15    // STEP 1
16    ierr = VecGetArray(X, &x);
17    ierr = VecSetValues(s->sX, ncol, ind, x, INSERT_VALUES);
18    ierr = VecAssemblyBegin(s->sX);
19    ierr = VecAssemblyEnd(s->sX);
20    ierr = VecRestoreArray(X, &x);
```

2. We project the sequential portion `sX` with a multiplication by the matrix `P`, obtaining the local coarse vector `lXc`. It is addressed as local because we have used local data only.

```
22    // STEP 2
23    ierr = MatMult(s->P, s->sX, s->lXc);
```

3. The local coarse vector `lXc` is then copied into the global coarse vector `gXc`. This step is required in order to gather the remaining coefficients from the overlapping degrees of freedom.

```
25    // STEP 3
26    ierr = VecGetArray(s->lXc, &lxc);
27    ierr = VecSet(s->gXc, 0.0);
28    ierr = VecSetValues(s->gXc, nrow, s->lind, lxc, ADD_VALUES
          );
29    ierr = VecAssemblyBegin(s->gXc);
30    ierr = VecAssemblyEnd(s->gXc);
31    ierr = VecRestoreArray(s->lXc, &lxc);
```

4. The vector `gXc` is a parallel object, so we can use it as source for the all-to-all communication via the `VecScatter` PETSc object, obtaining the sequential global coarse vector `Xc`. Each processor has a copy of this vector.

```
33    // STEP 4
34    ierr = VecScatterBegin(s->sct, s->gXc, s->Xc,
          INSERT_VALUES, SCATTER_FORWARD);
35    ierr = VecScatterEnd(s->sct, s->gXc, s->Xc, INSERT_VALUES,
           SCATTER_FORWARD);
```

5. We can now solve the coarse linear system, obtaining the solution in the sequential coarse vector `Yc`.

```
37    // STEP 5
38    ierr = KSPSolve(s->ksp, s->Xc, s->Yc);
```

6. Each processor owns the projection operator from the coarse element to the fine subdomain, hence we need to extract from the coarse corrected vector `Yc` the portion associated to the corresponding element. We obtain the local sequential vector `lYc`.

```
40    // STEP 6
41    ierr = VecGetArray(s->Yc, &yc);
42    for (idof=0; idof<nrow; idof++) {
43      val[idof] = yc[ s->lind[idof] ];
44    }
45    ierr = VecSetValues(s->lYc, nrow, ind, val, INSERT_VALUES)
          ;
46    ierr = VecAssemblyBegin(s->lYc);
47    ierr = VecAssemblyEnd(s->lYc);
48    ierr = VecRestoreArray(s->Yc, &yc);
```

7. The local sequential vector `lYc` is projected back into the fine space. The result of the projection is stored in the sequential vector `sY`.

```
50      // STEP 7
51    ierr = MatMultTranspose(s->P, s->lYc, s->sY);
```

8. Each processor has finally its portion of the final coarse grid corrected vector `Y`, but stored in the sequential vector `sY`. We construct `Y` as a standard PETSc `Vec`. This last procedure does not require communication as it has already been performed in Step 4.

```
53      // STEP 8
54    ierr = VecGetArray(s->sY, &sy);
55    ierr = VecSetValues(Y, ncol, s->gind, sy, INSERT_VALUES);
56    ierr = VecAssemblyBegin(Y);
57    ierr = VecAssemblyEnd(Y);
58    ierr = VecRestoreArray(s->sY, &sy);
59
60    return ierr;
61  }
```

This conclude the PETSc work-flow for a coarse correction grid with the exact isogeometric projection operator `P`.

5.4 On the computational times

The good analytic performance of the 2-level Additive Schwarz Preconditioner for the scalar equation has been established with the condition number estimation and the number of iterations. We present here some results about the computational times for solving the scalar elliptic equation. A selection of cases is considered, the most meaningful ones.
It is worth to notice that the actual time performance of an algorithm depends heavily on both the implementation and the hardware. While the author fully trusts the optimality and efficiency of PETSc, he does not ensure that the workflow presented in Section 5.3 of the core function `ShellPCApply` is the best implementation of the coarse correction grid. We disclaim that the following results are meaningful up to a certain point. Nevertheless, the 2-level Preconditioner outperforms the 1-level one in each test, even though for few percentage points in the 3D case. An accurate analysis of the execution time is beyond the purpose of this work.
Implementation efficiency apart, the iteration of the Conjugate Gradient Method with the 1-level Preconditioner involves MPI communication and the local problem solving. The same iteration with the 2-level Preconditioner requires the same amount of work plus the global construction of `gXc` (Step 3 in Section 5.3), the all-to-all communication for `Xc` (Step 4) and the local solving of the coarse linear problem (Step 5). Whenever the coarse grid size is comparable with the local grid size, one would expect that a 2-level preconditioned iteration requires roughly double the time of a 1-level preconditioned one, plus the most time-consuming work for a distributed memory parallel program: communication. For this reason, we measured the absolute execution time t_{tot} and the time per iteration t_{it} as simple ratio of absolute time and number of iterations.
Before commenting the actual the execution time of the solvers, we present the timing procedure with few lines of code:

```
begin = clock();
ierr = KSPSolve(ksp, fine_system.b,fine_system.x);
end = clock();
time = (double)(end-begin) / CLOCKS_PER_SEC;
```

Even though the measure is local, `KSPSolve` has synchronization barriers within, so that the variable `time` is roughly the same on each processor[2]. Like throughout this entire work, full circles in the graphs represent the 1-level Preconditioner while the empty ones represent the 2-level Preconditioner.

[2]The author has checked load balance by comparison of the minimum and maximum execution time, obtaining the ratio 1.03 for the most unbalanced test, so that load balancing is considered as fully achieved.

2D domains, diffusivity coefficient $k = 1$

		Square domain						Annulus domain					
		1-lvl			2-lvl			1-lvl			2-lvl		
p	N_{dom}	it.	t_{tot}	t_{it}	it.	t_{tot}	t_{it}	it.	t_{tot}	t_{it}	it.	t_{tot}	t_{it}
	16	28	0.98	0.04	18	0.90	0.05	57	1.24	0.02	37	1.12	0.03
	64	42	1.12	0.03	14	0.87	0.06	116	1.80	0.02	45	1.24	0.03
2	256	67	1.35	0.02	12	0.87	0.07	234	2.90	0.01	33	1.13	0.03
	1024	116	1.80	0.02	10	0.90	0.09	463	5.01	0.01	28	1.21	0.04
	4096	210	2.68	0.01	7	1.04	0.15	796	8.12	0.01	25	1.76	0.07
	16	25	2.39	0.10	17	2.32	0.14	49	2.76	0.06	28	2.52	0.09
	64	37	2.64	0.07	16	2.37	0.15	101	3.64	0.04	31	2.66	0.09
3	256	59	3.00	0.05	12	2.33	0.19	202	5.23	0.03	26	2.60	0.10
	1024	102	3.67	0.04	9	2.33	0.26	404	8.39	0.02	23	2.70	0.12
	4096	187	5.02	0.03	7	2.56	0.37	688	12.85	0.02	23	3.49	0.15
	16	24	5.04	0.21	17	4.94	0.29	45	5.54	0.12	27	5.21	0.19
	64	34	5.32	0.16	17	4.99	0.29	90	6.69	0.07	31	5.39	0.17
4	256	54	5.82	0.11	14	4.94	0.35	179	8.88	0.05	26	5.31	0.20
	1024	92	6.75	0.07	8	4.83	0.60	359	13.27	0.04	18	5.22	0.29
	4096	168	8.64	0.05	7	5.18	0.74	618	19.66	0.03	21	6.36	0.30

Table 5.1: 2D domains, $N_{el} = 4096 = 64 \times 64$, $k = 1$

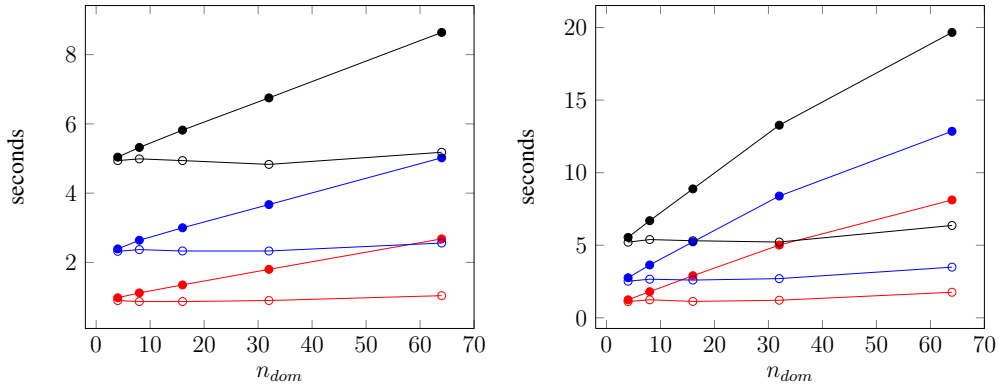


Figure 5.1: Square domain on the left, annulus domain on the right, $N_{el} = 4096 = 64 \times 64$, $k = 1$

Table 5.1 shows that the 2D scalar elliptic problem discretized with B-splines (square domain) or NURBS (annulus domain) is solved in almost scalable times whenever the 2-level Preconditioner is used. As predicted, the seconds per iteration t_{it} slightly increase with the number of subdomains because the coarse grid linear problem becomes larger and larger.
For comparison, the performance of the 1-level Preconditioner deteriorates with the number of subdomains and the execution time raises to the point that a quartic NURBS-based problem with the 2-level preconditioner requires less computational time of a quadratic NURBS-based one with the 1-level one. For what concern the seconds per iteration t_{it} measures of the 1-level Preconditioner, the author has no clear explanation apart from PETSc admirable efficiency.
Table 5.2 shows the execution times t_{tot} and the seconds per iteration t_{it} for scalar problem on the hose domain. Again, the 2-level Preconditioner shows good scalability. The seconds per iteration are more stable, and we suppose that the communication latency of the supercomputer has a key role in dominating the overall execution time, being the overlap (and hence the communication) enormously bigger than the 2D case. The 1-level Preconditioner, again, shows the black-box behavior analyzed for the 2D case.

Hose domain, diffusivity coefficient $k = 1$

p	$N_{dom} = n_{dom} \times n_{dom} \times n_{dom}$	1-level preconditioner			2-level preconditioner		
		it.	t_{tot}	t_{it}	it.	t_{tot}	t_{it}
$p = 2$	$64 = 4 \times 4 \times 4$	50	32.71	0.6542	31	31.63	1.0203
	$512 = 8 \times 8 \times 8$	92	35.42	0.3850	34	32.21	0.9474
	$1728 = 12 \times 12 \times 12$	137	38.34	0.2799	33	33.31	1.0094
	$4096 = 16 \times 16 \times 16$	184	41.24	0.2241	32	35.96	1.1237
$p = 3$	$64 = 4 \times 4 \times 4$	50	215.84	4.3168	29	206.15	7.1086
	$512 = 8 \times 8 \times 8$	100	228.79	2.2879	35	212.04	6.0583
	$1728 = 12 \times 12 \times 12$	117	233.23	1.9934	32	215.11	6.7222
	$4096 = 16 \times 16 \times 16$	157	243.55	1.5513	32	220.25	6.8828

Table 5.2: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 1$.

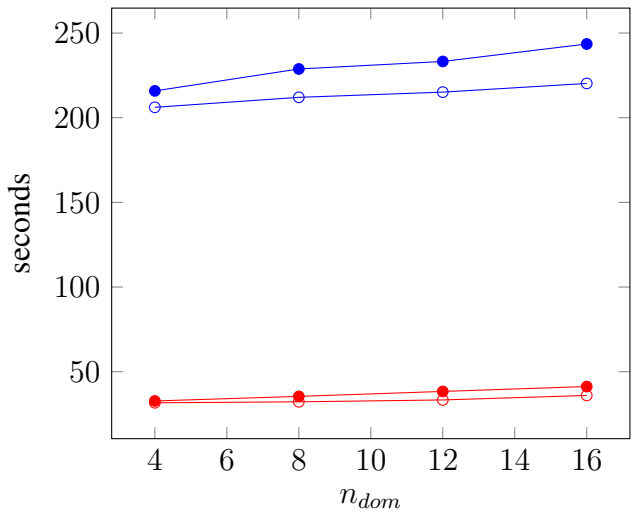


Figure 5.2: Quarter hose domain, $N_{el} = 4096 = 16 \times 16 \times 16$, $k = 1$.

Chapter 6

Conclusions

In this work we presented a multilevel Overlapping Additive Schwarz Preconditioner for the Isogeometric analysis of the scalar elliptic equations and the advection-diffusion equation. The preconditioner is endowed with a coarse correction grid in order to achieve scalability with respect to the number of subdomains involved in the decomposition. Theoretical results show that the condition number of the system matrix arising from an IGA discretization of the scalar elliptic equation is bounded by the ratio H/h, where H and h are respectively the coarse mesh size and fine mesh size. The result is proved to the full generality of the IGA framework: B-spline- and NURBS-based discrete functional spaces in both parametric and physical domain can be effectively used with the proposed Preconditioner.
Numerical results in massively parallel computational environments are presented. The 1-level Preconditioner is a black-box preconditioning technique with a reasonable performance up to few hundreds of subdomains. Higher numbers of subdomains are prohibitive for large scale simulations. Accordingly to the theory, the 2-level Preconditioner shows instead scalability for the scalar elliptic problem up to 4096 total subdomains for both 2D and 3D tests. The estimated condition number of the preconditioned matrix tends to a constant value, suggesting that the decomposition can be potentially pushed further with an even higher number of subdomains. The introduction of discontinuous diffusivity coefficients in the scalar elliptic equation deteriorates the condition number of the system matrix and the overall performance of the preconditioned iterative solver. It is particularly evident for the 1-level Preconditioner, while the 2-level one is still capable of maintaining an asymptotic trend for the condition number, thus achieving parallel scalability.
Although there are still no theoretical results, the 2-level Preconditioner is tested for the advection-diffusion equation as well, and compared again with the 1-level version. As long as the advection-diffusion problem is diffusion-dominated, the behavior of the 2-level preconditioner scales with the number of subdomains up

to 8100 processors, and outperform the 1-level one in both 2D and 3D tests. With more advection-dominated problems, the 2-level preconditioner presents difficulties in achieving convergence, especially for the 3D case, bringing the necessity of stabilization techniques such as the SUPG stabilization. Once the stabilization is plugged into the problem discretization, the 2-level preconditioner gains back stability and convergence. We suppose that these suboptimal behaviors are due to the mesh size related difficulties that are typical of the advection-diffusion equations. The coarse problem arises from a coarse mesh with a relatively large mesh size, so that the coarse correction grid may be sufficiently accurate only when dealing with large numbers of subdomains.
A way of dealing with the advection diffuculties may be found in moltiplicative and hybrid Domain Decomposition techniques, where the Schwarz operators $P_{\mathrm{j}} = R_{\mathrm{j}}^T A_{\mathrm{j}}^{-1} R_{\mathrm{j}}$ are not combined with a summation but with a productory or a mixture of the two. Non-overlapping Domain Decomposition methods such as the Balanced Domain Decomposition by Constraints Method (BDDC), already studied for scalar elliptic equations with isogeometric discretization in [10]) or FETI-DP Methods (already studied in [26]) can be considered and built for isogeometric discretization of the advection-diffusion equation.

Bibliography

[1] Fermi supercomputer. `http://www.hpc.cineca.it/content/fermi-reference-guide`.

[2] Mira supercomputer. `https://www.alcf.anl.gov/mira`.

[3] R.A. Adams. *Sobolev Spaces*. Academic Press, New York, 1975.

[4] A. Apostolatos, R. Schmidt, R. Wüchner, and K. Bletzinger. A Nitsche-type formulation and comparison of the most common domain decomposition methods in isogeometric analysis. *International Journal for Numerical Methods in Engineering*, 97:473–504, 2013.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2015.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[7] Y. Bazilevs, L. Beirão da Veiga, J.A. Cottrell, T.J.R. Hughes, and G. Sangalli. Isogeometric analysis: Approximation, stability, and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences*, 16(7):1031–1090, 2006.

[8] Y. Bazilevs, J.A. Cottrell, and T.J.R. Hughes. *Isogeometric Analysis - Towards Integration of CAD and FEA*. Wiley, 2009.

[9] L. Beirão da Veiga, D. Cho, L. F. Pavarino, and S. Scacchi. Overlapping Schwarz methods for isogeometric analysis. *SIAM Journal on Numerical Analysis*, 50(3):1394–1416, 2012.

[10] L. Beirão da Veiga, D. Cho, L.F. Pavarino, and S. Scacchi. BDDC Preconditioners for Isogeometric Analysis. *Mathematical Models and Methods in Applied Sciences*, 23:1099, 2012.

[11] L. Beirão da Veiga, D. Cho, L.F. Pavarino, and S. Scacchi. Isogeometric Schwarz preconditioners for linear elasticity systems. *Computer Methods in Applied Mechanics and Engineering*, 253:439–454, 2013.

[12] L. Beirão da Veiga, D. Cho, L.F. Pavarino, and S. Scacchi. Overlapping Schwarz preconditioners for isogeometric collocation methods. *Computer Methods in Applied Mechanics and Engineering*, 278:239–253, 2014.

[13] L. Beirão da Veiga, L.F. Pavarino, S. Scacchi, O. B. Widlund, and S. Zampini. Isogeometric BDDC preconditioners with deluxe scaling. *SIAM Journal on Numerical Analysis*, 36:1118–1139, 2014.

[14] M. Bercovier and I. Soloveichik. Overlapping non matching meshes domain decomposition method in isogeometric analysis. *arXiv preprint*, page arXiv:1502.03756, 2015.

[15] J. Bramble and S. Hilbert. Estimation of linear functionals on Sobolev spaces with application to Fourier transform and spline interpolation. *SIAM Journal on Numerical Analysis*, 7:112–124, 1970.

[16] A.N. Brooks and T.J.R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32:199–259, 1982.

[17] A. Buffa, H. Harbrecht, A. Kunoth, and G. Sangalli. BPX-preconditioning for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 265:63–70, 2013.

[18] L. Charawi. *Isogeometric overlapping Schwarz preconditioners in computational electrocardiology*. PhD thesis, Università degli Studi di Milano, 2014.

[19] L. Charawi. Isogeometric overlapping additive Schwarz solvers for the bidomain system. *Springer LNCSE, to appear*, 2015.

[20] N. Collier, L. Dalcin, D. Pardo, and V. M. Calo. The cost of continuity: Performance of iterative solvers on isogeometric finite elements. *SIAM Journal on Scientific Computing*, 35:A767–A784, 2013.

[21] M. Donatelli, C. Garoni, C. Manni, S. Serra-Capizzano, and H. Speleers. Robust and optimal multi-iterative techniques for IgA Galerkin linear systems. *Computer Methods in Applied Mechanics and Engineering*, 284:230–264, 2015.

[22] K. Gahalaut, J. Kraus, and S. Tomar. Multigrid methods for isogeometric discretization. *Computer Methods in Applied Mechanics and Engineering*, 253:413–425, 2013.

[23] C. Hesch and P. Betsch. Isogeometric analysis and domain decomposition methods. *Computer Methods in Applied Mechanics and Engineering*, 213-216:104–112, 2012.

[24] Q.-X. Huang, S.-M. Hu, and R.R. Martin. Fast degree elevation and knot insertion for B-spline curves. *Computer Aided Geometric Design*, 22(2):183–197, 2005.

[25] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194:4135–4195, 2005.

[26] S. K. Kleiss, C. Pechstein, B. Juttler, and S. Tomar. IETI - isogeometric tearing and interconnecting. *Computer Methods in Applied Mechanics and Engineering*, 247/248:201–215, 2012.

[27] J.T. Oden, I.B. Ska, and C.E. Baumann. A discontinuous hp finite element method for diffusion problem. *Journal of Computational Physics*, 146:491–519, 1998.

[28] L. Piegl and W. Tiller. *The NURBS Book (Monographs in Visual Communication), 2nd ed.* Springer-Verlag, New York, 1997.

[29] A. Quarteroni. *Numerical Models for Differential Problems*. Springer, 2008.

[30] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd ed.* Society for Industrial and Applied Mathematics, 2003.

[31] L.L. Schumaker. *Spline functions: basic theory*. Krieger, 1993.

[32] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition - Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[33] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*. Springer, 2005.

[34] J. Volker and L. Schumacher. A study of isogeometric analysis for scalar convection-diffusion equations. *Applied Mathematics Letters*, 27:43–48, 2014.