Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

TREBALL FINAL DE MÀSTER

Reduced order models for pollution transport (advection-diffusion) in urban areas

Treball realitzat per:
Claudia Álvarez Pujol

Dirigit per:
Pedro Díez Mejía
Alberto García González
Simona Perotto

Màster en:
Enginyeria de Camins, Canals i Ports

Barcelona, 14 de juny de 2019

Departament d'Enginyeria Civil i Ambiental

Reduced order models for pollution transport (advection-diffusion) in urban areas

by
Claudia Álvarez Pujol

Thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Civil Engineering

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Department of Civil and Environmental Engineering
Universitat Politècnica de Catalunya (UPC)
14th June, 2019

Acknowledgements

I would like to thank my thesis advisors, Pedro, Berto and Simona, for your help during this process. Your infinite patience and countless hours have fundamentally shaped this project, and each of you contributed in a unique way: Pedro's sense of practicality, Berto's wide-picture thinking and Simona's mathematical rigor and attention to detail proved invaluable in covering a wide range of perspectives in this thesis.

This project is dedicated to my family: especially, my parents, Juan Carlos and Maria Rosa, my stepfather Joan Manel and grandparents Joan and Rosario. I owe you absolutely everything I have achieved up to now, and most of what I will achieve from now on. I am where I am today because you have been pushing me from below: "Arrivare in alto è merito di chi ti ha sempre spinto dal basso", as they say.

I cannot forget my lifelong friends, those who have been there ever since I can remember: Marina, Elisenda, Marc, Paula, Andrea, Marta, Pol, Eric, Inés, Fabi and Esther. I also remember all of the friends I have met during my time as a university student: Sara, Iván, Javier, Ignasi, Elvira, Jana, Sandra, Alejandro, Daniela, Thérèse, Elisabeth, Tobias, Tarik, Meritxell and Illán. You made this entire process much more enjoyable and thanks to you I will not only leave this university with knowledge, but with fond memories and friends for life.

Contents

Abstract

The solution of the advection-diffusion equations, which describe pollution transport, for large urban areas using traditional numerical methods is computationally expensive. This makes applications such as real-time or interactive modeling of pollution transport unfeasible, applications which would be very useful in urban areas to improve pollution control policies and better manage urban spaces in a moment where increasing urbanization and transportation demand are leading to worsening in air quality in many cities around the world. In this thesis, an alternative method, the proper generalized decomposition (PGD), is applied to solve the pollution transport problem. PGD allows parametric modeling, resulting in a generic solution which can be obtained immediately for different parameter combinations once the approximation has been computed once, thus overcoming limitations of traditional numerical methods. The results in synthetic urban domains are promising and show that PGD can typically approximate the FEM solution with reasonable accuracy with a reasonable number of modes in the decomposition for simple cases. However, challenges remain involving the non-linear terms, which cannot be separated but are necessary for fully stabilizing advection-dominated problems. Furthermore, the implementation of the approach in meshes of real urban areas introduces further complications, thereby yielding inconclusive results about the ability of PGD to correctly approximate solutions in real cities. The main contributions of this thesis have been: i) the combination of PGD with stabilization terms for steady-state advection-diffusion equations, ii) the application of algebraic PGD to parametrize the solution as a function of the wind direction and module and iii) the implementation of the PGD in a mesh representing an actual urban area in Barcelona.

Abbreviations

1D/2D/3D: 1 Dimensional / 2 Dimensional / 3 Dimensional
BC : Boundary Condition
BSC : Barcelona Supercomputing Center
DBC : Dirichlet Boundary Condition
EPA : Environmental Protection Agency
FEM : Finite Element Method
HPC: High Performance Computing
LACAN : LAboratori de CÀlcul Numèric (UPC research group)
MATLAB: MATrix LABoratory (Computer Software)
NBC : Neumann Boundary Condition
ODE : Ordinary Differential Equation
PDE : Partial Differential Equation
PGD : Proper Generalized Decomposition
PM : Particulate Matter
RBC : Robin Boundary Condition
ROM : Reduced Order Model
UPC : Universitat Politècnica de Catalunya
SUPG : Streamline Upwind Petrov Galerkin
SVD : Singular Value Decomposition
US : United States
WHO : World Health Organization

Chapter 1

Introduction

Air pollution is a prevalent problem in the world today. While many of the compounds traditionally classified as "air pollution" were naturally found in the atmosphere thousands of years ago in very small quantities, human activity has caused the concentrations of many of these chemical species to increase. This trend has been especially important in the past decades as the world population has increased and the adoption of technologies contributing to the emission of these pollutants has been widespread. Air pollution has been gaining attention by both citizens and public entities as studies have revealed the negative impact that it can have on human health. Urban pollution is estimated to contribute to more than 3.8 million premature deaths annually through various mechanisms: it is a risk factor in the development of cardiovascular disease, increases the risk of suffering from a stroke, and can lead to increased chances of developing cancer in the respiratory system. Other surprising long-term effects, such as cognitive impairment in older people [30], as well as increased chances of dementia, Parkinson's disease and Alzeimer's disease, have also been reported [21].

The term "air pollution" is a broad term referring to a number of different chemical species which are found in the air in concentrations different than those which would be measured in normal conditions. According to the United States (US) Clean Air Act, in its title III, Section 7602(g) defines air pollutants in the following way:

"The term "air pollutant" means any air pollution agent or combination of such agents, including any physical, chemical, biological, radioactive (including source material, special nuclear material, and byproduct material) substance or matter which is emitted into or otherwise enters the ambient air." [25]

The definition does not describe what an "air pollution agent" is. However, an indication can be found in Title 42, Section 7408, which states that the US Environmental Protection Agency (EPA) must publish a list of air pollutants:

"emissions of which, in his judgment, cause or contribute to air pollution which may reasonably be anticipated to endanger public health or welfare"[26]

This is a very broad definition which encompasses many different compounds affecting humans in very diverse ways through various mechanisms. Some common pollutants which are the subject of many scientific studies include particulate matter (PM), ground-level ozone (O3), methane, nitrogen oxides, carbon monoxide and sulphur dioxide. These are known to have a hazardous effect on human health [12]. The World Health Organization (WHO) publishes guidelines regarding the maximum concentration of these pollutants

which are safe for humans to breathe.

Whether carbon dioxide (CO2) should or shouldn't be considered as a pollutant still remains a subject of scientific debate, given that emissions of CO2 are necessary for the balance of the planetary ecosystem and they don't directly affect human health by increasing the risk of suffering from an illness. However, given that the amount of CO2 endangering public welfare when emitted in the amounts which are typical of modern cities via exacerbation of climate change, as well as its potential for ocean acidification, CO2 will also be considered as a pollutant.

Although the problem of pollution is a global one, susceptible of affecting any person in any country, it is especially concentrated in cities around the planet. 75% of greenhouse gas emissions originate from cities, despite the fact that urban areas account for roughly only 2% of the Earth's surface. In cities, gases with high concentrations of pollutants are originated mostly by internal combustion engines in vehicles, residential buildings and industries (such as coal-burning electricity plants). In many developing cities around the world, the PM concentration can be as high as 4-15 times the one judged as safe by the WHO. According to a recent survey by the WHO, only 20% of the global urban population lives in areas which comply with WHO air quality guidelines for PM2.5, which is considered to be the most harmful air pollutant for humans [28]. In a context of increasing worldwide urbanization, with roughly 70% of the population expected to live in urban areas by 2050, and increasing pressure on existing infrastructure as motorized transport becomes ubiquitous in communities with rising middle classes, understanding pollution transport is more necessary than ever to make informed decision-making with the objective of mitigating its environmental and public health impacts.

Figure 1.1: Pollution from factories in Bratislava, Slovakia (Picture by Dominik Dancs)

Pollution doesn't stay indefinitely in the place where it is originated. Instead, it is transported through the air and is affected by various factors, such as the wind. A very clear example of pollution transport can be seen in Figure 1.1. Pollution is transported following a set of relatively well-known equations. Often referred to as the advection-diffusion-reaction equations or the convection-diffusion-reaction equations, this set of partial differential equations (PDE) describes the transport phenomena as a function of the derivatives of the pollutant concentration. Except for very simple 1-dimensional or 2-dimensional cases, there is rarely an analytic solution which can be derived for these equations. Instead, the use of numerical models and computational simulations is often required

for solving this equation. However, traditional methods have a number of limitations.

First, the complexity (which is determined by the number of degrees of freedom of the problem) of the computation can be very high, especially if we consider 3-dimensional domains which represent the real geometry of vast areas in cities. The solution of an advection-diffusion problem is usually found by solving a system of equations, the number of unknowns of which depends on the number of points at which we actually solve the equation and the dimensionality of the space. For instance, for the solution of a PDE using the Finite Element Method on a computational 3D Cartesian domain with n n nodes along each coordinate axis we can expect a complexity of n^3. As n increases, it soon becomes computationally very costly to solve the equations. Although most household computers can run some simple simulations, and modern computer clusters offer enough resources to perform computations which wouldn't have been possible twenty years ago, it is still a challenge to run three-dimensional simulations within a reasonable time frame, something which would be helpful in a number of applications.

Second, there is rarely a situation in which the input parameters can be perfectly described. The pollution concentration at any given point within a fixed domain is dependent on a lot of factors, including, for instance, the amount of emitted pollutant, the wind direction or the wind strength. If we hope to predict how the pollutant is transported, we need to have a detailed knowledge of what the variables that describe this phenomenon are. This has traditionally been tackled by the resolution of inverse problems, where the problem is "solved backwards" starting from the answer as we try to infer the parameters which may have led to that solution. However, it may take many attempts to find a combination of problem data which yields the solution we are looking for.

Finally, there is a certain time delay between the beginning of the computations and the visualization of the results, which can become very large as the complexity of the computations increases. This delay makes certain applications, such as real-time simulation or interactive modeling, impractical. The use of ROM can contribute to solving these issues by generating a small space where solutions can be sought, thereby bypassing the need to solve the equations in a computationally expensive way.

Recent advances have helped to partially overcome some of these difficulties. On the one hand, the continuous increase in average computational power during the past decades have made it much faster to run a pollution transport simulation on a standard laptop today than 20 years ago. On the other hand, extensive scientific research has been conducted to improve traditional methods. For instance, mesh adaptation aims to reduce the computational complexity of a given problem by performing more calculations only where it is really required to capture the solution.

A different approach to tackling the computational complexity problem has been to apply reduced order models (ROM). Model order reduction refers to a family of computational techniques which explicitly try to reduce the computational complexity associated with a given problem. In particular, the problem is somehow intentionally simplified, normally by either reducing the amount of degrees of freedom or by introducing further assumptions which result in a search for a solution in a much smaller solution space. Furthermore, the problem simplification and subsequent lower complexity allows us to introduce problem parameters into the ROM and solve for them, which eases parameter identification and simplifies the resolution of inverse problems. ROMs are not free from drawbacks: some of the simplifications they introduce can sometimes lead to a certain decrease in accuracy with respect to the original solution, and their proved robustness remains limited to a small number of cases. However, they show promising results and studying them to understand

when and under which circumstances they can serve as an additional tool for simulating air pollution will be the objective of this report.

1.1 Objectives

A specific type of ROM will be studied in detail in this report: the Proper Generalized Decomposition (PGD). This method is based on the idea that instead of solving one high-dimensional complex problem, we can solve a series of modified problems of lower dimensions and add the solutions together to come up with an approximated solution, somehow mimicking traditional mathematical concepts of Taylor or Fourier series but applied to numerical problems. This method, however, needs to be studied in detail to understand what its pros and cons when applied to our specific context. As a method which approximates the solution, it is essential to study its accuracy with respect to that of traditional methods, and understand what factors (both intrinsic to the problem as well as imposed by the model) may have the greatest influence on its performance.

This thesis has been carried out to gain a better understanding of the performance of the PGD approximation for the computation of the solution of pollution transport equations. A good approximation to the solution of the advection-diffusion equation using PGD would pose numerous advantages which would result in lower computational times, readily available solutions and possibility of overcoming traditional engineering bottlenecks such as real-time simulations.

An improved understanding of the way pollution transport happens in different contexts can be the decision basis used to recommend robust pollution-control policies and measure the impact that different scenarios are likely to have. This would lead to healthier air for people to breathe, reduced environmental impact, mitigation of climate change and more livable urban areas, to name a few benefits.

1.2 Document structure

This report is structured in the following way: Chapter 2 presents the problem statement, while Chapter 3 explains the Finite Element Method and applies it to the pollution transport problem in urban areas. Chapter 4 briefly introduces the PGD model reduction procedure, focusing specifically on parametric PGD (where parameters are considered as variables when solving the system of equations) and explains how to apply it to the problem at hand. Chapter 5 describes a specific urban area in the city of Barcelona where it would be interesting to solve the problem, and the procedure used for handling a computational representation of such a space. Then, after performing the analyses, in Chapter 6, the results the solutions to the advection-diffusion problem solved by applying the previously explained concepts are laid out. Finally, the discussion in Chapter 7 shares some insights into the advances made with this thesis and presents further aspects which could be studied to further substantiate the results.

Chapter 2

Problem Statement

2.1 The advection-diffusion equation

The advection-diffusion-reaction equation (sometimes also referred to as the convection-diffusion-reaction equation) governs, for example, the transport of species inside a fluid. While it will be used to model the transport of air pollution, it can also be used, for instance, when simulating transport phenomena of chemical species in liquids. In its most general expression, the equation is the following:

$$\underbrace{\frac{\partial c}{\partial t}}_{\text{time evolution}} = \overbrace{-\nabla \cdot (D\nabla c)}^{\text{diffusion}} + \underbrace{\nabla \cdot (\mathbf{v}c)}_{\text{advection}} + \overbrace{R}^{\text{reaction}}, \tag{2.1}$$

where c is the concentration of pollutant, generally expressed in kg/m^3 or mol/m^3, D is the diffusivity in m^2/s, $\mathbf{v}$ is a vector field representing the velocity in m/s, and R is a reaction term with units of $kg/(m^3 \cdot s)$ or $mol/(m^3 \cdot s)$. Each of the terms in (2.1) describes a different phenomenon.

The first term in (2.1), $\frac{\partial c}{\partial t}$, expresses the time variation of the concentration of the species. As long as the problem is evolving, this term will be different from zero. If the problem reaches steady-state, meaning that it is no longer evolving in time, then this term goes to zero.

The second term is the diffusive contribution. Diffusion is a phenomenon such that there is a transport of mass due to the existence of a nonzero concentration gradient for the species. Because of the differences in concentration, the pollutant will travel from the areas with the most concentration to the places with least concentration. Therefore, the gradient has a negative sign in front. This term is controlled by the diffusion coefficient D, also called diffusivity, which gives an idea of how the pollutant spreads around. This phenomenon develops in all the directions as long as there is a difference in concentration.

The third term in equation (2.1) is the advection (or convection) part. This expresses the species transport due to the existence of a convective field. For instance, the velocity of a fluid will cause the suspended particles inside the fluid to move along the direction of the velocity. This phenomenon therefore only happens in the direction of the flow at each point within the fluid.

The fourth term is the reactive contribution. It defines the unit volume rate of species creation or destruction. For instance, if our pollutant of interest reacted with another species in the air and degrades at a rate which is sufficiently high to be considered in the modeling, a negative reactive term would be included here to reflect the existence of a sink. A positive term would correspond to the existence of a volume source of pollutant within our domain.

In this thesis, we will forego the reaction term (therefore $R = 0$) as the pollutant will not be created nor destroyed inside the domain, so the advection-diffusion-reaction equation reduces to the advection-diffusion equation. The pollutant will instead be assumed to generate outside of the urban area under study and introduced into the domain through the boundaries.

We will study the steady state configuration of this equation, which means that we will assume that boundary conditions and parameters remain fixed and therefore the system has the time to reach a stable state where the solution does not evolve in time. It is this final configuration that we want to model. As such, the time-derivative will be set equal to zero ($\frac{\partial c}{\partial t} = 0$) . Therefore, a first resulting simplified equation is as follows

$$0 = -\nabla \cdot (D\nabla c) + \nabla \cdot (\mathbf{v}c) \tag{2.2}$$

This equation can be further simplified in our case, where we will assume that the diffusion coefficient D is a constant which is isotropic (identical in all directions) and uniform (homogeneous across the entire domain). This means that it can be expressed as a scalar without any spatial parameter dependence, and is thus not affected by the divergence operator. The equation can thus be rewritten in the following form:

$$0 = -\nabla \cdot (D\nabla c) + \nabla \cdot (\mathbf{v}c) = -(\underbrace{\nabla D}_{0} \cdot \nabla c + D\nabla \cdot (\nabla c)) + \nabla \cdot (\mathbf{v}c) = -D\Delta c + \nabla \cdot (\mathbf{v}c),$$

where Δ is the Laplacian operator, defined as the divergence of the gradient ($\Delta c = \nabla \cdot (\nabla c)$).

We can introduce another simplification, i.e. we will consider the flow incompressible. Incompressibility of a flow should not be confused with incompressibility of a fluid: if a flow is incompressible, it means that the density of a small volume moving along with the fluid velocity is constant. Compressible fluids, such as air, can display incompressible flows. Incompressibility of a flow mathematically results in a zero divergence condition for the velocity ($\nabla \cdot \mathbf{v} = 0$). This is a valid assumption for flows with a Mach number (velocity of the fluid with respect to the velocity of sound when traveling along the fluid) under 0.3 [8], which is a valid assumption for the air velocity in normal circumstances. A Mach number of 0.3 converts to a velocity of roughly 357.8 km/h. Because our final goal is to model pollution transport in an urban area, and in this thesis a small part of Barcelona has been adopted as the target area, we are interested in knowing what the maximum wind speeds are. In the 2013-2017 period, the maximum wind speed that was reached in Barcelona was of 94 km/h [2], which converts to less than 0.1 Mach. Therefore, incompressibility of the air flow will be assumed for the rest of this report. This means that we can introduce yet another simplification:

$$0 = -D\Delta c + \nabla \cdot (\mathbf{v}c) = -D\Delta c + (\underbrace{\nabla \cdot \mathbf{v}}_{0} c + \mathbf{v} \cdot \nabla c) = -D\Delta c + \mathbf{v} \cdot \nabla c, \tag{2.3}$$

which will be the preferred form of the equation under the considered conditions.

Before moving to the next section, special attention must be given to the diffusion coefficient D. We have assumed that D is a constant. However, it is not straightforward to choose a value for the D, because there are many ways to choose this coefficient. Diffusivity is not a fixed quantity, given that it depends on many factors: for instance, diffusivity is highly scale-dependant. Furthermore, diffusivity is pollutant dependent, as it may not be the same for ozone than for PM2.5 or for carbon dioxide. Nevertheless, it is generally accepted that for air pollution transport problems, the exact choice of D is generally unimportant as long as it is within the right order of magnitude [31]. This is because in outdoor and well-ventilated indoor pollution transport problems, the transport of pollution is dominated by the wind velocity and not by the diffusion. Typically the effects due to diffusion are negligible, which means that it is not necessary to accurately estimate the diffusivity coefficient. The most typical approach is therefore to ignore the differences between pollutants when solving a fluid dynamics problem, instead taking a typical value for the diffusivity which could vary by 1 or 2 orders of magnitude, but still have a very small impact on the final results due to the much more important effect of advection. In this report, a value of 10^{-4} has been typically assumed for D.

2.2 The potential flow model

One of the inputs which are necessary for computing solution to the advection-diffusion equation within a given domain is the velocity field. However, this field is a priori unknown and must be computed before solving the advection-diffusion problem we are interested in.

The flow of air is governed by the Navier-Stokes equations. These correspond to a system of coupled equations which describe the relationship between the velocity, pressure, temperature, and density of any given moving fluid. The equations are complex and have no closed analytic solution. Furthermore, they are computationally costly to solve and are inherently prone to instabilities.

Alternatively, by means of a few hypotheses, the Navier-Stokes equation can be simplified to uncoupled expressions with fewer terms which are not as difficult to solve. One idealized simplified model of the Navier-Stokes is the potential flow model. In this case, we assume the existence of a velocity potential Φ of a flow. This potential Φ satisfies the Laplace equation:

$$\Delta\Phi = 0, \tag{2.4}$$

for a given set of conditions. However, if we want to solve this equation instead of the Navier-Stokes one, we have to make sure that our flow behaves according to some additional assumptions:

1) Incompressible flow: as already discussed in the section above, this assumption has been accepted in urban areas in the context of the advection-diffusion equation. Mathematically, this means that the divergence of the velocity is zero: $\nabla \cdot \mathbf{v} = 0$.

2) Irrotational flow: An irrotational flow doesn't present any vorticity. This means that the curl of the velocity field is zero: $\nabla \times \mathbf{v} = 0$. This is a reasonable assumption if we don't consider the effects due to the boundary layer [18]. Because we are interested

in what happens on a large scale, where small vortices become negligible, we can consider this assumption to hold for our case study.

3) Inviscid flow: An inviscid flow is one such that viscosity effects can be neglected. In reality, all fluids are inherently viscous. However, if the effects of viscosity are very small when compared to the inertial ones for a given flow, they can be ignored, thereby greatly simplifying the solution of the problem. To see if a flow can be considered to be inviscid, we consider its Reynolds number, which is computed as follows:

$$Re = \frac{UL}{\nu},$$

where U is the fluid characteristic velocity, L is the characteristic length of the object the fluid is flowing over and ν is the kinematic viscosity of the fluid. If the Reynolds number of a fluid is very large ($Re \to \infty$), the fluid can be considered as inviscid. It is usually considered that when $Re > 10^3$, the viscous effects become negligible with respect to the inertial ones and the viscosity can thus be ignored [29]. In our particular case, where we are trying to model pollution dispersion in the Eixample district in Barcelona, air is flowing over and around buildings. The characteristic dimension of a building block will be considered as the length of a typical block in the Eixample, which is 113.3m [19], but will be rounded to 115m. The lowest velocity we will be dealing with will be considered to be 0.1 m/s and the kinematic viscosity of air at 25°C and 1 bar of pressure is $1.78 \cdot 10^{-5}$ [24]. Therefore, a first approximation of the lowest Reynolds number we would consider in the Eixample district will be:

$$Re = \frac{0.1 \cdot 115}{1.78 \cdot 10^{-5}} \approx 6.5 \cdot 10^5,$$

which is more than two orders of magnitude higher than the one we need to justify the assumption of an inviscid flow, and is therefore a sufficient condition to make this choice. Even if we were to take a single isolated house (with a characteristic length of 10m) and lower the velocity by 1 order of magnitude (to 0.01 m/s), our Reynolds number would still be higher than the one we need to justify our inviscid approximation to fluid flow. This will therefore be taken as a valid simplification under all cases that we will study.

Equation (2.4) is solved numerically to obtain the value of the potential flow within the area of interest. This equation can be solved using a variety of computational methods. In our case, this has been done using the Finite Element Method, which is described in the following chapter. This method solves the equation above within a bounded domain. The solution to the problem will fundamentally depend on the boundary conditions which are applied. As different values for the potential are chosen at the boundaries, the velocity field inside the domain will change.

Our problem of interest is a problem in the 3D space. However, when imposing boundary conditions to compute the velocity field, only potentials varying along the x and y dimensions are considered. This means that the potential is not considered to vary along the vertical axis. This, however, does not mean that the velocity inside the domain does not have any vertical component. As the air flows inside the domain and it encounters obstacles (buildings in our case), its trajectory is substantially modified and vertical components may develop, becoming especially large in the vicinity of impediments.

Once we have solved equation (2.4), the velocity field is simply obtained as the gradient of the scalar field:

$$\mathbf{v} = \nabla\Phi = \Big(\frac{\partial\Phi}{\partial x}, \frac{\partial\Phi}{\partial y}, \frac{\partial\Phi}{\partial z}\Big),$$

where $\mathbf{v}$ is a vector field. This last step explains why this model is referred to as the potential flow model, given that it corresponds to the traditional notion of potential in Physics and Mathematics.

In our problem, a generic velocity field is expressed as follows:

$$\mathbf{v} = \nu(\sin(\theta)\mathbf{v}_1 + \cos(\theta)\mathbf{v}_2), \tag{2.5}$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the velocity fields obtained by solving potential flow model with a potential varying only along the x and y axis respectively and ν is a parameter which controls the module of the velocity. If the domain is symmetric with respect to the x and y axes and has the same dimension along the x and y directions and the potential imposed on the boundaries is the same, then the two velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ will be identical but rotated by $\pi/2$.

The velocity field is therefore be explicitly dependent on the angle of the incoming wind, θ. By setting $\theta = 0$, we obtain a velocity which is equal to $\mathbf{v}_1$, and by setting $\theta = \pi/2$, we obtain that the velocity is equal to $\mathbf{v}_2$. By choosing an angle in between, we obtain a velocity in-between as a linear combination of the two velocities for the extreme angles. This means that the velocity field generated by the wind from any direction can be computed as the weighted average of two input velocities. Once we have obtained the desired direction for the velocity, its module can be adjusted by tuning the parameter ν.

Therefore, by solving for the two velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ we can find the velocity obtained by the imposition of a potential flow which varies in any of the directions in the xy plane.

Chapter 3

The Finite Element Method

Humans are taught to solve equations analytically using diverse methods. The analytic solution of a PDE involves finding an explicit expression of the function which solves the problem. This means that the solution is expressed as a function of the problem variables, and when specific points in the domain and problem parameters are plugged into the expression, a particular numerical solution is obtained. Computers are unable to find these analytic solutions: they are good, however, at performing arithmetic computations very fast, typically solving millions of simple operations per second. The idea behind using computational methods to solve PDEs, such as the one in (2.3), is the following: we are interested in transforming PDEs into alternative forms which can be solved numerically (i.e. by obtaining discrete values which represent the solution at certain points) rather than analytically (i.e. by obtaining a general expression of the target equation). This typically involves performing some mathematical manipulations which yield an alternative form of the original PDE, which can be solved more easily and results in an approximation of the solution to the original problem. This change in problem formulation is done not only because of computer limitations, but also because many PDEs simply don't have a known analytic solution, but can still be approximated numerically.

3.1 Problem geometry

Before solving a PDE problem, we have to define the area of interest where we want to find our numerical solution. Since we are going to compute a discrete solution with a computer which has finite memory to store numerical values, we identify a domain of the PDE within a finite portion of the 3D Cartesian space where we will obtain the approximated solution.

We are interested in approximating the advection-diffusion equation in urban areas: as such, we will create a simple model recreating a part of a city where we will solve our problem. As we will see later, to appropriate solve the equations this geometry needs to be bounded.

A mathematical model of a city is not an actual representation of a real city. Rather, it is a simplified rendering of an urban area with its major defining elements. Therefore, we will not include many of the elements which characterize cities: people, vehicles, trees or street furniture to name a few. Instead, we will only keep the features which are important for the model, that is, the ground and the buildings. There will also be some fictitious lateral walls of our model - which do not represent actual walls, but just the limits of the

model, just like the borders of a map. In a similar way, there will also be a top limit. Therefore, we can imagine our computational domain of a city as a box (which may be regular, or not) with indentations on the bottom surface which correspond to the buildings. The "limits" of our computational model, the fictitious "walls" of the box, will be hereon named "boundaries", introducing the terminology which will be used in the rest of this report.

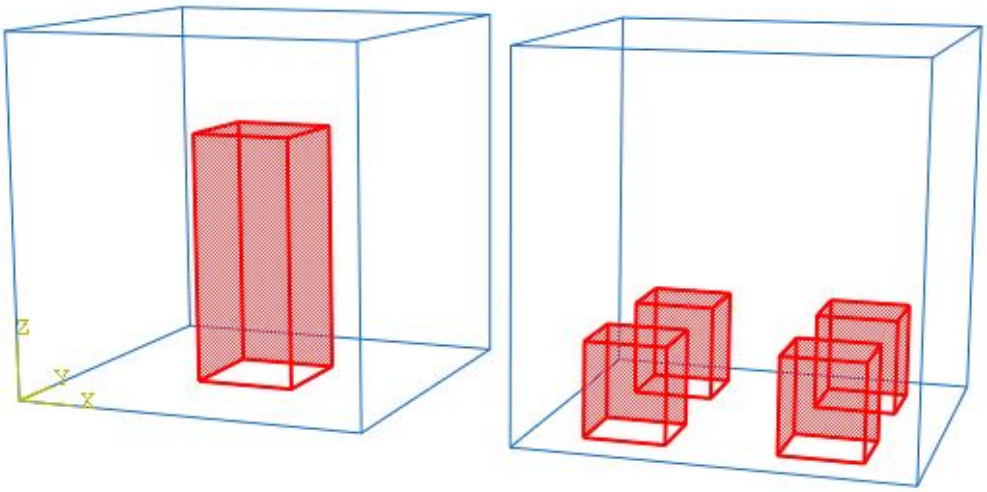

Figure 3.1: Wireframes of two synthetic computational domains of small urban areas, featuring a single high-rise building (red, left) and four low-storey buildings (red, right)

Two sample synthetic urban geometries have been generated and can be seen in Figure 3.1. We can see a three dimensional domain consisting of some boundaries, with indentations at the bottom representing the buildings, which have been highlighted in red.

3.2 Finite element formulation for an advection-diffusion problem

There are various ways in which the solutions of PDEs can be found numerically. One of the most famous and versatile methods is the Finite Element Method (FEM).

The FEM is based on the discretization of PDEs within a standard framework which allows us to find a solution in a convenient space. This requires applying a series of well-defined steps to find an alternative formulation of the original problem which can be solved within the finite element framework. These steps apply different mathematical manipulations to modify the equation to be solved, while ensuring that the solution of the modified equation is a valid solution for the original problem in the distributional sense. The original equations are usually called strong form of the problem, while the manipulated ones constitute the so-called weak (or variational) form and finally, the equations in the finite dimensional space represent the discretized FEM form. They will now be applied to the problem of pollution transport in equation (2.3).

1) Strong form

The starting point to find the FEM formulation of the advection-diffusion equation is stating the strong form of the problem. This defines the equation to be solved along with the boundary conditions that express certain characteristics the solution needs to fulfill at

the boundaries. If boundary conditions are not explicitly provided, the so-called "natural boundary conditions" are imposed.

In the case study of interest, we are dealing with a steady-state problem, so there is no need to explicit the initial conditions since no time evolution will be considered. If we consider a transient problem, we also need to define the initial conditions within the domain at the beginning of the simulation, and the equations then yield the time evolution of the initial concentration.

There are 3 types of boundary conditions: Dirichlet, Neumann and Robin conditions. Dirichlet boundary conditions impose values for the solution to the problem, in our case, the concentration, on the boundaries. Neumann boundary conditions express a condition on the derivative of the solution, which can be interpreted as a flow by multiplying the derivative by a coefficient which corrects the units and gives the condition a physical meaning. Robin boundary conditions state the desired value of a combination of both the solution and its derivative, and are considered as an extension of Neumann boundary conditions.

Let Ω be the domain where we are solving the problem. Let $\partial\Omega$ be the boundary of such domain. We can then decompose the boundary into components, not only depending on what they represent in the real world (buildings, streets, etc.) but also depending on what type of boundary condition is applied to that portion. Let $\partial\Omega_D$ be the part of the boundary where Dirichlet conditions are applied, $\partial\Omega_N$ the part of the boundary with Neumann conditions and $\partial\Omega_R$ the component with Robin boundary conditions. It must then hold that $\partial\Omega_D \cup \partial\Omega_N \cup \partial\Omega_R = \partial\Omega$.

For each of the boundary portions $\partial\Omega_D, \partial\Omega_N$ and $\partial\Omega_R$, there can be further subdivisions as there can be different boundary conditions of the same type but with different values. Therefore, for instance, $\partial\Omega_D = \partial\Omega_{D_1} \cup \partial\Omega_{D_2} \cup ... \cup \partial\Omega_{D_m}$, where m different Dirichlet conditions are applied on different parts of the Dirichlet boundaries. This decomposition can be extended to Robin and Neumann BC as well.

The strong form of the problem therefore consists of the PDE to be solved together with the boundary conditions assigned to the corresponding parts of the boundary. For a steady-state advection-diffusion problem with generic boundary conditions, we have:

$$\begin{cases} 0 = -D\Delta c + \mathbf{v} \cdot \nabla c & on\ \Omega & \\ c = a_i & in\ \partial\Omega_{D_i} & \forall \partial\Omega_{D_i} \in \partial\Omega_D \\ D\frac{\partial c}{\partial \mathbf{n}} + b_j = 0 & on\ \partial\Omega_{N_j} & \forall \partial\Omega_{N_j} \in \partial\Omega_N \\ D\frac{\partial c}{\partial \mathbf{n}} + d_k c = e_k & on\ \partial\Omega_{R_k} & \forall \partial\Omega_{R_k} \in \partial\Omega_R \end{cases} \tag{3.1}$$

where $\frac{\partial c}{\partial \mathbf{n}} = \nabla c \cdot \mathbf{n}$ is the normal derivative of the concentration, with $\mathbf{n}$ the exterior unit normal at each point of the boundary.

Going back to the classification of boundaries with respect to their physical meaning, the boundary $\partial\Omega$ of our urban area domain become the following: $\partial\Omega = \partial\Omega_{bot} \cup \partial\Omega_{top} \cup \partial\Omega_{lat} \cup \partial\Omega_{build}$. It should be noted that $\partial\Omega_{bot}$ refers to the bottom of the domain, which will model the streets, $\partial\Omega_{lat}$ models the fictitious lateral boundaries, $\partial\Omega_{build}$ models the buildings and $\partial\Omega_{top}$ is the upper boundary which is a plane in the air above the urban area.

We have chosen to take a cubic-like shape for our urban model, which corresponds to the existence of 4 lateral sides, as can be seen, for instance, in Figure 3.1 or 3.2. We

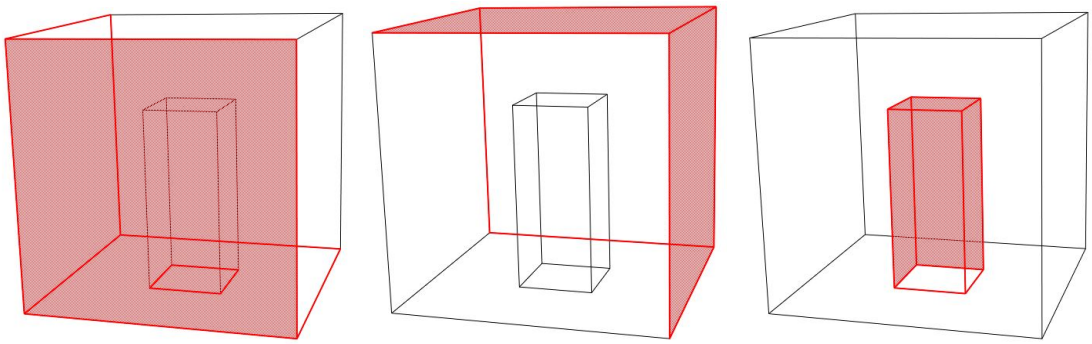

Figure 3.2: Assuming a wind that flows from left to right and from front to back, the Dirichlet boundaries are highlighted in the first picture (inlet and bottom), the Neumann boundaries in the second (outlet) and the Robin in the last (building)

can further subdivide the laterals into inflow and outflow boundaries depending on the direction of the wind. The inflow boundaries are the ones through which our wind enters the domain (corresponding to the points where the exterior normal to the surface times the wind vector at that point corresponds to a negative vector product) and the outflow boundaries are those through which the wind exits the domain (which means that the cross product of the wind vector and the exterior normal results in a positive or zero quantity). This means that $\partial\Omega_{lat} = \partial\Omega_{lat,in} \cup \partial\Omega_{lat,out}$.

The inflow part of the boundary, $\partial\Omega_{lat,in}$, will be assumed to have a zero concentration value. This is because the high Péclet numbers we are working with result in most of the pollutant being whisked away from the inflow at a fast enough rate as to consider negligible the amount that actually reaches this boundary.

The streets of the city, which correspond to $\partial\Omega_{bot}$, will be assumed to be emitting pollutant as the cars in the city use fuel to move around. As such, they will be assumed to have a constant concentration of pollutant, so a constant Dirichlet condition will be imposed on the bottom. Of course, urban streets do not emit pollution - the cars circulating on them do - but we can assume that, for this simplified model, the cars can be integrated with the streets, which then become sources of pollutant gases.

The outflow and top of the city $\partial\Omega_{lat,out}$ and $\partial\Omega_{top}$ are actually open as they don't represent any real boundaries, but just fictitious limits that we have to set in order to solve the equations on a closed domain. Therefore, any pollutant that reaches these boundaries should be allowed to leave the domain. This corresponds to a "natural" boundary condition, free-flow, or zero flux ($\frac{\partial c}{\partial \mathbf{n}} = 0$).

The buildings $\partial\Omega_{build}$ will be assumed to absorb some of the concentration in the air. This is due to the nature of the materials which buildings are made of. Most buildings in modern cities are made out of concrete, which is a porous material. This means that some of the particles floating in the air will get trapped in the material pores when the polluted air hits against the walls and ceiling of the buildings. Therefore, the buildings will be assumed to absorb part of the pollutant floating around. The absorbed amount of pollutant is assumed to be proportional to the concentration with a factor β, which is determined by the concrete porosity. This absorption is therefore represented as a flux which depends on the concentration, which is formalized as a Robin boundary condition.It should therefore be noted that buildings will be considered to be residential and/or office buildings, with neither industries nor factories within the urban area we are modeling.

Figure 3.2 shows three views of a synthetic urban domain with highlighted Dirichlet,

Neumann and Robin conditions as explained above. In Figure 3.3, we can see a longitudinal section of a further synthetic urban domain with the same boundary conditions sketched all together.

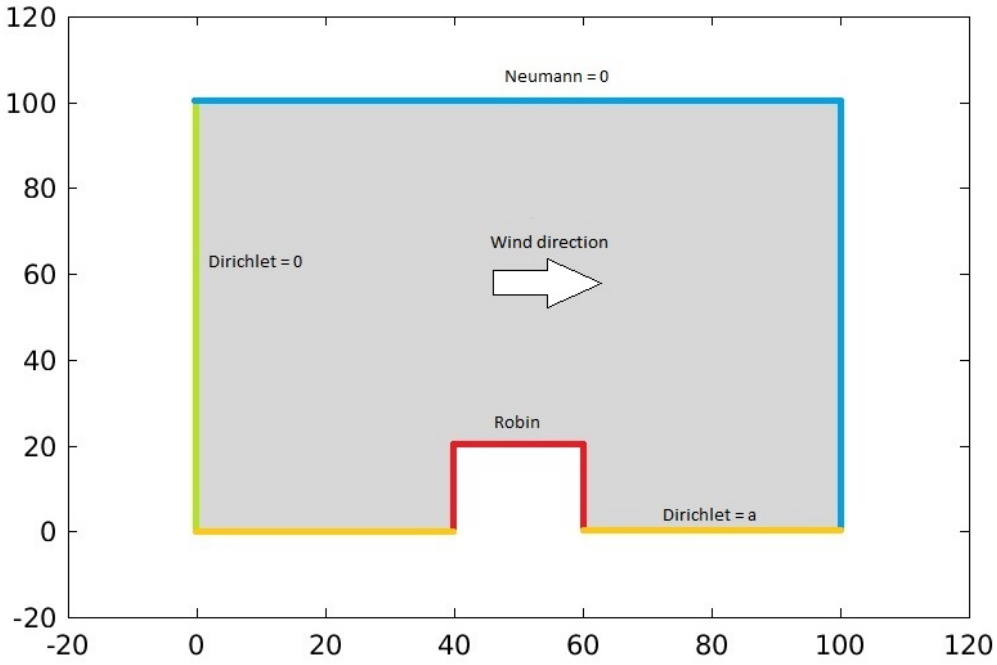

Figure 3.3: 2D longitudinal section of a synthetic domain with the corresponding BCs in different colors. Wind direction is assumed to be from left to right.

The mathematical representation of these boundary conditions together with the corresponding equation allow us to properly define the strong form of our problem, which is the following:

$$\begin{cases} 0 = -D\Delta c + \mathbf{v} \cdot \nabla c & in\ \Omega \\ c = a & on\ \partial\Omega_{bot} \\ c = 0 & on\ \partial\Omega_{lat,in} \\ D\frac{\partial c}{\partial \mathbf{n}} = 0 & on\ \partial\Omega_{lat,out} \cup \partial\Omega_{top} \\ D\frac{\partial c}{\partial \mathbf{n}} + \beta c = 0 & on\ \partial\Omega_{build} \end{cases} \tag{3.2}$$

When modeling the problem, we choose an appropriate parameter a which represents the expected concentration of pollutant in the streets of the city.

We can therefore see that, in this particular expression of our problem, $\partial\Omega_{bot} \cup \partial\Omega_{lat,in} = \partial\Omega_D$, $\partial\Omega_{lat,out} \cup \partial\Omega_{top} = \partial\Omega_N$ and $\partial\Omega_{build} = \partial\Omega_R$. Therefore, the three different types of boundary conditions appear in different parts of the boundary.

Before going on to the next step it is important to note that, while the problem as stated above was the one solved in the majority of cases, for some particular situations that will be explained in the Results section, the model above was too ideal and yielded results that were overly optimistic. To obtain results which gave a more accurate idea of the actual performance of the techniques under study, a slightly more intricate boundary condition was imposed in some cases where instead of applying a homogeneous non-zero Dirichlet boundary condition on the entire base, it was only done on a part of the bottom boundary. A longitudinal sketch of the resulting section can be seen in Figure 3.4.

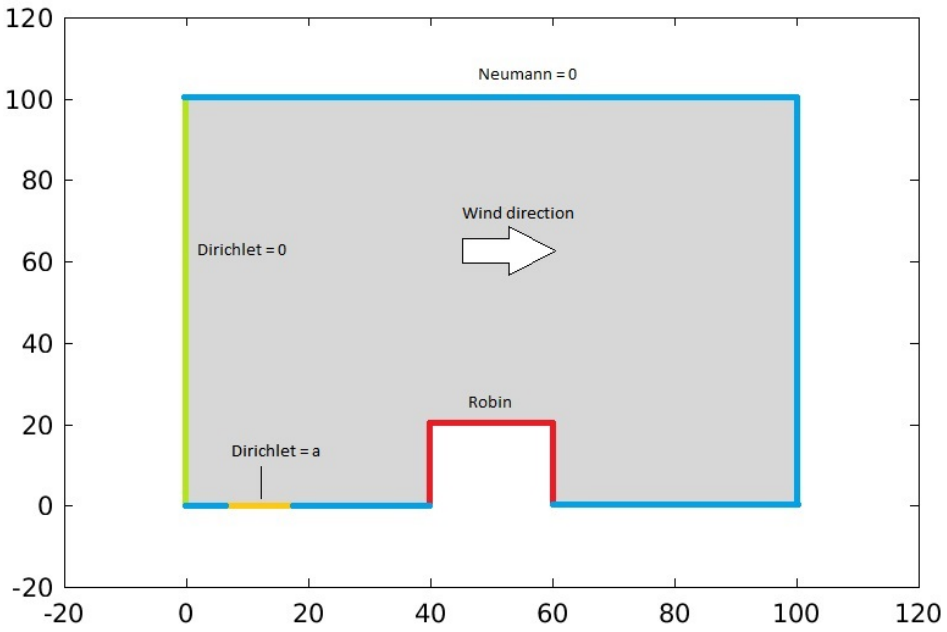

Figure 3.4: 2D longitudinal section of a synthetic domain with modified boundary conditions such that the non-zero Dirichlet BC is only applied on part of the bottom boundary.

However, we are going to proceed by applying the FEM framework only to the initial case. This is because, in this new problem statement, equation (3.2) still holds, with only changes in the boundaries where we are applying the boundary conditions. Because only boundaries with zero Neumann and Dirichlet boundary conditions are affected by this modification, as it will be seen, the resulting equations remain exactly the same for both cases as the terms involving these types of conditions soon vanish from the formulation. The only needed changes are at the very end of the procedure when the final form of the advection-diffusion equation has been obtained and we are choosing on what part of the boundary we want to solve it.

In the Results section, it will be duly noted whenever this test case was used instead of the original one.

2) Weak form

We are interested in solving equation (3.2), which means that we want to find the function $c \in C^2(\overline{\Omega})$, which is the space of continuous functions with continuous first and second order derivatives in $\overline{\Omega}$. This form is referred to as the strong form of the problem because we are imposing "strong" constraints on the space where the solutionis defined.

However, we can manipulate the strong formulation of (3.2) to obtain an equivalent form easier to be solved by relaxing the constraints on the solution space so that c no longer needs to belong to $C^2(\overline{\Omega})$.

Let us choose a certain arbitrary function v. Because we can choose any function that we want, this is not a problem. This function will also be referred to as "test function". We now multiply the original problem by this test function and integrate over the domain.

We get:

$$0 = \int_{\Omega} (-D\Delta c + \mathbf{v} \cdot \nabla c) v d\Omega \tag{3.3}$$

We can then now integrate the second order term by parts by using the relation:

$$\nabla \cdot [(D\nabla c)v] = [\nabla \cdot (D\nabla c)]v + D\nabla c \cdot \nabla v,$$

which yields the equation:

$$0 = \int_{\Omega} \big(\nabla \cdot [(-D\nabla c)v] + D\nabla c \cdot \nabla v + \mathbf{v} \cdot \nabla c v\big) d\Omega \tag{3.4}$$

We now use the divergence theorem (also referred to as the Green-Ostrogradski theorem), which allows us to transform a volume integral into a surface one, being:

$$\int_{\Omega} \nabla \cdot [(-D\nabla c)v] d\Omega = \int_{\partial\Omega} \mathbf{n} \cdot [(-D\nabla c)v] ds,$$

where $ds = d\partial\Omega$ is a surface differential on the boundary. By applying this relationship to the first term in equation (3.4), we obtain:

$$0 = \int_{\partial\Omega} \mathbf{n} \cdot [(-D\nabla c)v] ds + \int_{\Omega} \big(D\nabla c \cdot \nabla v + \mathbf{v} \cdot \nabla c v\big) d\Omega \tag{3.5}$$

We now have a term which is integrated over the boundaries of the domain. Let us rewrite separately this boundary contribution by taking into account the different boundary portions:

$$\int_{\partial\Omega} \mathbf{n} \cdot [(-D\nabla c)v] ds = \int_{\partial\Omega_D} \mathbf{n} \cdot [(-D\nabla c)v] ds + \int_{\partial\Omega_N} \mathbf{n} \cdot [(-D\nabla c)v] ds + \int_{\partial\Omega_R} \mathbf{n} \cdot [(-D\nabla c)v] ds$$

Each of these terms is treated differently:

1: Dirichlet Boundary conditions

$$\int_{\partial\Omega_D} \mathbf{n} \cdot [(-D\nabla c)v] ds = 0$$

Because we already know the value of the solution in the part of the boundary where we apply boundary conditions and we have said that we can choose our test function as we please, we set the test function to be 0 in this part of the domain given that we don't need to test the solution. Therefore, v=0 on $\partial\Omega_D$ and this term disappears from the integral.

2: Neumann Boundary conditions

$$\int_{\partial\Omega_N} \mathbf{n} \cdot [(-D\nabla c)v]ds = \int_{\partial\Omega_N} -D\frac{\partial c}{\partial \mathbf{n}}vds$$

Because in our pollution transport problems Neumann conditions are homogeneous such that $D\frac{\partial c}{\partial \mathbf{n}} = 0$, we can substitute this value into the integral to find:

$$\int_{\partial\Omega_N} -D\frac{\partial c}{\partial \mathbf{n}}vds = \int_{\partial\Omega_N} 0vds = 0$$

3: Robin Boundary conditions

$$\int_{\partial\Omega_R} \mathbf{n} \cdot [(-D\nabla c)v]ds = \int_{\partial\Omega_R} -D\frac{\partial c}{\partial \mathbf{n}}vds$$

Similarly to Neumann conditions we have an expression which can be directly introduced into the expression above being $D\frac{\partial c}{\partial n} = -\beta c$, yielding one term which depends on the unknown concentration:

$$\int_{\partial\Omega_R} -D\frac{\partial c}{\partial \mathbf{n}}vds = \int_{\partial\Omega_R} \beta cvds$$

This Robin boundary condition is therefore the only boundary condition which provides an explicit contribution to (3.5). After introducing this term into the original form of the boundary conditions, we can rearrange (3.5) in a new form which includes, on the left hand side, terms containing the solution c and the test functions v, while the right hand side vanishes because all of terms which only depend on v are zero. Namely, this yields:

$$\int_{\Omega} (D\nabla c \cdot \nabla v + \mathbf{v} \cdot \nabla cv)d\Omega + \int_{\partial\Omega_R} \beta cvds = 0 \tag{3.6}$$

We can now properly select the test function v. If possible, the idea is to pick both c and v in the same function space V. We remark that both functions are derived only once in the weak form, which means they have the same constraints. The standard choice for V for a 3D problem coincides with a suitable set of the Sobolev space $\mathrm{H}^1(\Omega)$ [17], with:

$H^1(\Omega) = \{v \in L^2(\Omega) : \nabla v \in (L^2(\Omega))^3\}$

$L^2(\Omega) = \{v : \int_\Omega |v(x)|^2 dx < +\infty\}$

In particular, space V has to include the Dirichlet boundary terms so that we have:

$V = \{v \in H^1(\Omega); \gamma_0[v] = 0, \text{ on } \partial\Omega_D\}$

where $\gamma_0[v]$ is the trace of function v on the boundary [22]. This is the subspace of functions in the Sobolev space which vanish on the Dirichlet boundary as the solution at these points does not need to be tested.

Therefore, the weak form of the problem can be stated as follows:

$$\text{Find } c \in U \text{ such that } B(c, v) = F(v), \forall v \in V \tag{3.7}$$

Where $B(\cdot,\cdot)$ is a bilinear form on the space U x V corresponding to the left hand side of the equation (3.6) and $F(\cdot)$ is a linear form defined on the space V in this case identically equal to zero.

The only thing that needs to be done in order to proceed is to define the space U where the solution c will be sought. By taking a look at (3.6), we see that second derivatives of the solution no longer appear in the problem statement, which now instead only involves first derivatives in the distributional sense. This means that we are now looking for the solution in a larger space. Therefore, the constraints of the solution have been relaxed, as under this new problem formulation we now obtain solutions which we would not have considered in the strong form of the problem. This is the goal of recasting the equation in the weak form: we are now free to look for solutions in a different and larger vector space.

Starting from eq. (3.6), it becomes obvious that we take a space U which is quite similar in nature to space V given that the constraints on the derivatives are identical for both the test and trial functions, and define the new space as follows:

$$U = \{c \in H^1(\Omega); \gamma_0[v] = a, \text{ on } \partial\Omega_D\}$$

The only constraint that we impose here is that the trace of the solution on the Dirichlet boundary corresponds to the solution restricted to this boundary portion.

However, in order to simplify our problem, we would ideally like spaces U and V not only to be similar in nature but identical. For this purpose, a lift function is introduced. The lift function $\tilde{c}$ on Ω is such that $\tilde{c} = a$ on $\partial\Omega_{bottom}$. We also introduce a function w such that $c = w + \tilde{c}$. Then, we have that $B(c, v) = B(w, v) + B(\tilde{c}, v) = F$. By choosing $\tilde{c} = a, \forall \mathbf{x} \in \Omega$, we can now modify U:

$$U = \{c \in H^1(\Omega); \gamma_0[v] = 0, \text{ on } \partial\Omega_D\}$$

This modifies our weak form, which now becomes:

$$\text{Find } w \in V \text{ such that } B(w, v) = F(v) - B(\tilde{c}, v) \;\; \forall v \in V \tag{3.8}$$

Now that U=V and we can proceed.

3) Demonstrate well-posedness

It is not obvious that the solution to the weak form will exist and be unique. The existence and uniqueness of solutions of boundary value problems in the weak form is usually determined by the Lax-Milgram lemma [22]:

Lax-Milgram lemma. *Let V be a Hilbert space with inner product $(\cdot,\cdot)_V$ and associated norm $\|\cdot\|_V$ (i.e. such that $\|w\|_V^2 = (w, w)_V$), and let U=V. Moreover, it is assumed that the following conditions hold:*

- *Continuity of B: $\exists M>0$ such that $\forall u, v \in V$, $|B(u,v)| \leq M\|u\|_V\|v\|_V$;*
- *Continuity of F: $\exists C>0$ such that $\forall v \in V$, $|F(v)| \leq C\|v\|_V$;*
- *Coercivity of B: $\exists \alpha > 0$ such that $\forall u \in V$, $B(u,u) \geq \alpha\|u\|_V^2$*

Then, the boundary value problem is well-posed, i.e. there exists a solution, this solution is unique and it continuously depends on the data.

Thereby, by proving the three conditions demanded by the Lax-Milgram lemma to hold, we can prove the well-posedness of the weak formulation. However, proving continuity and coercivity specific analytical tools (trace theorem, Poincaré inequality, Sobolev embedding theorem, etc.) which have not been directly considered in this chapter. For the sake of completeness we explicitly prove the well-posedness of 3.6 in Appendix A. All the theory needed for understanding the proof can be found in [11] and [17] .

4) Galerkin method

A Galerkin dimension reduction will now be considered. This procedure consists in replacing the Hilbert space where we are looking for the solution in the weak form with a finite dimensional subspace. Therefore, we are now taking the opposite approach: while the idea of recasting the strong form of the equation in the weak form was to allow us to look for solutions in a larger space, we are now going to restrict the space where we will consider approximating our equation.

The new space will be noted V_h for the trial and test functions. If this Hilbert subspace is equipped with its corresponding scalar product, it is still a Hilbert space, and the well-posedness of the problem still holds [17].

Therefore, we now have a new formulation of our problem, which is referred to as the discrete formulation, which then reads:

$$\text{Find } c_h \in V_h \text{ such that } B(c_h, v_h) = F(v_h), \forall v_h \in V_h \tag{3.9}$$

We define the dimension of the discrete space V_h as N_h, such that $N_h \underset{h\to 0}{\longrightarrow} +\infty$. If V_h is dense in V, we can then assume that the internal approximation converges to c for $h \to 0$ [17]. This means that:

$$||c - c_h||_{V_h} \underset{h\to 0}{\longrightarrow} 0$$

which means that as h gets closer to 0, the approximated solution c_h converges to the actual solution c or, likewise, that we can approximate c with the accuracy that we desire by increasing the dimension of the discrete space.

From this part on, to simplify notation, test functions v_h will be renamed as N.

The Galerkin method is the most classical method used to find an approximation of the problem (3.9). This method consists of two main steps:

1) The solution c is interpolated by $c_h = \sum_{i=1}^{M} c_i N_i$ which acts as the trial function.

2) Test the equation for all basis functions $N_i \in V_h$

Physically, what this means is that we choose a finite number of points M within our domain where we will actually solve our equation. Each points has a finite number of associated degrees of freedom. Thereafter, we can obtain the concentration in any point of the domain by computing the weighted sum of the concentrations in the entire domain times their basis functions N_i, hereafter referred to as shape functions. Shape functions therefore act as interpolating functions, as they allow us to compute the solution at the points outside those where we have actually computed the solution. Nevertheless, we

cannot choose whatever basis functions we want. Instead, we will define N such that it is zero outside the finite element which contains it.

Each point j where the solution is computed corresponds to a value of N_i=1 if $i = j$, and the shape function has a value of N_i=0 on all the other nodes of the element ($i \neq j$). Therefore, for a given node x_j:

$c_j = \sum_{i=1}^{M} c_i N_i(x_j) = c_1 \times N_1(x_j) + c_2 \cdot N_2(x_j) + ... + c_j \cdot N_j(x_j) + ... + c_N \cdot N_N(x_j) = c_1 \times 0 + c_2 \cdot 0 + ... + c_j \cdot 1 + ... + c_N \cdot 0 = c_j$

which is exactly the result that we expected to find.

3.3 Finite element approximation

Up until now, we haven't defined what space V_h is. The Finite Element approximation is a specific type of Galerkin method which consists in taking space V_h in (3.6) to be the space of piecewise polynomial functions.

By applying the finite element approximation to equation (3.6), we obtain the following problem. Find c_i, with $i = 1, ...M$, such that:

$$\sum_{i=1}^{M} \left(\int_{\Omega} D\nabla N_i \cdot \nabla N_j + \mathbf{v} \cdot \nabla N_i N_j d\Omega + \int_{\partial\Omega_R} \beta N_i N_j ds \right) c_i = 0 \qquad \forall j = 1, .., M \tag{3.10}$$

This form is referred to as the discrete form. From here on, we need to continue manipulating the equation given that (3.10) cannot be computed just yet with a computer. However, the manipulations no longer require changing the essential problem formulation, as the solutions will be sought in V_h. But by applying a few further steps which do not affect the unknowns c_i, (3.10) can be recast in a much more practical form.

3.4 Mesh and tetrahedral elements

The mesh is a fundamental concept in the FEM. A 3D mesh consists in the division into smaller volumes of the 3D space on which we are approximating our PDE. These smaller volumes will hereon be named "elements".

In theory, our elements are not restricted to any particular shape, and we can choose any volume we please: cubes, pyramids, octaedrons... However, certain shapes have easier analytic expressions which make solving the equation simpler. For this reason, the domain has been decomposed into tetrahedral elements, which are one of the most convenient element types. A sample computational domain divided into tetrahedral elements can be seen in Figure 3.5.

The defining points of the solution are the nodes. In our particular examples, the nodes are the vertices of the tetrahedra which define our domain. This is because linear finite elements have been used for solving the problem, which means that the only nodes where the solution is computed are the vertices of the polyhedra. More generally, nodes coincide with vertices if we are using first order linear finite elements, whatever shape

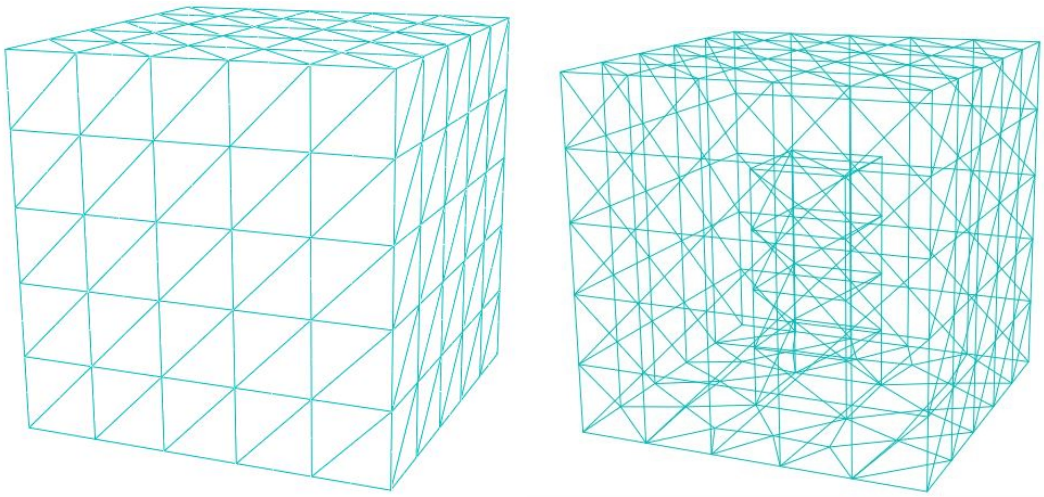

Figure 3.5: Synthetic computational domain of urban area meshed with tetrahedral elements. View of the exterior faces (left) and all the faces (right).

and dimension these may have. Nevertheless, it is also possible to use finite elements of higher order, in which case, nodes are no longer restricted to vertices, as further nodes appear at the middle of the edges of the element or inside the elements. But given that we are working with linear tetrahedral elements, the terms node and vertex will be used interchangeably.

There are a few considerations to take into account when choosing the size of our elements, which defines how far apart the nodes are found:

- We would ideally like to compute the solution in as many points as possible so as to have more accurate approximations. This results in a smaller mesh size, which in turn increases the computational cost. Therefore, users generally have to make compromises, trading off accuracy for computing efficiency.
- Our choice of mesh not only impacts the accuracy of the solution, but also the shape of our domain. For instance, if we are using linear paralallelepipeds as elements, approximating curved surfaces requires small elements to simulate the curving effects. The same happens if we are trying to capture very small spatial features which may be relevant for the solution. Therefore, meshes are also determined by the shape of the domain we are working with.

In our case, we are working with the mesh of an urban area. In a synthetic mesh such as the one in figure 3.5, we can use quite large elements as there are no curved surfaces, so the only limitation we deal with is the computational one. However, in real urban areas, which may include curved streets and buildings of very different sizes, it is often necessary to refine the mesh (i.e. use elements of smaller size) locally to capture these features.

Once we have divided our computational domain into suitable elements, we have to solve the equation for each element. However, the most usual approach consists in defining a standard reference tetrahedral element and doing all the computations on this "natural" space, which is specifically chosen to make the derivation and integration as easy as possible. Once the solution has been found, a suitable mapping is employed to bring the solution back to the real space and scale it to the size of the corresponding element as pictured in Figure 3.6. This greatly simplifies calculations as we don't need to

compute local derivatives for each element, instead having all the data readily available for a standard element and transporting the results to the actual space we are working in once they have been computed.

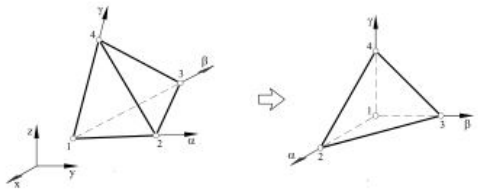

Figure 3.6: Mapping of a tetrahedral element to the reference element. Source: [20]

Technical details for this procedure can be found in Appendix B. This results in shape functions N_i being converted into functions $\hat{N}_i$ which refer to the reference elements and in the use of Jacobian matrices which perform the change of coordinates from local (i.e. belonging to the element of reference in a generic space) to global (i.e. belonging to the domain of the PDE). $\mathbf{J}^{(e)}$ will be used to refer to the Jacobian matrix used to change the element volume coordinates for local to global, while $\mathbf{J}_s^{(e)}$ is the Jacobian for the element faces.

Then we convert the integrals into weighted sums using Gaussian quadratures. This is because computers are unable to compute integrals analytically (i.e. by performing actual explicit integration), so definite integrals are approximated by multiplying the value of the function at certain points within the element domain times certain tabulated weights. This results in the followind approximation:

$\int_K f(\mathbf{x})dK \approx \sum_{k=1}^{n} w_{n,k} f(\lambda_{0,n,k}, ..., \lambda_{d,n,k})$

where K is the element we are integrating over (in our case, K is a tetrahedron if we are integrating over an element and a triangle if we are integrating over an element face), $w_{n,k}$ is a weight which can be found in a table, n refers to the order of the integration (in our case, we will choose $n = 1$, integration with a single point, which yields an exact solution for linear tetrahedra) and k defines the type of element that we are integrating over (so, for instance, $k = 4$ defines a tetrahedron because it has 4 points).

The exact procedure used to get the final form of the problem is also explained in Appendix B.

This results in the following starting form for equation (3.6):

$$\sum_{i=1}^{M}(\sum_{k=1}^{4} w_{1,k}(D(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j+\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j)|\mathbf{J}^{(e)}|+\sum_{k=1}^{3} w_{1,k}\beta N_i N_j|\mathbf{J}_s^{(e)}|)c_i = 0 \tag{3.11}$$

This is an expression which can now be understood by a computer and can be used as a basis for approximating our PDE via the FEM.

3.5 System of equations and treatment of Dirichlet boundary conditions

We have gone from integrals over the entire domain in equation (3.2) to a discrete system of equations with as many equations as nodes times the number of unknowns per node in (3.11). Because (3.11) has been fully discretized, we end up with a system of linear equations which can be recast in matrix form. Each of the equations in the system solves the PDE for a given node, so the contributions of all the elements in the domain which share that node must be added.

Approximations for the solution in the elements are computed separately and a local matrix of size 4x4 is obtained for each tetrahedron in the case of volume integrals, and of size 3x3 when integrating over the triangular faces. If there were surface terms on the right hand side, their integration would yield yield 3x1 vectors, but in our case, the right hand side is null so no integration is performed. However, these local matrices and vectors then need to be assembled into a single global matrix $\mathbf{K}$ in the case of the left hand side, and a single global vector f in the case of the right hand side. This final step before solving the equations is called assembly, and involves the positioning of the individual element matrix terms inside the global matrix. This is done by mapping each node in global numbering to the corresponding node in each element when locally numbered.

For generic steady problems, the resulting equations are then obtained in a matrix form which reads: $\mathbf{Ku} = \mathbf{f}$, where $\mathbf{K}$ is referred to as the stiffness matrix, $\mathbf{u}$ as the vector of unknowns and $\mathbf{f}$ the load vector. Because in our problem there is only one unknown per node (the concentration), the vector of unknowns $\mathbf{u}$, which is relabeled as $\mathbf{c}$, is a vector with one column and as many rows as the number of unknowns in the problem. The corresponding terms in matrices $\mathbf{K}$ and $\mathbf{f}$ are those shown in (3.11). More generally, stationary linear PDE problems can be discretized and recast as linear systems involving a stiffness matrix, a load vector and a vector of unknowns.

By solving the system of advection-diffusion equations $\mathbf{Kc}$=$\mathbf{f}$, the expression of $\mathbf{c}$ yields the solution of the problem at the nodes. Because matrices $\mathbf{K}$ and $\mathbf{f}$ can be obtained from the known terms, the solution of the system is conceptually easy once we have $\mathbf{K}$ and $\mathbf{f}$, for very large systems, computing the result can be quite computationally costly. We want our matrix $\mathbf{K}$ to be as sparse as possible so that the matrix is computationally cheap to store. This is why we chose basis functions characterized by a local support.

Although the solution of the system can be conceptually expressed as $\mathbf{c} = \mathbf{K}^{-1}\mathbf{f}$, this system is solved without inverting the matrix, an operation which is always avoided in computational methods. This is due to the fact that the inverse of a sparse matrix is generally not sparse, which could lead to very costly operations as well as a lack of storage if we tried to compute it directly. Instead, these systems are solved via specific algorithms and operators which depend on the selected programming language.

A special mention should be given to Dirichlet boundary conditions. When finding the weak form of our problem, we said that we would make the test function zero on the Dirichlet boundary because the solution there was known. This is effectively the case, which means that by the time matrix $\mathbf{K}$ is assembled, the Dirichlet terms have not been taken into account. This is done a posteriori and right before solving the system of equations by performing an operation known as matrix reduction.

The idea is the following: we actually don't need to solve the equations for the nodes where we have imposed Dirichlet boundary conditions, because we already know what the

solution at those nodes is going to be. Therefore, we can reduce our matrix correspondingly by imposing those values at the corresponding nodes and making our system smaller. This is done via the following steps:

1) Eliminate the rows of the stiffness matrix which correspond to the nodes with imposed Dirichlet boundary conditions. Also eliminate the corresponding term from the force vector.

2) Take the columns of the stiffness matrix, multiply them times their corresponding Dirichlet boundary conditions, which are known, and then move the term to the right hand side of the equation, as it is no longer an unknown.

As an example, let's assume we have a system of equations expressed as **Kc**=**f** of the following shape:

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & ... & k_{1n} \\ k_{21} & k_{22} & k_{23} & k_{24} & ... & k_{2n} \\ k_{31} & k_{32} & k_{33} & k_{34} & ... & k_{3n} \\ k_{41} & k_{42} & k_{43} & k_{44} & ... & k_{4n} \\ \vdots & \vdots & \vdots & \vdots & ... & \vdots \\ k_{n1} & k_{n2} & k_{n3} & k_{n4} & ... & k_{nn} \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ c_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_n \end{Bmatrix}$$

We assume that we have Dirichlet boundary conditions p and q applied on globally numbered nodes 2 and 3 respectively. Therefore, c_2 and c_3 are no longer unknowns and take on certain concentration values imposed by the boundary conditions. We remove the rows in the system corresponding to nodes 2 and 3, and then move the corresponding columns of the stiffness matrix to the right hand side multiplied by the boundary value which is imposed there. This results in the following reduced system:

$$\begin{bmatrix} k_{11} & k_{14} & ... & k_{1n} \\ k_{41} & k_{44} & ... & k_{4n} \\ \vdots & \vdots & ... & \vdots \\ k_{n1} & k_{n4} & ... & k_{1n} \end{bmatrix} \begin{Bmatrix} c_1 \\ c_4 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} f_1 - k_{12}p - k_{13}q \\ f_4 - k_{42}p - k_{43}q \\ \vdots \\ f_n - k_{n2}p - k_{n3}q \end{Bmatrix}$$

We now solve this last system, which yields the solution at all the nodes except for the ones where we have imposed Dirichlet boundary conditions. However, the solution at these nodes does not need to be computed given that it is already known.

Because Neumann and Robin boundary conditions appear in the weak form of the problem and are already part of the equations to be solved, it is not necessary to perform any kind of manipulation to enforce them.

3.6 Stabilization

While solving the advection-diffusion problem by the application of the FEM would yield a certain solution, there will be some cases in which the solution of the approximated problem will differ significantly from the real solution. This phenomenon most frequently happens

in advection-dominated problems. When advection effects become more important than diffusive ones, solutions may start to display non-physical spurious oscillations: without any warning, the concentration will start becoming arbitrarily negative and positive. The intuitive explanation of this is the following: when we use a Galerkin formulation, in which the shape functions don't prioritize any given node, we are giving as much importance to what is happening upstream as to what is happening downstream. However, in advection-dominated problems, there is a clear flow of information from one direction towards the other due to the existence of a strong velocity field. If we try to solve the problem "symmetrically" (i.e. by ignoring the existence of a preferred direction), the solution will begin to display a strange behaviour in the preferential direction.

Advection-diffusion problems begin displaying this non-physical behaviour when the Péclet number, which is defined as the ratio of advection to diffusion while taking into account the size of the mesh, exceeds a critical value. There are two main definitions of this number, the global Péclet number (where the size of the entire mesh is taken into account) and the local Péclet number (which only takes into account the characteristic dimension of a single element in the mesh). For stability purposes, we are interested in the local Péclet number, which is defined as follows:

$$Pe = \frac{\nu h}{2D},$$

where ν is the module of the velocity in that element keeping consistency with notation introduced in equation (2.5), h is the characteristic dimension of the element and D is the diffusivity coefficient. Advection-diffusion problems usually start displaying oscillations when the Péclet number becomes greater than 1. The higher the Péclet number, the more advection predominates over diffusion and the more unstable our problem becomes, with the oscillations reaching more extreme positive and negative numbers.

For instance, in Figure 3.7 we show a synthetic computational domain with the shape of a cube. When imposing a wind field which is strong enough to make the Péclet number larger than one, then the concentration inside our domain oscillates and displays a behaviour which very clearly does not correspond to what is observed in reality. The concentration then reaches both extremely high values, which is impossible due to the fact that the maximum concentration is reached at the bottom, as well as negative concentrations, which has no physical meaning.

Therefore, to avoid the oscillatory behaviour we are interested in solving problems with small Péclet numbers, ideally under 1. However, if we compute an estimation for the Péclet number that we would expect to see in a problem of advection-diffusion in an urban area with some typical parameters, we see that this is not the case to be expected.

The velocity of the wind in a city will be assumed to be within the order of magnitude of 2 m/s, while the diffusivity of pollutants in air will be taken at 10^{-4} m^2/s . The mesh is a priori controlled by the user, but it cannot be arbitrarily small. For meshing a small part of an urban area which can fit inside a 100m x 100m x 100m cube, if 100 nodes are taken on each side, thereby yielding a node side of roughly 1m, we would have roughly 100^3 nodes, or 1 million nodes, which can already be considered a fine mesh. Therefore, we will assume a characteristic length of 1m as an initial estimate. This yields a Péclet number of:

$$Pe = \frac{2 \cdot 1}{2 \cdot 10^{-4}} = 10^4,$$

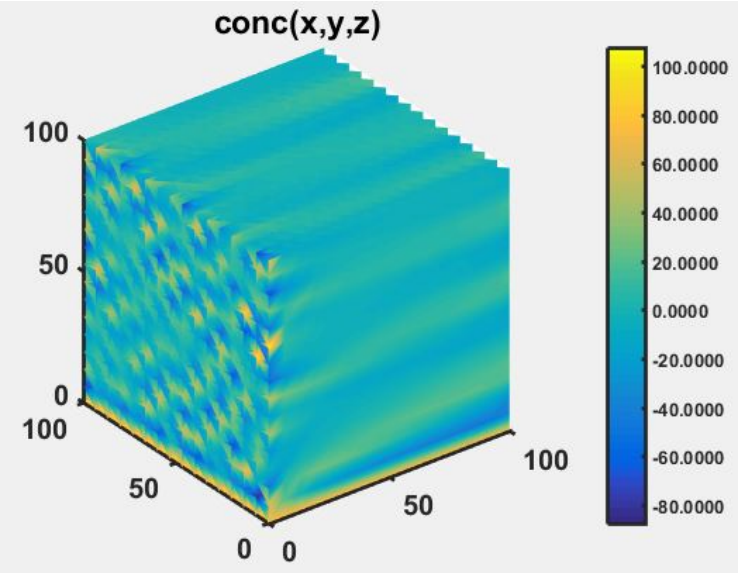

Figure 3.7: Sample oscillations for a not stabilized advection-diffusion problem with a local Péclet number of roughly 60

meaning that, if we solved the problem with these estimated quantities, the Péclet number would be 4 orders of magnitude higher than the one which ensures an oscillation-free solution. We can try to decrease the Péclet number by checking if different values for each of the parameters which come into play at the definition can be assumed while maintaining the physical soundness of the model:

- Decreasing the velocity: because we want to consider the full spectrum of realistic velocity modules when solving the PDE, we cannot limit the velocity to cases which are convenient. We must instead work with the entire range, for velocities ranging from 0.1 to 10 m/s.
- Increasing the diffusivity: the diffusivity is a physical parameter where some freedom can be taken, but which can only be increased until the problem under consideration no longer corresponds to the reality. This typically happens well below the diffusivity value that would be needed to reach a Péclet number of 1.
- Refining the mesh: this is the most common approach, given that the mesh is not governed by any physical considerations and is often chosen by the user. However, reducing the mesh size increases the number of equations to compute, making the problem more computationally expensive. For a small increase in accuracy, we may need to solve many more equations, which soon makes this strategy impractical as the simulations begin taking up too many computational resources. Furthermore, in some applications the mesh is given and cannot be changed. This is often the case if we are working with a mesh which has been generated by third parties and not by ourselves.

None of the three options is feasible if we want to systematically obtain an accurate approximation which behaves according to physical laws with an acceptable computational effort. To solve the numerical difficulties associated with the solution of the advection-diffusion problem using the FEM in the presence of a high Péclet number, a stabilization scheme was implemented. This scheme is based on a modification of the discrete form to be solved which changes the nature of the problem by increasing the diffusion along

certain preferential directions, thereby increasing the Péclet number along certain flow lines. This removes the numerical non-physical oscillations, and yields a solution much more representative of the real one. By wisely choosing the directions along which we hope to stabilize the solution, we hope to obtain an approximation which does not differ much from the expected behaviour.

There are various ways to perform this stabilization. They can be broadly classified into two types: consistent and not consistent. On the one hand, not consistent stabilization schemes are those in which the amount of added diffusion does not depend on how close the numerical solution is to the analytic one. Therefore, the same quantity of diffusion is considered even when the solution does not need it. On the other hand, consistent stabilization schemes add less diffusion as the numerical solution gets closer to the exact one. In other words, if the solution is exact, no diffusion will be added. This makes the solution closer to the one we would expect. Therefore, consistent stabilization techniques will be prioritized.

The first consistent stabilization procedure used in this project is the Streamline Upwind Petrov-Galerkin (SUPG) method. The SUPG method adds a term to the discrete form. This term is a function of the residual, which is defined as follows for the pollution transport problem that we are interested in solving:

$$\Re(c) = -D\Delta c + \mathbf{v} \cdot \nabla c \tag{3.12}$$

If the solution is exact, the residual goes to zero. Therefore, consistent stabilization methods usually compute the diffusion to be added as a function of the residual. The general form of a consistent stabilized technique is the following:

$$B(c, v) + \sum_e \int_{\Omega^e} \wp(v)\tau\Re(c)d\Omega = 0 \tag{3.13}$$

where $B(u, v)$ corresponds to the terms in the left hand side of the weak form (3.9) and Ω^e refers to a given finite element. $\wp(v)$ is a given operator which is applied to the test function (its definition is what distinguishes different consistent stabilization methods), τ is a stabilization parameter and $\Re(c)$ is the residual of the differential equation as described in (3.12) [9].

SUPG is defined by the choice of the following test function operator:

$$\wp(v) = \mathbf{v} \cdot \nabla v$$

The weak form of the problem to be solved therefore becomes:

$$B(c, v) + \sum_e \int_{\Omega^e} (\mathbf{v} \cdot \nabla v)\tau(-D\Delta c + \mathbf{v} \cdot \nabla c)d\Omega = 0 \tag{3.14}$$

The discretized form of this problem is the same, but defined in a different functional space:

$$B(c_h, v_h) + \sum_e \int_{\Omega^e} (\mathbf{v} \cdot \nabla v_h)\tau(-D\Delta c_h + \mathbf{v} \cdot \nabla c_h)d\Omega = 0 \tag{3.15}$$

The residual term is not integrated by parts, and the second derivative of the concentration therefore remains. However, in this particular problem we are working with linear finite elements, which means that the second derivative of c_h is zero. For that reason, the equation is simplified, resulting in the following discretized form:

$$B(u_h, v_h) + \sum_e \int_{\Omega^e} (\mathbf{v} \cdot \nabla v_h)\tau(\mathbf{v} \cdot \nabla c_h)d\Omega = 0 \tag{3.16}$$

The addition of the term based on the residual adds diffusion in the direction of the velocity. The amount of added diffusivity therefore depends on the gradient, the velocity and the choice of parameter τ. While some analytic expressions for τ exist for very simple 1D and 2D problems, there is no closed expression for this parameter in 3D problems. τ is highly problem dependent, and different values should be tested to find the most suitable one, usually values which will depend on problem parameters. However, it is obvious that τ must vanish when the mesh size gets finer, given that SUPG is a consistent method, and solutions for infinitely fine meshes are exact. Therefore, τ will have a general expression of the shape:

$$\tau = \frac{\phi h^e}{2|\mathbf{v}|}$$

where ϕ is a coefficient and h^e is the mesh size for the element under consideration.

Coefficient ϕ is an empirical coefficient which the user can pick. A lot of research has been performed to find the best choice of coefficient under different circumstances. Best practices for 2D cases recommend taking ϕ such that [9]:

$$\phi = coth(Pe) - \frac{1}{Pe}.$$

It is often the case that this same approach is taken in 3D problems. However, this stabilization method can be insufficient in very high Péclet number situations such as the ones we are working with, where Pe $>>$ 1. As noted by Codina in [5], sometimes small oscillations appear in the direction perpendicular to the gradient of the transported concentration. This is usually due to the existence of interior or boundary layers in the solution, as these oscillations usually appear in narrow areas along these sharp layer, where SUPG has the most potential for over and undershooting [15, 27]. Although the solution of the velocity field with the potential flow model does not allow the development of velocity boundary layers, the sharp decrease in concentration as we go from the bottom to the top of the model generates an internal concentration layer, causing the solution to become unstable. As we can see in Figure 3.8, if we only implement SUPG without any additional terms, instabilities happen in the crosswind direction (meaning, in the plane perpendicular to the wind direction) in the vicinity of the sharp interior concentration layer, although the solution is much more stable with respect to that of figure 3.7 as the oscillations are much more damped and only occur locally instead of globally.

No matter how much SUPG diffusion is increased, these instabilities will remain, given that the stabilization is only carried out in the direction of the gradient itself and not in

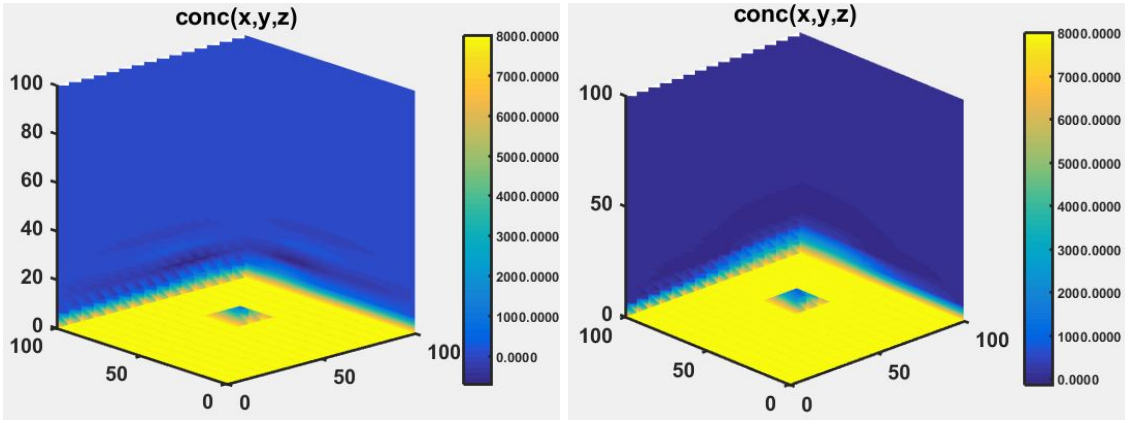

Figure 3.8: Concentration for regular SUPG (left) and SUPG with ϕ times 1000 (right). Instabilities (darkest blue) remain in the crosswind direction next to the concentration layer.

misaligned directions. Therefore, sometimes it is convenient to include a small term in the direction normal to the concentration gradient. This procedure, carried out by changing the weighting functions, is usually referred to as discontinuity capturing or shock capturing.

The idea behind discontinuity capturing is to introduce additional crosswind dissipation only in areas where streamline diffusion proves insufficient in damping the oscillations. The location of these problematic areas is a priori unknown, so it is determined using proxies for solution smoothness. In our case, we will consider this to be the norm of the residual of the solution obtained with the SUPG approximation. Therefore, it is still a strongly consistent method. Nevertheless, care must be taken, because the introduction of the norm of the residual into the test function operator of the new solution makes the problem non-linear, and adds a layer of additional difficulty.

As explained by Codina in [5], a reference the reader can refer to for a detailed theoretical explanation of how the expression is derived, we modify the test function operator and introduce another term to the left-hand side of the problem which adds diffusion in the crosswind direction. The crosswind direction is obtained by subtracting the normalized streamline velocity ($\frac{\mathbf{v}\otimes\mathbf{v}}{||\mathbf{v}||^2}$) from the identity matrix. The final modified discrete form of the stabilized advection-diffusion problem, including both SUPG and the crosswind stabilization, is the following:

$$\begin{aligned} &B(c_h, v_h) + \sum_e \int_{\Omega^e} (\mathbf{v}\cdot\nabla v_h)\tau_1(\mathbf{v}\cdot\nabla c_h)d\Omega + \\ &\sum_e \int_{\Omega^e} \Big(\frac{1}{2}\alpha_c^e h^e \frac{|\mathbf{v}\cdot\nabla c_h|}{|\nabla c_h|}\nabla c_h \cdot (\mathbf{I} - \frac{\mathbf{v}\otimes\mathbf{v}}{|\mathbf{v}^2|})\cdot\nabla v_h\Big)d\Omega = 0, \end{aligned} \tag{3.17}$$

where α_c^e is a stabilization parameter and $\mathbf{I}$ is the identity matrix. The quantity $\tau_2 = \frac{1}{2}\alpha_c^e h^e$ is chosen so that it is always smaller than the SUPG parameter τ_1, otherwise, crosswind diffusion would dominate over streamline diffusion. If this was the case, we would get an over-diffused solution, given that most of the stabilization needs to take place along the streamlines, which is the direction where the major numerical oscillations happen. Just enough diffusion has to be added in the crosswind direction as to stabilize the perpendicular oscillations, but this term should generally be much smaller than the streamline one.

Term α_c^e is a term which is defined as follows:

$$\alpha_c^e = max(0, C - \frac{1}{Pe^e}) \tag{3.18}$$

where C is a user-determined parameter and Pe^e is the local Péclet number for the element being considered.

3.6.1 Iterative scheme

The method used to solve the nonlinear problem induced by the introduction of crosswind diffusion is the Picard scheme, which is a particular case of the alternate direction scheme. The idea is the following: we want to find an iteration function Φ such that we can deduce c^{n+1} from c^n, where c^n is the solution found at iteration n ($c^{n+1} = \Phi(c^n)$). We start from an initial guess c^0 and from there we iterate until the solution no longer changes [16].

Before applying crosswind diffusion, our problem was linear, and could be written as a system of algebraic equations $\mathbf{Kc} = \mathbf{f}$. We now rewrite it and make its dependency explicit to show the nonlinearity. Our problem thus becomes:

$$\mathbf{K(c)c} = \mathbf{f}$$

Therefore, conceptually:

$$\mathbf{c} = \mathbf{K(c)}^{-1}\mathbf{f}$$

The idea behind the Picard method is to use the right hand side of the expression above as the iteration function Φ. Therefore, we have:

$$\Phi(\mathbf{c}) = \mathbf{K(c)}^{-1}\mathbf{f}$$

Which, by implementing an iterative procedure, becomes:

$$\mathbf{c}^{n+1} = \mathbf{K(c}^n\mathbf{)}^{-1}\mathbf{f}$$

Once solution $\mathbf{c}^{n+1}$ has been obtained, it is compared with solution $\mathbf{c}^n$ to determine if the iterative procedure should end. The stopping criterion usually has the following expression:

$$||\mathbf{c}^{n+1} - \mathbf{c}^n|| < \epsilon||\mathbf{c}^n||$$

where ϵ is a user-determined tolerance.

By recovering the original expressions of the bilinear and linear forms and exploiting the finite element basis, we obtain the following expression:

$$\sum_{i=1}^{M}\Bigg(\int_\Omega \Big(D\nabla N_i\cdot\nabla N_j + \mathbf{v}\cdot\nabla N_i N_j + (\tau_1\mathbf{v}\cdot\nabla N_j)(\mathbf{v}\cdot\nabla N_i) + \frac{1}{2}\alpha_i h_i\frac{|\mathbf{v}\cdot\nabla N_i c_i|}{|\nabla N_i c_i|}\nabla N_i\cdot(\mathbf{I} - \frac{\mathbf{v}\otimes\mathbf{v}}{|\mathbf{v}^2|})\cdot\nabla N_j\Big)d\Omega + \int_{\partial\Omega_R}\beta N_i N_j ds\Bigg)c_i = 0$$

Finally, we refer the terms in the problem to the reference element, perform a change of coordinates and integrate the terms numerically, and we reach the final configuration of the equation:

$$\begin{aligned}
&\sum_{i=1}^{M}\left(\sum_{k=1}^{4} w_{1,k}\Big(D(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j\Big)+\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j+\right.\\
&\Big(\tau_1\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j\Big)\Big(\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\Big)+\\
&\left.\frac{1}{2}\alpha_i h_i\frac{|\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i c_i|}{|(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i c_i|}(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\cdot(\mathbf{I}-\frac{\mathbf{v}\otimes\mathbf{v}}{|\mathbf{v}^2|})\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j\right)|\mathbf{J}^{(e)}|+\\
&\sum_{i=1}^{M_R}\Big(\sum_{k=1}^{3} w_{1,k}\beta\hat{N}_i\hat{N}_j|\Big)\mathbf{J}_s^{(e)}|=0
\end{aligned}\tag{3.19}$$

Equation (3.19) is the form that is actually implemented and solved by the computer. It is the form that a computer can actually work directly with: an algebraic system of equations where all the terms can be computed as weighted additions of scalars, vectors and matrices. The solution of this form of the equation should be as close as possible to the solution of equation (3.2). The details of the implementation will be provided in the following section.

3.7 Implementation of the FEM advection-diffusion problem using MATLAB

The final step is to implement the final discrete stabilized form of the advection-diffusion PDE so that a computer can perform the computations. For this, the MATrix LABoratory (MATLAB) language has been used. MATLAB is a programming language especially suited for operations with matrices, which is the preferred format for solving the advection-diffusion equations computationally using the FEM.

As it has been shown in the previous sections, the approximated solution of the stationary advection-diffusion equation can be expressed in FEM form as a linear system of equations, $\mathbf{Kc}=\mathbf{f}$ prior to the introduction of the crosswind diffusion, and a nonlinear system of equations $\mathbf{K(c)c}=\mathbf{f}$ after the introduction of the discontinuity capturing term. From now on, we will solve the nonlinear problem, but for the sake of keeping notation simple, the explicit non-linearity will be dropped from the notation that is, the nonlinear problem will be written as $\mathbf{Kc}=\mathbf{f}$.

By going back to the discretized equations (3.19), we see that $\mathbf{K}$ results from the addition of various terms, which conceptually correspond to different phenomena: diffusion, advection, absorption, streamline stabilization and crosswind stabilization. Therefore, the implementation of the different terms is actually carried out separately, as this will greatly simplify the application of reduced order models in the following chapter to the problem. This division results in different MATLAB scripts which compute different parts of the matrix, and finally add the individual terms to obtain the full system. Following physical and numerical considerations, the decomposition has been carried out as follows:

$$\mathbf{K}=\mathbf{K}_{diff}+\mathbf{K}_{conv}+\mathbf{K}_{supg}+\mathbf{K}_{cw}+\mathbf{K}_{rob}$$

$$\mathbf{f} = \mathbf{0}$$

where:

- $\mathbf{K}_{diff}$ corresponds to the diffusion part of equation (3.19):

$$\sum_{i=1}^{M}\left(\sum_{k=1}^{4} w_{1,k}\Big(D(\mathbf{J}^{(e,i)})^{-1}\hat{\nabla}\hat{N}_i \cdot (\mathbf{J}^{(e,i)})^{-1}\hat{\nabla}\hat{N}_j\Big)|\mathbf{J}^{(e,i)}|\right)$$

- $\mathbf{K}_{conv}$ comprises the terms related to advection:

$$\sum_{i=1}^{M}\left(\sum_{k=1}^{4} w_{1,k}\Big(\mathbf{v}\cdot(\mathbf{J}^{(e,i)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j\Big)|\mathbf{J}^{(e,i)}|\right)$$

- $\mathbf{K}_{supg}$ includes the stabilizing terms corresponding to the SUPG stabilization:

$$\sum_{i=1}^{M}\left(\sum_{k=1}^{4} w_{1,k}\Big(\tau_1\mathbf{v}\cdot(\mathbf{J}^{(e,i)})^{-1}\hat{\nabla}\hat{N}_j\Big)\Big(\mathbf{v}\cdot(\mathbf{J}^{(e,i)})^{-1}\hat{\nabla}\hat{N}_i\Big)|\mathbf{J}^{(e,i)}|\right)$$

- $\mathbf{K}_{cw}$ includes the stabilizing terms corresponding to the crosswind stabilization:

$$\sum_{i=1}^{M}\left(\sum_{k=1}^{4} w_{1,k}\frac{1}{2}\alpha_i h_i\frac{|\mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i c_i|}{|(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i c_i|}(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\cdot\Big(\mathbf{I}-\frac{\mathbf{v}\otimes\mathbf{v}}{|\mathbf{v}^2|}\Big)\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j\right)|\mathbf{J}^{(e)}|$$

- $\mathbf{K}_{rob}$ contains the Robin conditions as they appear in the left-hand side, which in our case correspond to the absorption of the buildings, which is proportional to the concentration of pollutant:

$$\sum_{i=1}^{M_R}(\sum_{k=1}^{3} w_{1,k}\beta N_i N_j|\mathbf{J}_s^{(e,i)}|)$$

Once the different terms have been computed, including the operations required to reach the final expression of $\mathbf{K}_{cw}$, the solution of a problem of the form $\mathbf{Kc}$=$\mathbf{f}$ is found by using the backslash \ operator: therefore, in MATLAB, the solution of operation $\mathbf{K}\backslash\mathbf{f}$ yields the same result as $\mathbf{K}^{-1}\mathbf{f}$ at a fraction of the computational cost. Remembering that this operation is performed with the reduced versions of the matrices corresponding to the nodes without Dirichlet BCs, the only things that's left is to expand the resulting vector $\mathbf{c}$ with the known values, and the computation of the concentration is complete.

Computing the FEM approximation of our problem is useful for two reasons: first, we need something against which we can compare the PGD results to determine accuracy. Given that the FEM is the method of choice of many people for solving the advection-diffusion equation, it makes sense to choose it as our control result. Second, as we will see in the following section, although we are going to use another method to solve these equations, the input for this method requires knowing the exact discretized expression of the matrices involved in our problem, $\mathbf{K}_{diff}, \mathbf{K}_{conv}, \mathbf{K}_{supg}, \mathbf{K}_{cw}$ and $\mathbf{K}_{rob}$. Therefore, the results above will be useful for the development of the following section.

Chapter 4

Proper Generalized Decomposition

4.1 Introduction to PGD

The Proper Generalized Decomposition (PGD) is a specific type of ROM. It is categorized as an "a priori" ROM given that the order reduction is carried out prior to the resolution of the problem, and it is a different version of the problem which is subsequently solved. It should be noted that "a posteriori" methods also exist, where the order reduction is performed on the solution so that it is less expensive to store and reconstruct. The main idea behind a posteriori model order reduction techniques is to obtain some snapshots of the solution, find the eigenmodes with the largest corresponding eigenvalues in the autocorrelation matrix and use them as the new basis for further simulations with different values for the problem parameters. These techniques have been applied in different branches of science and technology under different names, such as Principal Component Analysis (PCA), Singular Value Decomposition (SVD) or Karhunen-Loève theorem, to name a few [6]. While the details of these methods vary depending on their application, the basic idea is the same: using various empirical realizations of the problem under study with different parameters, compute an alternative basis to express the solution, and project the solution onto this new, optimized space where it can be described much more compactly. These methods are therefore usually referred to as projection-based methods.

The PGD approach is somewhat different. Its origins can be traced to the traditional method of separation of variables, also known as Fourier method, for the solution of PDEs. The PGD finds an approximation to the solution of the PDE without any prior information about the solution, unlike a posteriori methods. Let's consider a problem defined within a space of dimension D where the solution u is a field with $u(x_1, x_2, ..., x_D)$, where x_i denotes any coordinate of the problem, such as space or time, but can also include other problem parameters such as material coefficients, boundary conditions or geometric considerations to name a few. We will begin by considering a total separation, which means that an approximate solution u^N will be sought in the following form:

$$u(x_1, ..., x_D) \approx u^N(x_1, ..., x_D) = \sum_{i=1}^{N} F_i^1(x_1) \times ... \times F_i^D(x_D) = \sum_{i=1}^{N} \prod_{j=1}^{D} F_i^j(x_j)$$

This means that the solution is expressed as a summation of N products of D functions, each of the corresponding D problem coordinates. The N number of terms in the solution is a priori unknown, as are the different functions $F_i^j(x_j)$. The N terms are computed sequentially, beginning with 1, followed by 2, and so on until reaching N. Let's assume we want to compute the functions $F_i^j(x_j)$ at step n. At this point, we already know the solution of such functions for $i \leq n$ given that we have computed them at previous steps.

We can therefore use the information from the previously computed functions to find this new function in a procedure usually called successive enrichment. This means that thee terms in the sum are computed one after the other using information from all the other previously computed steps. For the first step, where no information is available, an initial approximation based on the boundary conditions of the problem is used.

The total number of terms N in the approximation is a priori unknown. If we wanted the solution to be exact, in the most general case we would need an infinite number of terms to obtain the best representation of the solution. However, this is not feasible nor practical, so we usually choose an appropriate truncation level to determine N by setting a stopping criterion for the enrichment.

To solve this problem at each step, we have to find the solution of the problem for each term we want to compute. However, the separated representation involves the multiplication of functions F_i^j which need to be determined, resulting in a non-linear problem independently on whether the original problem was linear or not. This means that an iterative scheme is needed to find the solution at each enrichment step [3]. The most common type of scheme used is the fixed point iteration scheme. It is assumed that the iterative solver will lead to convergence towards the actual solution of the problem.

The advantages of the PGD are numerous. One of the most interesting possibilities is the inclusion of additional problem parameters in the problem statement, something referred to as parametric PGD. This allows us to compute the solution of the problem only once for all values of the involved parameter within a given range, which provides a sort of "general" solution which can then be post-processed to obtain the results for any particular combination of values within the explicit range. This general solution is often referred to as the "computational vademecum" [4] and is carried out in what is typically referred to as an offline phase. Afterwords, finding the desired solution of the problem for a given combination of parameters only requires consulting this vademecum in what is called the online phase. Because the solution has already been computed, this is an extremely fast operation. This means that many of the traditional limitations of engineering and scientific simulations, for instance, the possibility of obtaining results in real time, can be overcome.

The parametric PGD strategy, however, involves increasing the number of dimensions in the problem with respect to those in the original problem (which typically don't include problem parameters). If tackled with traditional methods, this approach would lead to a rapid increase in computational complexity referred to as the "curse of dimensionality". However, the nature of PGD makes it a method capable of foregoing the curse of dimensionality as each of the parameters is solved separately instead of taking into account all of their possible combinations [32].

Up until this point, this introduction to PGD has used an example of total separation, where all the variables involved in the solution have been separated. However, this is not necessary in order to perform PGD. We can also choose to only separate a certain number of coordinates, especially if performing a full separation would be impractical due to implementation issues, physical interactions or the large number of terms needed to approximate the solution. In this case, we may prefer a partial separation, which would mean to consider, for a general unknown u:

$$u(x_1, ...x_D) \approx \sum_{i=1}^{N} F_i^1(\mathbf{x}_1) \times ... \times F_i^{D^*}(\mathbf{x}_{D^*}) \quad where \quad D^* < D$$

where the separated functions take values in low-dimensional spaces, meaning that $\mathbf{x}_i \in \Omega_i \subset \mathbb{R}^{q_i}$, with $q_i = 1$, 2 or 3 typically [6]. Everything that has been assumed for totally

separated variables can be applied to partially separated ones.

Finally, it should be noted that while PGD will be applied to find the solution to the advection-diffusion equation, it is by no means a method restricted to transport problems in environmental engineering. Other successful applications of the methods can be found in fields as diverse as geophysics [32], material science [23], virtual surgery [1] and magnetostatics [13] to name a few.

4.2 PGD for the advection-diffusion problem

The PGD will be applied to solving the advection-diffusion problem as presented in the problem statement. In this case, the unknown that we are tying to find is the concentration which approximates the solution of the strong form of the advection-diffusion equation with its corresponding boundary conditions. The PGD applied to this problem has been the parametric PGD: in other words, the objective is to introduce further parameters other than the spatial ones into the problem formulation to obtain a generic solution of the advection-diffusion problem as a function of these additional parameters. Therefore, there has been no separation of space performed - instead, two parameters have been added as extra problem variables. This results in a separation with 3 parameters: $\mathbf{x}, \theta$ and ν. $\mathbf{x}$ refers to the vector of 3D spatial variables (x, y, z) which are the coordinates of the FEM problem, while θ and ν refer to the velocity angle and module as defined in equation (2.5). A partial separation is therefore obtained, given that $\mathbf{x} \in \Omega_x \subset \mathbb{R}^3$, $\theta \in \Omega_\theta \subset \mathbb{R}$ and $\nu \in \Omega_\nu \subset \mathbb{R}$, where Ω_x is the spatial domain and Ω_θ and Ω_ν are closed intervals.

The idea is therefore to exploit a decomposition which is explicitly dependent on the velocity module and angle. The desired output is then a computational vademecum such that, when specific values for the velocity angle and module are plugged in, a solution for those parameters is directly obtained without having to perform any further numerical operations, allowing users to obtain results extremely quickly. This solution would allow us to visualize the pollution transport for the chosen velocity angle and module. The specific structure of our computational vademecum will be explained in the following section.

Therefore, the separation will be carried out under the form:

$$c(\mathbf{x}, \theta, \nu) \approx c^N(\mathbf{x}, \theta, \nu) = \sum_{i=1}^{N} F_i^1(\mathbf{x}) \times F_i^2(\theta) \times F_i^3(\nu)$$

where F_i^j is the separated function for parameter j. We want the approximated solution c^N to be as close as possible to the actual solution of equation (3.2). By taking the FEM solution to the equation as an adequate proxy of the solution, the accuracy of the PGD approximation can be measured by comparing the results obtained with both approaches.

The technical details regarding how the approximation of the solution is performed are provided in Appendix C. There, we can see the alternate direction scheme on which the implemented algorithm is based to compute the approximation. Given that the algorithm itself was not implemented, as it was available from existing codes, the appendix is left as an addition for the readers interested in seeing how an approximation of this kind is performed. For illustration purposes, the equations solved in Appendix C refer to the weak form of the unstabilized advection-diffusion problem as shown in equation (3.6). This is therefore the system which is solved for Péclet numbers which are lower than 1. The same procedure could be carried out for stabilized formulations allowing us to solve for any Péclet number.

The convergence of the PGD approach to the actual solution of the steady advection-diffusion problem, which is classified as an elliptic PDE, is proved in [10]. More generally, convergence of the PGD approximation towards the approximated solutions of the PDE obtained with other numerical methods can be proved for any elliptic PDE.

4.3 Algebraic PGD

In practice, the PGD has to be carried out by a computer and, just like for the FEM, the equations need to be recast in a form which can be easily interpreted and solved computationally. The PGD does not necessarily have to be combined with the FEM: it is a versatile tool which can use a variety of numerical methods for solving the equations, including the FEM but also finite differences and finite volumes. However, the type of PGD applied in this case has been algebraic PGD, which computes the solution using tensors as inputs. The tensors are based on the matrix form of the equations solved by the FEM. A separate discretization of the paramters must also be provided.

In the particular case of the algebraic PGD, the method aims to obtain an approximation of a tensor F of order d in a separable form given by:

$$F = \sum_{m=1}^{M} \tilde{\mathbf{f}}_1^m \otimes \tilde{\mathbf{f}}_2^m \otimes ... \otimes \tilde{\mathbf{f}}_d^m ,$$

or equivalently:

$$F = \sum_{m=1}^{M} \sigma_m \mathbf{f}_1^m \otimes \mathbf{f}_2^m \otimes ... \otimes \mathbf{f}_d^m ,$$

where vectors $\mathbf{f}_i$ are vectors of unit length and σ_m represents the amplitude of the corresponding term. This last expression is usually preferred because σ_m provides information determining the relative importance of each mode in the approximation, and is usually used to decide whether the solution has reach the desired level of accuracy [23].

The standard FE discretized formulation of the problem had a complexity expressed by the following formula:

$$C_{FULL} = n_{dof} \prod_{i=1}^{n_p} n_{d,i},$$

where n_{dof} is the number of degrees of freedom per node (only 1, which is the concentration, if we are solving the advection-diffusion equation), n_p is the total number of parameters which we are integrating over and $n_{d,i}$ refers to the dimension of each of the parameters. For a 3D stationary advection-diffusion equation, typically $n_p = 3$ corresponding to the three spatial coordinates and not including any other parameter. $n_{d,i}$ then refers to the number of discrete values d that are taken for each of the spatial coordinates i .

However, the algebraic PGD approximation shows a different complexity C_{PGD}:

$$C_{PGD} = n_{RankOne}(n_{dof} + \sum_{i=1}^{n_p} n_{d,i}),$$

where $n_{RankOne}$ is the total number of rank one approximations (terms in the summation of the separated approximation). Therefore, the complexity of the solution is mostly affected by the number of rank one approximations: however, as long as these remain below the thousands, for a 3D problem, we can typically expect $C_{PGD} << C_{FULL}$. So even if the PGD problem is non-linear, its complexity is much lower than for linear problems, which typically makes it a less expensive option even if the solution of non-linear problems poses its own set of computational challenges.

4.4 Application of algebraic PGD to the case study

By applying algebraic PGD to our problem, we hope to express our general solution as a tensor resulting from the summation of generic vectors which express the solution modes times a set of parametric functions:

$$\mathbf{C} \approx \mathbf{C}^n = \sum_{m=1}^{n} \mathbf{c}^m G_\theta^m(\theta) G_\nu^m(\nu)$$

From now on, to alleviate notation, the explicit dependency of parametric functions on their respective parameters will be dropped as the subscript will yield this information. Therefore, $G_\theta^m(\theta) = G_\theta^m$ and so on for all the other functions and parameters.

In particular, if we define an appropriate norm for $\mathbf{c}^m$ and parametric functions G, generally the L_2 norm (although other norms may be chosen if it makes sense in the context of the problem), we can normalize the terms in the expression above and reach an alternative configuration of the summation:

$$\mathbf{C} \approx \mathbf{C}^n = \sum_{m=1}^{n} \mathbf{c}^m G_\theta^m G_\nu^m = \sum_{m=1}^{n} \beta^m \tilde{\mathbf{c}}^m \tilde{G}_\theta^m \tilde{G}_\nu^m, \tag{4.1}$$

where $\beta^m = \|\mathbf{c}^m\|_{L_2} \|G_\theta^m\|_{L_2} \|G_\nu^m\|_{L_2}$ is the amplitude of term m of the summation and $\tilde{\mathbf{c}}^m, \tilde{G}_\theta^m, \tilde{G}_\nu^m$ are of unit norm.

The starting point for the application of algebraic PGD to the advection-diffusion problem is equation (3.19). In particular, as seen before, once we have the discretized expression we can easily recast the equation in a matrix form which can then be solved for algebraically. If we include the parameters as problem unknowns, the final matrix form reads:

$$\mathbf{K}(\boldsymbol{\mu})\mathbf{C}(\boldsymbol{\mu}) = \mathbf{F}(\boldsymbol{\mu}),$$

where $\boldsymbol{\mu}$ is a generic vector collecting the different problem parameters. This equation is of algebraic nature, and should be seen as already integrated in space. Therefore, as noted before, for this particular problem, $\boldsymbol{\mu} = (\theta, \nu)$. When referring to the two parameters simultaneously, this compact notation will be preferred. We drop the spatial coordinates from the problem parameters because we are already working with the discretized form, so they will simply continue to play the role of coordinates.

A special mention needs to be made regarding tensors $\mathbf{K}(\boldsymbol{\mu})$ and $\mathbf{F}(\boldsymbol{\mu})$. Because the algebraic PGD works with the matrices which are directly used to solve the advection-diffusion problem, $\mathbf{K}(\boldsymbol{\mu})$ and $\mathbf{F}(\boldsymbol{\mu})$ are actually the reduced versions of the full corresponding tensors as seen in equation (3.19). This corresponds to the matrices in the expression after having performed matrix reduction to eliminate the terms corresponding to Dirichlet boundary conditions and bringing them to the right hand side of the equation. Therefore, $\mathbf{F}(\boldsymbol{\mu})$ is no longer zero as terms from tensor $\mathbf{K}(\boldsymbol{\mu})$ appear on the right hand side as well to account for Dirichlet boundary conditions.

To express the algebraic advection-diffusion equation in integral form, we use its residual:

$$\Re(\mathbf{C}(\boldsymbol{\mu})) = \mathbf{F}(\boldsymbol{\mu}) - \mathbf{K}(\boldsymbol{\mu})\mathbf{C}(\boldsymbol{\mu}),$$

and integrate it in its parametric dependence using a weighted residuals approach:

$$\int_{\Omega_\theta}\int_{\Omega_\nu} \mathbf{U}^*(\boldsymbol{\mu})\Re(\mathbf{C}(\boldsymbol{\mu}))d\theta d\nu = 0, \tag{4.2}$$

where $\mathbf{U}^*$ is a modified test function. Integration is only performed for non-spatial parameters.

The PGD solver discretizes and solves the separated form of equation (4.2). To perform the integration, bounded intervals have to be provided for parameters θ and ν such that $\theta \in \Omega_\theta \subset \mathbb{R}$ and $\nu \in \Omega_\nu \subset \mathbb{R}$, and these intervals have to be discretized explicitly. While the discretization of the spatial coordinates in the FEM form of the problem was performed via the mesh, we don't have a priori discretizations of our problem parameters. We therefore have to choose a number of points within the intervals of definition of our parameters, at which the solutions will be found. This choice is important for appropriate convergence: if our discretization is too coarse, the algorithm may try to infer information from two solutions which are very different from one another, resulting in a bad separation. We ideally want to discretize the parameters so that the evolution of the solution between consecutive values of the discretization can be captured by the modes in the approximation. Once we have obtained the solution for the points in our discretization, if we want to obtain the approximated results for values of the parameters not included in the discretization, we will interpolate linearly.

The approximation of the solution is performed using a greedy strategy. This means that the different terms in the approximation are computed successively, and we don't know a priori how many terms there will be. The algorithm will keep on enriching the approximation $\mathbf{C}^n$ until a certain stopping criterion is met, at which point the solution will be considered as accurate enough as will be said to have converged. We begin by computing $\mathbf{C}^1$, then $\mathbf{C}^2$, and we check if the convergence criteria is met. We then compute $\mathbf{C}^3$, and check for convergence. This process continues until we have an appropriate (i.e. accurate enough) approximation. The fact that more modes lead to greater accuracy for the steady-state advection-diffusion equation is ensured by the proved convergence of the PDG approach for elliptic problems.

The accuracy of the approximation is based on coefficient β^m in the last term of the current approximation as defined in equation (4.1). Let us assume that we have computed $n-1$ terms in the approximation, so that our current approximation of the solution is:

$$\mathbf{C}^{n-1} = \sum_{m=1}^{n-1} \beta^m \tilde{\mathbf{c}}^m \tilde{G}_\theta^m \tilde{G}_\nu^m = \sum_{m=1}^{n-2} \beta^m \tilde{\mathbf{c}}^m \tilde{G}_\theta^m \tilde{G}_\nu^m + \beta^{n-1}\tilde{\mathbf{c}}^{n-1}\tilde{G}_\theta^{n-1}\tilde{G}_\nu^{n-1}$$

and we want to judge whether the approximation is already accurate enough. This is done by taking β^{n-1} and checking its value with respect to that of β^1. We check the following criterion:

$$\frac{\beta^{n-1}}{\beta^1} \leq \epsilon_{PGD},$$

where ϵ_{PGD} is a user-determined tolerance. If the expression holds true, then the algorithm stops. Otherwise, the current approximation of $\mathbf{C}$ (which here would be $\mathbf{C}^{n-1}$) is judged not to be accurate enough and the next term is computed.

Conceptually, what the expression above means is that we want to know when the approximation is capturing modes with an amplitude which is below a certain importance. Assuming for instance, that ϵ_{PGD} is 10^{-4}, we would stop the approximation when the amplitude of the $(n-1)$-th term is equal to or lower than 0.01% of the amplitude of the first term in the approximation. Because the amplitude of the terms is directly related to the importance of the effects they capture on the solution, this means that we will stop considering any effects which have 4 orders of magnitude less than the main one as we will consider that they have a negligible impact on the overall solution of the problem.

We now assume that we have computed the first $n-1$ terms in the approximation, the convergence criterion above has been checked, and it has been determined that the solution is not yet accurate enough. The enrichment must then continue, and term n in the approximation must be computed. Based on the definition of $\mathbf{C}$:

$$\mathbf{C}^n = \mathbf{C}^{n-1} + \mathbf{c}^n G_\theta^n G_\nu^n$$

Because we already know $\mathbf{C}^{n-1}$, solving this problem boils down to finding $\mathbf{c}^n$, G_θ^n and G_ν^n such that the new approximation C^n fulfills equation (4.2). The unknown term is said to be of rank one because it is computed as the product of functions.

It must be noted that this new term to be computed is non-linear given that it can be expressed as a product of unknowns. This may seem like a step back from the original FEM problem, which was linear. However, as it has been noted, this increase in the complexity is usually compensated by the lower rank of the approximation so that the overall number of computations remains lower than for the full problem as long as the number of terms in the approximation is kept relatively low.

The term $\mathbf{c}^n G_\theta^n G_\nu^n$ is computed using an alternate direction scheme. A full example of explicit computation of the n-th term of an algebraic PGD problem can be found in [23] while the algorithm which was used for the resolution of this particular problem has been laid out in detail in [7]. It is similar to the procedure described in Appendix C, but instead of working with the weak form, the resolution is carried out for the algebraic form of the equation.

4.4.1 Separated form of the advection-diffusion equation

Unlike the FEM version of the advection-diffusion problem, where a code was written in MATLAB specifically for this problem, the PGD approximation was computed using existing routines. In order to apply the algebraic PGD procedure as described in [7], the input data $\mathbf{K}(\boldsymbol{\mu})$ and $\mathbf{F}(\boldsymbol{\mu})$ have to be expressed in separated form. For our particular problem, it follows:

$$\begin{aligned} \mathbf{K}(\boldsymbol{\mu}) &= \sum_{l=1}^{n_k} \mathbf{K}^l B_\theta^l B_\nu^l \\ \mathbf{F}(\boldsymbol{\mu}) &= \sum_{m=1}^{n_f} \mathbf{f}^m S_\theta^m S_\nu^l m \end{aligned} \tag{4.3}$$

where n_k and n_f are the number of terms which are needed to approximate the tensors $\mathbf{K}(\boldsymbol{\mu})$ and $\mathbf{F}(\boldsymbol{\mu})$ separately.

The interest of having written equation (3.19) explicitly therefore becomes evident: if we want to perform the separation of the variables of interest, having the available equations makes it much easier to do.

Going back to the MATLAB implementation of the advection-diffusion problem and the details of matrix reduction, we see that the FEM problem was decomposed into submatrices which capture the different physical phenomena and numerical additions. This boiled down to the following matrices and vectors as the final configuration right before finding the solution of the system of equations:

$$\mathbf{K}_{red} = \mathbf{K}_{diff,red} + \mathbf{K}_{conv,red} + \mathbf{K}_{supg,red} + \mathbf{K}_{cw,red} + \mathbf{K}_{rob,red},$$

and

$$\mathbf{f}_{dir} = -(\mathbf{K}_{diff,dir} + \mathbf{K}_{conv,dir} + \mathbf{K}_{supg,dir} + \mathbf{K}_{cw,dir} + \mathbf{K}_{rob,dir}) * a,$$

where the subscript 'red' is used to refer to the reduced version of the stiffness matrix (where the rows and columns corresponding to Dirichlet boundary conditions have been subtracted) and subscript 'dir' refers to the columns of the stiffness matrix corresponding to the non-zero Dirichlet boundary conditions (which only refer to a in the bottom boundary as defined in equation (3.2)).

Because we see that the same terms appear in both sides of the equation (left and right hand sides), we see that in the separated form (4.3), $B_\theta = S_\theta$ and $B_\nu = S_\nu$ for those functions which refer to the same matrix.

Our objective is now to recast the matrices which compose the global stiffness matrix into the form imposed by equation (4.3) so as to make them explicitly dependent on the parameters that we are interested in solving for (velocity module and angle). For some matrices, the process is relatively straightforward, while for others the mathematical manipulations which need to be performed are complex. The advantage is that the decomposition of the matrices noted above are valid for both the left and right hand sides, given that only the stiffness matrix needs to be taken into account to compute the solution due to the force vector being zero.

For two of the matrices, after writing their expressions it becomes clear that there are no velocities involved in their computation, so these matrices do not need to be modified from those in the original problem:

$$\mathbf{K}_{diff,red} = \sum_{i=1}^{M_{ND}} (\sum_{k=1}^{4} w_{1,k} (D(\mathbf{J}^{(e)})^{-1} \hat{\nabla} \hat{N}_i \cdot (\mathbf{J}^{(e)})^{-1} \hat{\nabla} \hat{N}_j) |\mathbf{J}^{(e)}|)$$

$$\mathbf{K}_{rob,red} = \sum_{i=1}^{M_R} (\sum_{k=1}^{3} w_{1,k} \beta N_i N_j |\mathbf{J}_s^{(e)}|)$$

where M_{ND} refers to the nodes where no Dirichlet BCs are applied and M_R denotes the nodes with Robin BCs.

The convection matrix can also be quite neatly separated as it only involves one

velocity term. We then substitute the velocity decomposition of equation (2.5) into the expression, separate the terms, and subsequently conveniently rewrite the expressions to arrive at the following conclusion:

$$\mathbf{K}_{conv,red} = \nu \sin(\theta)\mathbf{K}_{conv,v_1} + \nu \cos(\theta)\mathbf{K}_{conv,v_2}$$

Where the decomposition has been carried out as follows:

$\mathbf{K}_{conv,red}$:

$$\sum_{i=1}^{M_{ND}} \Big(\sum_{k=1}^{4} w_{1,k}\mathbf{v} \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \Big)|\mathbf{J}^{(e)}| =$$

$$\sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k}\nu\Big(\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)\Big) \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \right)|\mathbf{J}^{(e)}| =$$

$$\sum_{i=1}^{M_{ND}} \Big(\sum_{k=1}^{4} w_{1,k}\nu\mathbf{v}_1 \sin(\theta) \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \Big)|\mathbf{J}^{(e)}| + \sum_{i=1}^{M} \Big(\sum_{k=1}^{4} w_{1,k}\nu\mathbf{v}_2 \cos(\theta) \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \Big)|\mathbf{J}^{(e)}| =$$

$$\sum_{i=1}^{M_{ND}} \nu \sin(\theta)\Big(\sum_{k=1}^{4} w_{1,k}\mathbf{v}_1 \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \Big)|\mathbf{J}^{(e)}| + \sum_{i=1}^{M} \nu \cos(\theta)\Big(\sum_{k=1}^{4} w_{1,k}\mathbf{v}_2 \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j \Big)|\mathbf{J}^{(e)}| =$$

$$\nu \sin(\theta)\mathbf{K}_{conv,v_1} + \nu \cos(\theta)\mathbf{K}_{conv,v_2}$$

The SUPG matrix can also be separated quite neatly, although given that the terms include squared velocities, the decomposition is slightly more intricate. It results in the following conclusion:

$$\mathbf{K}_{supg,red} = \nu^2 \sin^2(\theta)\mathbf{K}_{supg,v_1} + \nu^2 \cos^2(\theta)\mathbf{K}_{supg,v_2} + \nu^2 \sin(\theta)\cos(\theta)(\mathbf{K}_{supg,v_1v_2} + \mathbf{K}_{supg,v_2v_1})$$

This is readily shown by substituting the velocity vector by its separated expression and expanding the resulting terms. To make notation compact, the following equivalence for notation will be assumed in the following proof: $\tilde{\mathbf{v}} = \mathbf{v} \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j$, $\tilde{\mathbf{v}_1} = \mathbf{v}_1 \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j$ and $\tilde{\mathbf{v}_2} = \mathbf{v}_2 \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j$. Referring to equation (2.5), the decomposition of $\tilde{\mathbf{v}}$ yields $\tilde{\mathbf{v}} = \nu \sin(\theta)\tilde{\mathbf{v}_1} + \nu \cos(\theta)\tilde{\mathbf{v}_2}$. We insert this decomposition into the original expression of $\mathbf{K}_{supg}$ and separate the terms accordingly:

$\mathbf{K}_{supg,red}$:

$$\sum_{i=1}^{M_{ND}} \Big(\sum_{k=1}^{4} w_{1,k}\tau_1\tilde{\mathbf{v}} \otimes \tilde{\mathbf{v}} \Big)|\mathbf{J}^{(e)}| =$$

$$\sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k}\tau_1\Big(\nu \sin(\theta)\tilde{\mathbf{v}_1} + \nu \cos(\theta)\tilde{\mathbf{v}_2}\Big) \otimes \Big(\nu \sin(\theta)\tilde{\mathbf{v}_1} + \nu \cos(\theta)\tilde{\mathbf{v}_2}\Big) \right)|\mathbf{J}^{(e)}|$$

$$\sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k}\tau_1\Big(\nu^2(\sin^2(\theta)\tilde{\mathbf{v}_1} \otimes \tilde{\mathbf{v}_1} + \sin(\theta)\cos(\theta)(\tilde{\mathbf{v}_1} \otimes \tilde{\mathbf{v}_2} + \tilde{\mathbf{v}_2} \otimes \tilde{\mathbf{v}_1}) + \cos^2(\theta)\tilde{\mathbf{v}_2} \otimes \tilde{\mathbf{v}_2})\Big) \right)|\mathbf{J}^{(e)}| =$$

$$\nu^2 \sin^2(\theta)\mathbf{K}_{supg,v_1} + \nu^2 \cos^2(\theta)\mathbf{K}_{supg,v_2} + \nu^2 \sin(\theta)\cos(\theta)(\mathbf{K}_{supg,v_1v_2} + \mathbf{K}_{supg,v_2v_1})$$

Because we are working with tensor products of vectors, in the most general case $\tilde{\mathbf{v}_1} \otimes \tilde{\mathbf{v}_2} \neq \tilde{\mathbf{v}_2} \otimes \tilde{\mathbf{v}_1}$ so the two cross terms cannot be confounded.

$\mathbf{K}_{cw}$ includes the stabilizing terms corresponding to the crosswind stabilization. It has the most complex mathematical expression out of all the considered matrices, and its dependence on the velocity is such that it cannot be fully separated. This is because the velocity appears inside an absolute value operator multiplied by another vector, which results in an expression which is not separable at all with respect to the velocity angle (it is completely separable, however, with respect to the velocity module). Furthermore, the velocity also appears in the denominator of a fraction, making separation more complicated. Therefore, in order to proceed with the separation, we have relaxed the constraint of strong consistency on the crosswind diffusion, substituting the troublesome quantities instead with average values:

- For the fraction involving the residual in equation (3.17) (residual/concentration), the average value of quantity for the first 20 iterations of the Picard scheme has been taken as the mean value. If was checked that the average value of this term for a synthetic mesh for the 20th iteration was roughly a little over twice the value of the average fraction at the 1st iteration. Therefore, there is not a large difference in the order of magnitude of the average terms, which justifies the use of this approach as an initial approximation.
- The norm of the velocity squared was also computed for a synthetic mesh simulating an urban area for different angle combinations (notice that the velocity module has already been removed from the expressions). While here the maximum velocity was typically 7-8 times larger than the minimum one, if we take the percentile 1% and 99% of the velocities, we see the differences drop rapidly as the module of the percentile 99% velocity was only roughly twice that of the percentile 1%. For this reason, the extreme velocity module values were considered few enough to justify taking an average value for the velocity on the entire domain.

The problem with this approach is that is does not retain the strong consistency of the approximation. This is because the strong consistency depends on the norm of the residual appearing in the computation. If we substitute the actual residual by an average residual, it means that we are adding crosswind diffusion in the same proportion in our entire domain. This may lead to our solution being slightly overdiffused in some areas and still presenting some oscillations in others. The result of taking an average denominator will be that we will not be adding diffusion exactly in the crosswind direction at each point, but in a plane which will be a small angle away from the crosswind one. Nevertheless, the solution will still be more stable than the one with only SUPG given that diffusion will be added in the crosswind direction, so we have continued with this approach.

The separation is performed with the following aspects in mind:

-The property of matrix multiplication distributivity on both the left and right hand sides allows us to write, for any matrices $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ and $\mathbf{A}_4$ which allows for compatibility of the operations, that $\mathbf{A}_1 \cdot (\mathbf{A}_2 + \mathbf{A}_3) \cdot \mathbf{A}_4 = \mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \mathbf{A}_4 + \mathbf{A}_1 \cdot \mathbf{A}_3 \cdot \mathbf{A}_4$.

-The term $\frac{1}{2}\alpha_c^e h^e \frac{|\mathbf{v}\cdot\nabla\hat{N}_i \cdot c_i|}{|\nabla\hat{N}_i \cdot c_i|}$ will be renamed as $\tilde{r}$ for simplifying notation. The term $|(\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta))^2|$ will be renamed as $\tilde{m}$.

-The matrix $\sum_{i=1}^{M}(\sum_{k=1}^{4} w_{n,k}(\tilde{r}\nabla\hat{N}_i \cdot \mathbf{I} \cdot \nabla\hat{N}_j)|\mathbf{J}^{(e,i)}|)$ will be renamed $\tilde{\mathbf{I}}$ given that it does not include the velocity terms.

It follows:

$$\sum_{i=1}^{M_{ND}} \Big(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot (\mathbf{I} - \frac{\mathbf{v} \otimes \mathbf{v}}{|\mathbf{v}^2|}) \cdot \nabla \hat{N}_j \Big) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} + \sum_{i=1}^{M_{ND}} \Big(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot (-\frac{\mathbf{v} \otimes \mathbf{v}}{|\mathbf{v}^2|}) \cdot \nabla \hat{N}_j \Big) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} + \sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot \left(- \frac{\nu^2 (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)) \otimes (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta))}{|\nu^2 (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta))^2|} \right) \cdot \nabla \hat{N}_j \right) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} + \sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot \left(- \frac{\nu^2 (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)) \otimes (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta))}{\nu^2 \tilde{m}} \right) \cdot \nabla \hat{N}_j \right) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} + \sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot \left(- \frac{(\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)) \otimes (\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)}{\tilde{m}} \right) \cdot \nabla \hat{N}_j \right) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} + \sum_{i=1}^{M_{ND}} \left(\sum_{k=1}^{4} w_{1,k} \tilde{r} \nabla \hat{N}_i \cdot \left(- \frac{(\sin^2(\theta) \mathbf{v}_1 \otimes \mathbf{v}_1 + \cos^2(\theta) \mathbf{v}_2 \otimes \mathbf{v}_2 + \sin(\theta)\cos(\theta)(\mathbf{v}_1 \otimes \mathbf{v}_2 + \mathbf{v}_2 \otimes \mathbf{v}_1))}{\tilde{m}} \right) \cdot \nabla \hat{N}_j \right) |\mathbf{J}^{(e,i)}| =$$

$$\tilde{\mathbf{I}} - \sin^2(\theta) K_{cw,v_1} - \cos^2(\theta) K_{cw,v_2} - \sin(\theta)\cos(\theta)(K_{cw,v_1 v_2} + K_{cw,v_2 v_1})$$

4.4.2 Practical implementation issues

The codes used to perform the PGD were authored by members of the LAboratori de CAlcul Numèric (LACAN) from the Universitat Politècnica de Catalunya (UPC). These codes are publicly available and can be found in the LACAN github at the following link (as of 14/06/2019): https://git.lacan.upc.edu/zlotnik/algebraicPGDtools. The exact algorithm used by the code is presented in [7].

The algebraic PGD approach is divided into several subroutines which work together as a whole.

The first important subroutine, named pgdLinearSolve, is used for computing the rank-one approximations to the problem. The script requires two compulsory structures as inputs: we need to provide a separated left hand side and a separated right hand side which the algorithm then uses to parametrically solve the system of equations. Each of the structures consists of two compulsory sub-structures: first, we have to provide the different matrices (in the left hand side) or vectors (in the right hand side) which are then added to reach the solution of our problem. Second, we also have to provide the explicit parametric dependence of the matrices. The algorithm then proceeds on to compute the decomposition of the solution.

The exact inputs to this model were obtained from the separation of the matrices shown above. Having decomposed all of the matrices in the approximation, we regroup the terms as a function of their dependency. The matrices and dependencies were those that are shown in Table 4.1.

Another parameter is compulsory is the maximum number of terms in the approximation. The algorithm computes as many terms as those needed to reach the desired tolerance. However, if after having computed the maximum number of terms that tolerance hasn't been reached, the algorithm will stop and return the separation up to that mode. All the other parameters which are necessary for the fixed point algorithm and the

Matrix	Angle dependency	Velocity module dependency
$K_{diff} + K_r + \tilde{I}$	none	none
K_{conv,v_1}	$\sin(\theta)$	ν
K_{conv,v_2}	$\cos(\theta)$	ν
K_{supg,v_1}	$\sin^2(\theta)$	ν^2
K_{supg,v_2}	$\cos^2(\theta)$	ν^2
$K_{supg,v_1v_2} + K_{supg,v_2v_1}$	$\sin(\theta)\cos(\theta)$	ν^2
K_{cw,v_1}	$-\sin^2(\theta)$	none
K_{cw,v_2}	$-\cos^2(\theta)$	none
$K_{cw,v_1v_2} + \theta K_{cw,v_2v_1}$	$-sin(\theta)\cos(\theta)$	none

Table 4.1: Table with inputs to the PGD model

successive enrichment to be computed (tolerances, etc.) are optional, as default values are set, but the user can tune them to suit the particular needs of the problem.

Ideally, we want as many modes as possible so that our solution gets closer to the one that is reached with the FEM. Nevertheless, the more modes the approximation computes, the longer it takes to compute the subsequent mode. This is because to compute term n of the approximation, we need to compile all the information from the previous $n-1$ modes. This operation becomes longer the more modes we accumulate, until there comes a point at which is becomes unfeasible to compute the subsequent mode due to various reasons, including:

- For meshes which are simple enough to be computed by standard computers without crashing, computations with very large numbers of nodes can take a long time to converge, especially if the tolerance is set to be very small. Small instabilities may remain even after stabilization (full stabilization would overdamp the oscillations and lead to overdiffused solutions) and they may be captured by modes with very low mode amplitudes, which can lead the algorithm not to easily converge and thus make the computation of the solution overly costly.
- For very fine meshes, such as the one that will be seen in the following section, the separation has to be performed on a computing cluster due to the sheer size of the problem, which typically needs more working memory than that available with regular commercial PCs. If the matrices are very large, it is then necessary to resort to parallel computing to run the routines. While aspects of high performance computing (HPC), parallelization and job scheduling are not the object of this thesis, it should be noted that computing clusters, especially those with parallel threads, usually only let jobs run for a limited time before interrupting them to avoid jobs that run for innecessary long times and take up too many resources. In the Clonetroop cluster, a HPC cluster with 56 nodes belonging to LACAN, where this code was run, the limit for jobs to run is 24 hours, during which typically no more than 30 modes can be computed for the mesh that will be presented in the following section. If the separation job is interrupted, all results are lost, so it is only possible to compute a very limited number of modes before the job is terminated.

A solution exists to overcome these two problems: tensor compression. If we have computed n modes of our solution, we can choose a number d such that $d < n$ and compress the n modes into d terms yielding a generic tensor S_d. This approximation can

then be fed into the pgdLinearSolve algorithm which then goes on to compute the subsequent approximations taking S_d as an initial approximation. The term which would have been at the $n+1$ position in our approximation is now in $d+1$, yielding an obvious computational advantage with respect to the uncompressed version. Therefore, the second important routine is the pgdTensorCompression, which allows us to compress the output of the pgdLinearSolve with n terms into a similar approximation with d terms. Another benefit of tensor compression is that eliminates redundancies in the modes. When computing the successive modes, orthogonality of the modes is not imposed [23]. Because the compression is doing by projecting the tensor onto an orthonormal basis, these redundancies are more effectively dealt with in the compressed version of the tensor.

Finally, once the decomposition algorithm has converged, the solution is available in its parametric dependence. To visualize a certain solution on our mesh, we need to reconstruct the solution for a given set of parameters. This means that we need to sum the specific outputs of the pgdLinearSolve routine for our target parameters. A routine called fullTensor allows us to do just this, inputting the index of the parameters that we wish to particularize our solution for and outputting the corresponding solution.

Chapter 5

Case study: Pollution transport in Barcelona

Solving pollution transport equations on fictitious idealized domains has a limited interest. It is useful to understand the qualitative behaviour of the numerical method under study. However, once the method has been validated, the final objective is to apply it to a real situation to obtain results which can be used to improve decision-making in our urban areas.

The most interesting property of the FEM is that all the mathematical manipulations can be performed without ever actually specifying the exact coordinates of the nodes, the number of nodes or the elements resulting from the nodes. Therefore, while the results obtained at the end of the previous sections may have been obtained with a generic mesh in mind, the derived equations, boundary conditions and parameters are just as applicable to a synthetic mesh as to a mesh actually reproducing the physical world. This means that the equations we are going to solve are exactly the same. The only varying factor will be the mesh, which needs special treatment due to two reasons:

1. The mesh is no longer generated by the user, but is given to him as an input. This severely limits mesh refining, as the user does not necessarily have the tools to subsequently remesh the domain while remaining faithful to the original problem geometry if the node structure is modified. Therefore, the mesh is what it is and cannot be refined to suit the user's needs.
2. Meshes of real urban areas typically capture large areas of land, resulting in large numbers of nodes and elements if one wants to meaningfully capture the most relevant urban elements. This results in meshes which require a special treatment.

Luckily, recent years have brought by advances in mesh generation from existing data. For cities, meshes can be generated from visual information such as satellite images. This means that a mesh can mostly be generated without much human intervention, which would otherwise be a long and tedious process, and probably not as accurate as one computed automatically.

One of such meshes was available for our study. It covered a roughly 80 km^2 area including the entire city of Barcelona and some of the surrounding areas. The original mesh was generated and handed over by our colleagues at the Barcelona Supercomputing Center (BSC).

The district of the Eixample was chosen for the simulation due to its nice geometric properties, with the square blocks being relatively similar to the blocks generated synthetically in the previous section. This selection was done because carrying the simulation in the entire city needed far more computational power than was available for this thesis. In fact, the complete mesh had more than 80 million nodes and more than 320 million elements, making so large that it could not be processed with commercial tools nor with other lower-level methods in a small cluster. Instead, it had to be pre-processed using a number of tools. A small part of the total mesh was selected and after the pre-processing of the mesh, the results of the potential flow and advection-diffusion problems were computed with MATLAB.

5.1 Mesh manipulation

The first step was to pre-process the mesh. To do this, a MATLAB script was created to read the file (which was received in a .geo file which could not be understood by any pre-processing software), choose a certain subset of elements within the mesh, and create a file with the new sub-mesh in the .inp format, which can then be fed into ABAQUS. Node and element selection was carried out because a commercial computer would have been unable to visualize the entire mesh. To ensure connectivity, the chosen elements were those with barycenters inside a given rectangle. The mesh could then be visualized and used to solve the equations. The shear size of the mesh meant that typically no more than 2% of the nodes and elements in the mesh could be visualized at any given time without the software crashing. The resulting mesh can be seen in Figure 5.1, and corresponds to a mesh of roughly 70.000 nodes and and 360.000 elements.

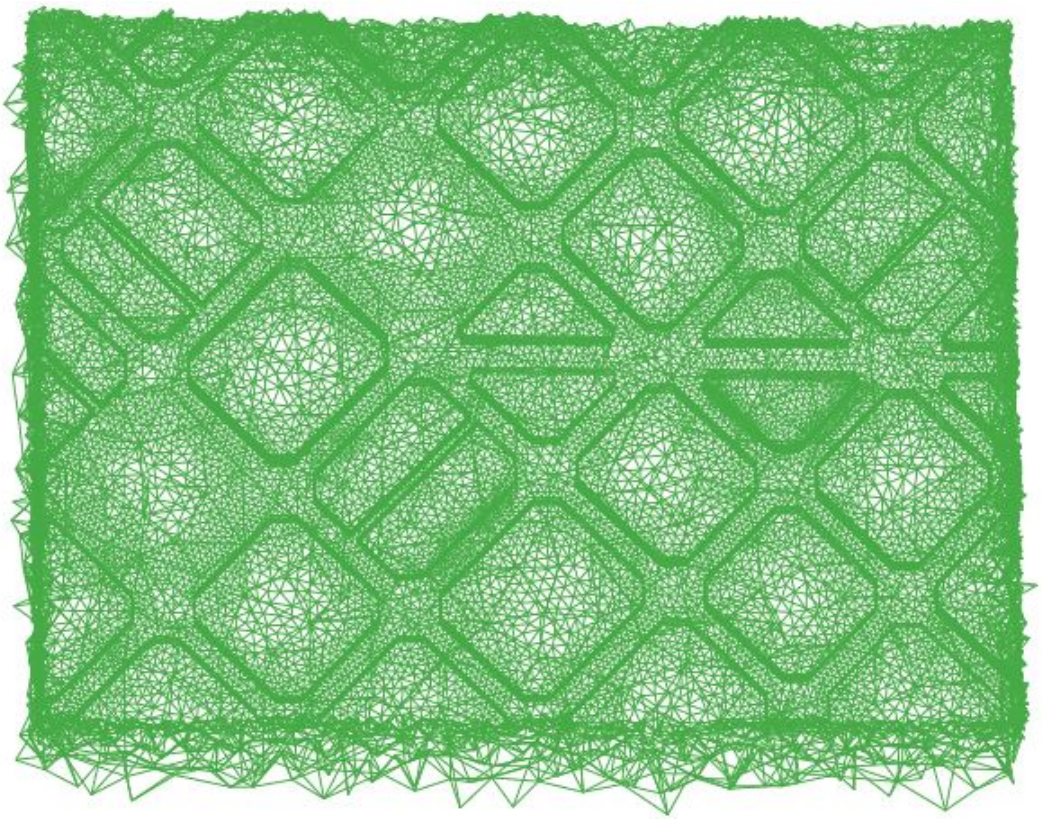

Figure 5.1: Mesh of some streets in the Eixample in Barcelona. The mesh is seen from below.

To work with the resulting mesh, ABAQUS was the commercial software of choice. ABAQUS not only lets the user visualize a mesh generated from a text file input, but also allows the user to subsequently perform certain operations the mesh. While modifying the underlying structure of the mesh is not doable with this software, the idea was to be able to perform certain conditions which would allow us to determine the inputs of equations (2.4) and (3.2).

The second step was to select the nodes corresponding to the boundary conditions of these equations (except for those that appear as zero Neumann boundary conditions for both problems). Given that these nodes need to be operated with (Robin and non-zero Neumann terms appear in the weak form, and Dirichlet nodes are used to carry out matrix reduction), it is necessary to flag them. While this is usually trivial in a self-generated mesh, where the user has complete control over the geometry and discretization, when a mesh is manipulated and its original boundaries are deleted, it is necessary to specify what nodes are now on the boundary of the domain. Notice that the resulting mesh from the previous operation has very irregular boundaries which lack flat faces.

The resulting flagged nodes where Dirichlet and Robin types of boundary conditions are applied can be seen in Figure 5.2. Zero Neumann boundary conditions are assumed to hold in all the other boundaries.

Dirichlet boundary conditions are zero on the laterals and a on the small area in the middle of the domain, where a is the non-zero Dirichlet boundary condition defined in (3.2). It should be noticed that in this particular version of the problem statement, the non-zero Dirichlet boundary condition is no longer being applied to the entire bottom surface, but to a subset of nodes corresponding roughly to a park which is located in l'Eixample. The pollution being emitted from this area could correspond, for instance, to bushes which have caught fire. Robin boundary conditions correspond to the buildings. There are two areas in which Robin boundary conditions have not been applied: these correspond to two parks in the Eixample, which are not assumed to absorb pollution.

Once the appropriate boundaries for the different types of boundary conditions were chosen, the resulting mesh was exported to use it for solving the advection-diffusion problem using the FEM and PGD codes. However, the ABAQUS writing format, .inp, is not readily recognizable by MATLAB. Therefore, further processing of the text file was needed. For this purpose, another MATLAB routine was created which converted text files from ABAQUS format to MATLAB-readable matrices.

5.2 Modified problem statement

As seen in Figure 5.2, the nodes at which we are applying the Dirichlet boundary conditions have changed from those considered in equation (3.2), where we were considering the entire bottom of the model to have a non-zero Dirichlet value. For this reason, we will restate the problem statement with the new boundary conditions.

Following the notation in the previous chapters, our boundaries will be decomposed as $\partial\Omega = \partial\Omega_{bot} \cup \partial\Omega_{top} \cup \partial\Omega_{lat,in} \cup \partial\Omega_{lat,out} \cup \partial\Omega_{build}$. In this problem, the bottom area has been further decomposed into two parts: an area of the park (noted $\partial\Omega_{bot,park}$) and the rest of the bottom (noted $\partial\Omega_{bot,nopark}$). Therefore, in our new configuration, $\partial\Omega = \partial\Omega_{bot,park} \cup \partial\Omega_{bot,nopark} \cup \partial\Omega_{top} \cup \partial\Omega_{lat,in} \cup \partial\Omega_{lat,out} \cup \partial\Omega_{build}$. We now revisit equation (3.2) and apply the boundary conditions to their corresponding boundaries for this new problem:

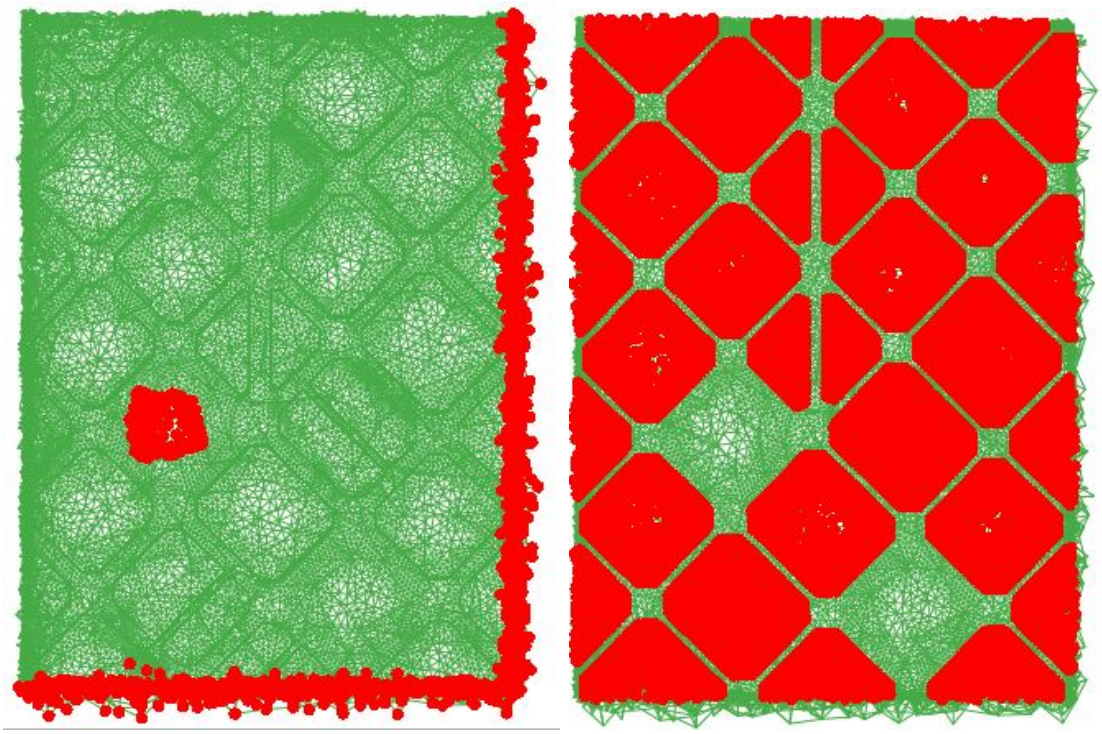

Figure 5.2: Nodes with Dirichlet boundary conditions (left) and Robin boundary conditions (right).

$$\begin{cases} 0 = -D\Delta c + \mathbf{v} \cdot \nabla c & in\ \Omega \\ c = a & on\ \partial\Omega_{bot,park} \\ c = 0 & on\ \partial\Omega_{lat,in} \\ D\frac{\partial c}{\partial \mathbf{n}} = 0 & on\ \partial\Omega_{lat,out} \cup \partial\Omega_{top} \cup \partial\Omega_{bot,nopark} \\ D\frac{\partial c}{\partial \mathbf{n}} + \beta c = 0 & on\ \partial\Omega_{build} \end{cases} \tag{5.1}$$

Because only the boundaries where we are applying the boundary conditions have changed, but the conditions remain essentially the same, it is not necessary to repeat the entire process of going from the strong to the fully discretized form of the equation. All the steps previously derived, including the demonstration of the well-posedness, apply to this modified problem. The only change that needs to be made is in the matrix reduction procedure carried out in the final step, where the columns and rows being extracted from the stiffness matrix K will correspond to those of the nodes with applied non-zero Dirichlet boundary conditions in this new formulation.

Chapter 6

Results

To compute the results, concrete values for the variables a and β as they appear in equations (3.2) and (5.1) have been provided. The final choice has been to set $a = 8000$ and $\beta = 10^{-5}$.

6.1 Quality control

Because all the code for the FEM was developed and implemented from scratch in MATLAB, a few basic tests have been carried out to ensure that no programming mistakes have been made. Given that we will be comparing our PGD solution to the one obtained with the FEM, it is important to ensure that the FEM solution is accurate and behaves according to expectations.

For this, we have chosen to study the formulation of our problem without added stabilization to determine if the numerical instabilities begin when we expect them to. This serves as a quality check of the advection-diffusion code without stabilization. To do this, we have studied the evolution of the minimum concentration in the domain as a function of the Péclet number. The minimum concentration is a reliable indicator of the oscillations, given that it will start taking negative values as soon as the spurious behaviour begins. We obtain the values in Figure 6.1:

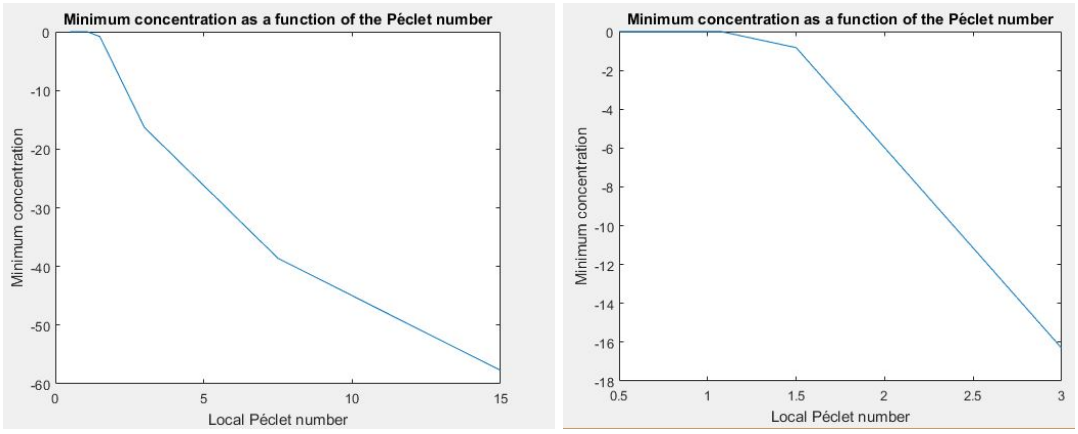

Figure 6.1: Evolution of the minimum concentration with respect to the Peclet number (general at the left and zoom-in at the right)

We can see that the solution behaves like we expect it to: the concentration is non-

negative until a Péclet number of 1 is reached (see Figure 6.2). For Péclet numbers greater than one, the oscillations begin and the solution starts showing negative values for the concentration, thereby signaling that the solution is no longer stable. Furthermore, the higher the Péclet number, the stronger the oscillations become as highlighted by the decrease in the the minimum value, something which was expected.

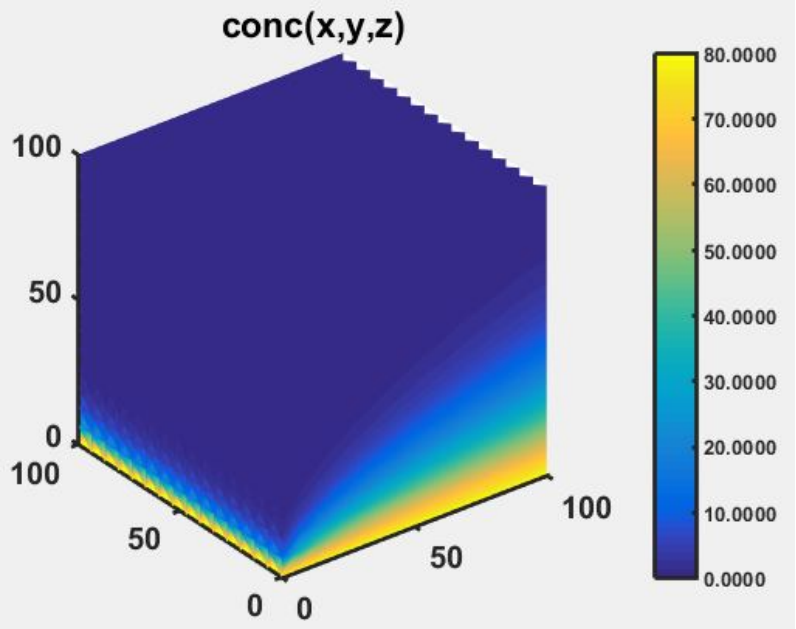

Figure 6.2: Solution for the limit case of Pe=1, just before the solution begins oscillating

For higher Péclet numbers, which will be the ones in most air pollution transport problems in urban areas, we need to add the SUPG stabilization terms to stabilize the solution. Doing this will bring the concentration values back up to positive numbers and the solution will display a much more physical behaviour.

We therefore increase the velocity values and monitor the evolution of the minimum concentration for different Péclet numbers under this new stabilized formulation of the problem to see at which point additional diffusion will be needed.

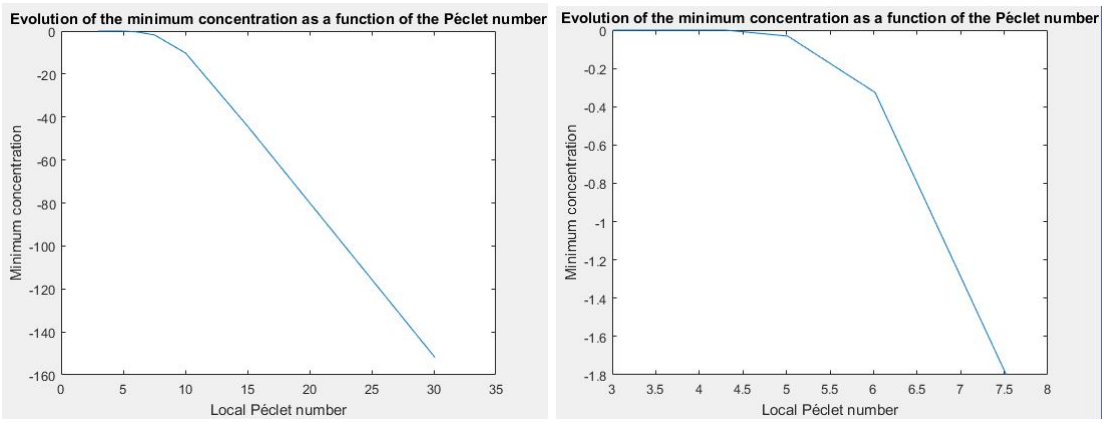

Figure 6.3: Evolution of the minimum concentration for the SUPG-stabilized formulation of the problem as a function of the Péclet number (zoom-in in the right)

As shown in Figure 6.3, the quality of the solution improves for Péclet numbers higher than 1. By adding SUPG stabilization, the concentration takes positive values again until a Péclet number of about 5 is reached, proving that the implementation of the SUPG

stabilization has been successful. However, as the Péclet number keeps increasing beyond 5 for this test case, SUPG is no longer sufficient to fully stabilize the solution as the scheme will begin over and undershooting in the areas with the stronger concentration gradients.

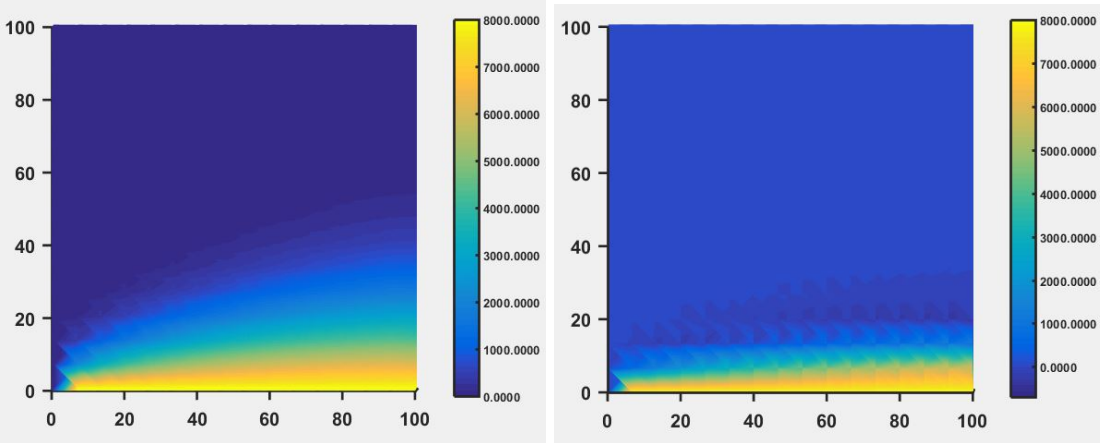

Figure 6.4: Lateral view of the domain with concentration visualization for Pe=5 (left), fully stabilized, and Pe = 50 (right), with instabilities in sharp gradient areas

Figure 6.4 displays a lateral side of our synthetic urban domain with wind blowing from left to right. We can see, on the left, a fully stabilized solution for a Péclet number of 5, where the solution displays a normal behaviour. However, on the picture of the right, where the Péclet number is one order of magnitude higher, oscillations (marked by values below 0, seen in dark blue) begin developing. At this point, crosswind stabilization has to be introduced to fully stabilize the solution.

6.2 Crosswind stabilization parameters tuning

Crosswind stabilization includes two empirical parameters, C and the tolerance of the iterative scheme. Therefore, these two parameters have to be chosen appropriately.

We therefore want to study the relationship between the two main problem parameters, wind speed and wind angle, with the two crosswind stabilization numerical parameters determined by the user. For this, we need to carry out a sensitivity analysis to understand the relationship between the parameters and to see what factors may have an impact on the others. This has been done by fixing three of the parameters that we are studying, making the last one vary, and observing how the stability of the scheme, as determined by the minimum concentration, evolves.

However, although the minimum concentration serves as a good proxy to understand when the solution is stable, it is not informative about the physical soundness of the approximation. This is because crosswind stabilization is increasing diffusion in the crosswind direction, and if too much is added, the behaviour of the solution will begin to diverge from the one that is expected. As such, another parameter has been measured to gauge the physical soundness of the solution, namely the Euclidean norm of the solution vector. If two solutions contain only positive values for the concentration, the one with the lower norm will be preferred. A lower norm of the solution vector means that, overall, there are fewer nodes with non-zero concentration values, and that the solution is less likely to be affected by excess diffusion.

Furthermore, to study this dependence the boundary conditions of the problem have been slightly modified to make the problem less symmetric and have a clearer idea of the impact of the angle. This is why the modified version of the original problem was introduced in the problem statement, and can be seen in Figure 3.4. If the impact of the angle is studied on a problem where all the bottom boundary is emitting pollution and the geometry is symmetrical, the effect of the angle on the solution is underestimated. Instead of imposing a Dirichlet boundary condition on the entire bottom, only two rows of nodes in the bottom boundary simulating a street have now a fixed concentration. No boundary condition has been specified on the rest of the bottom, thereby applying a natural Neumann boundary condition which imposes non-penetrability. Therefore, $\partial\Omega_D = \Omega_{bottom,street}$ and $\partial\Omega_N = \Omega_{top} + \Omega_{lat,out} + \Omega_{bottom,nostreet}$. This does not affect at all the discretized form (3.17) as the only thing that changes is the matrix reduction procedure carried right before the solution, where fewer rows are subtracted from the stiffness matrix. Figure 6.5 shows the bottom side of this new domain where the Dirichlet boundary condition is only applied to a restricted part of the bottom boundary. This corresponds to a 3D rendering of Figure 3.4 as seen from below.

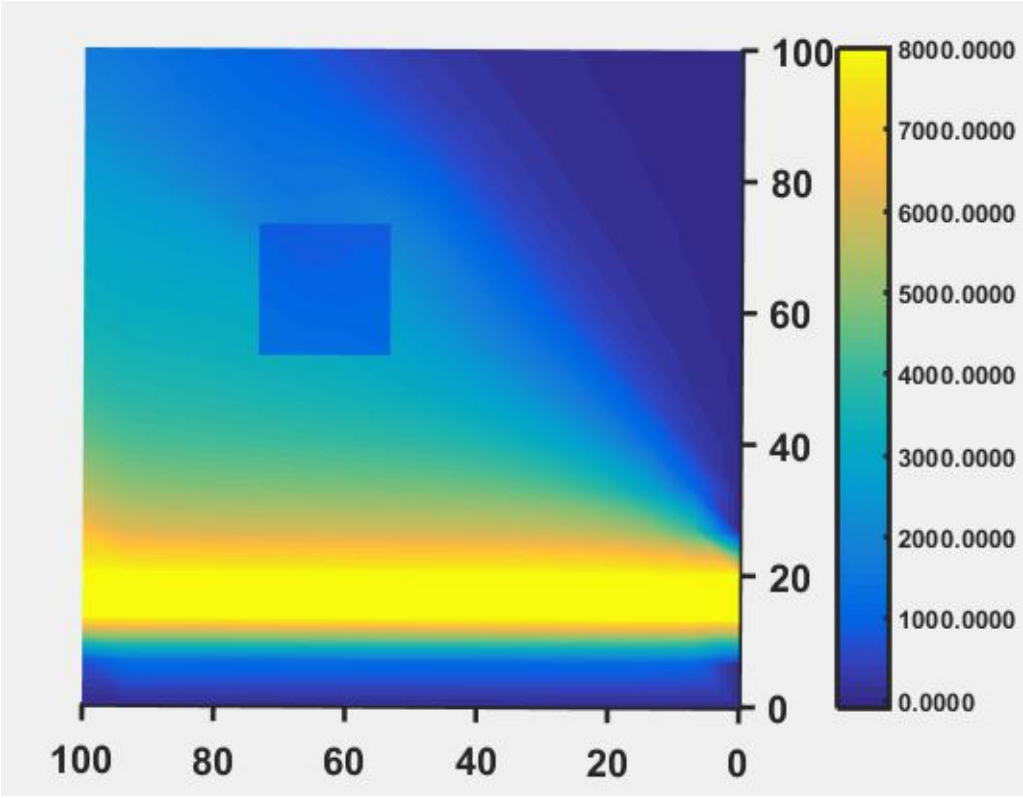

Figure 6.5: Solution of the advection-diffusion problem for an alternative synthetic domain, as seen from below. The Dirichlet boundary condition (in yellow) is applied only on some bottom nodes instead of the entire bottom side.

The sensitivity analysis has been carried out for the four parameters (wind angle, wind velocity, crosswind added diffusion parameter and crosswind iteration tolerance). Unless the parameter itself was affected, the default values have been taken as follows:

- Velocity angle: $\frac{\pi}{2}$
- Velocity module: such that the average local Péclet number is equal to 7500.
- Crosswind added diffusion parameter C: 0.05

• Tolerance for the crosswind iterative scheme: 0.002

Wind Angle

We keep all parameters fixed and evaluate the impact of the angle on the solution for 9 different angles in the range $[0, \frac{\pi}{2}]$. We then monitor the minimum concentration.

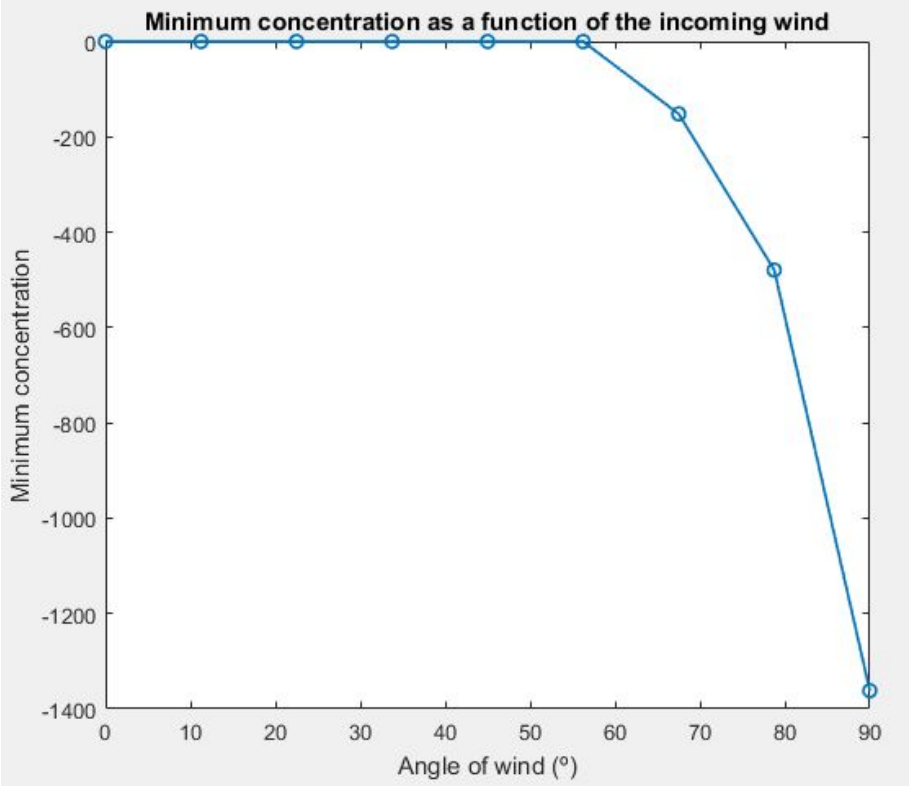

Figure 6.6: Minimum concentration as a function of the angle

As we can see in Figure 6.6, the angle does have an impact on the minimum concentration. An angle of $\frac{\pi}{2}$, which is the most unfavorable, corresponds to a velocity aligned with the pollutant emitting "street", resulting in a very thin and elongated cloud of pollution. This seems to suggest that, if the angle of the wind is such that the cloud of pollutant is very thin, crosswind diffusion may need to be increased.

Wind velocity module

We start by changing the velocity module so that we obtain the results for different Péclet numbers. 9 different Péclet numbers in the range from 1400 to 14.000 have been tested, and the results in Figure 6.7 have subsquently been obtained.

Although the graph seems to suggest that the minimum concentration sharply decreases when the Péclet number decreases, the scale along the y axis has to be noted. Therefore, for this range of Péclet numbers which varies by one order of magnitude, the velocity does not have an impact on the results. Therefore, the wind velocity seems to make a low impact on the stability of the solution once a stable configuration has been reached with the other parameters.

Crosswind added diffusion

The shock capturing technique also involves an empirical parameter C which appears in (3.18) and greatly influences the amount of crosswind diffusion that is added by deter-

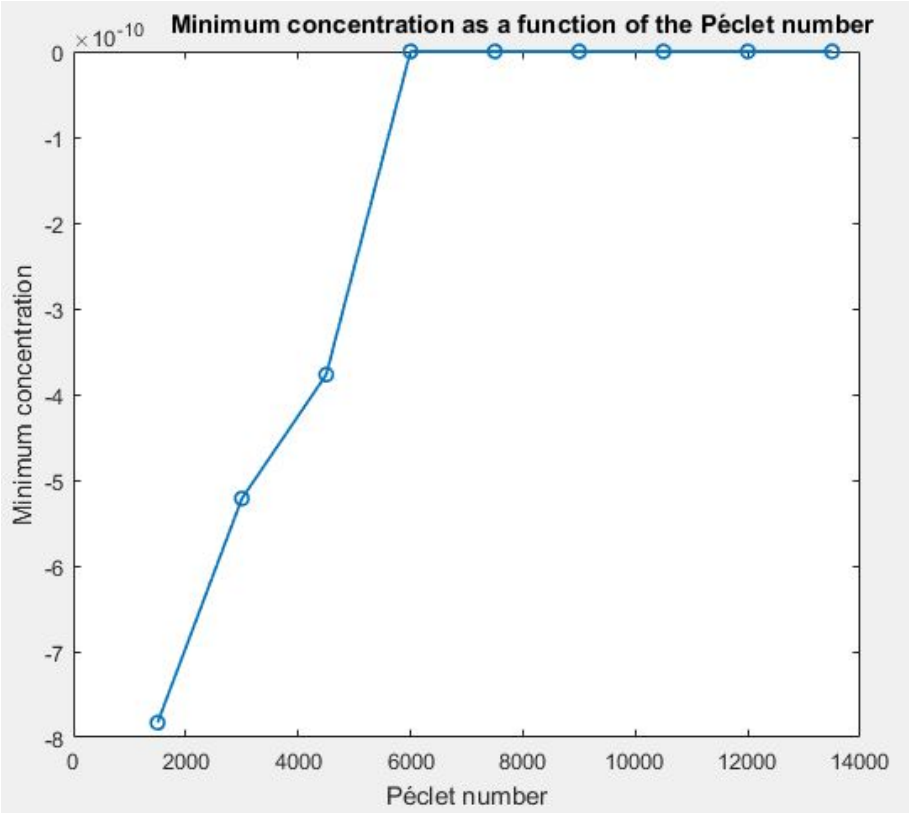

Figure 6.7: Minimum concentration as a function of the Péclet number

mining what elements need it. Therefore, we study its evolution.

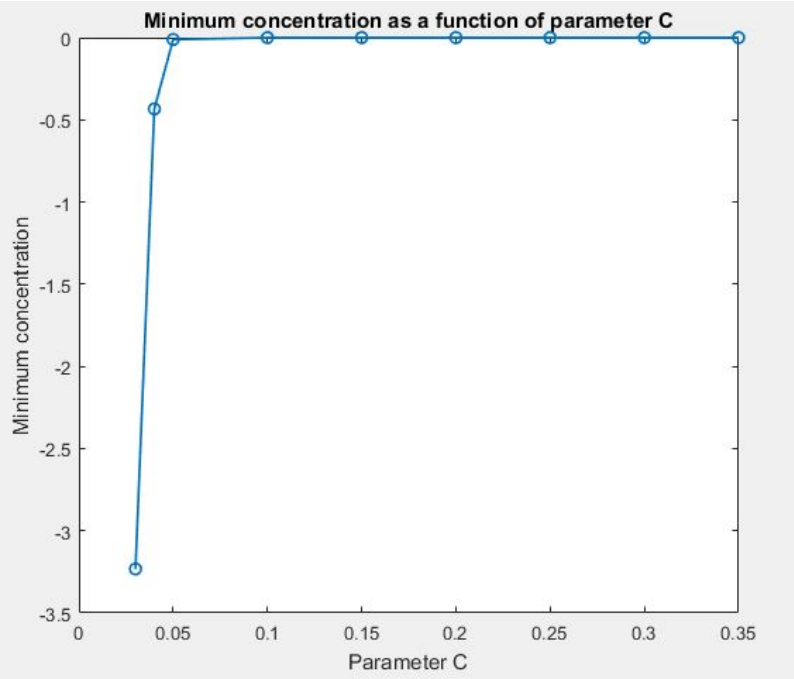

Figure 6.8: Minimum concentration as a function of the shock capturing parameter

In Figure 6.8, we see that for very low values of C, the stabilization is insufficient and the solution still displays negative values for a fixed tolerance. However, once a minimum value of C is reached, all of the concentrations are positive, which means that the solution

is stable. Given these results, we try to take a very high value of C to be on the safe side. However, as seen in Figure 6.9, this is not the best strategy. Once the solution has been stabilized, the more crosswind diffusion we add, the more overdiffused our solution becomes as evidenced by the increase in the norm of the concentration vector. Therefore, if we choose C to be very high and the tolerance to be relatively low, our solution will no longer be representative of the actual behaviour of the solution.

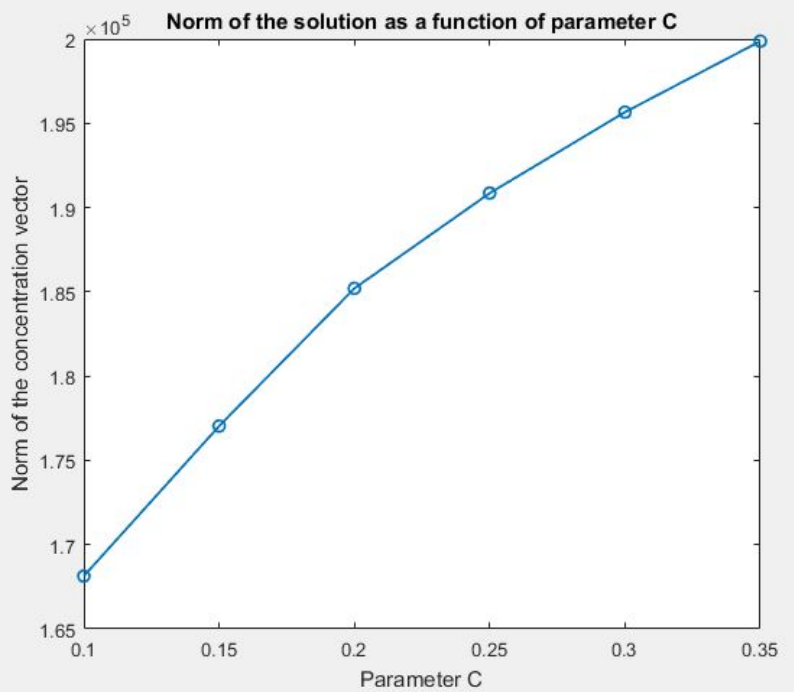

Figure 6.9: Norm of the concentration vector as a function of parameter C which controls the added crosswind diffusion

Figure 6.10 shows a comparison between two stabilized solutions and illustrates what overdiffused solutions look like. The wind is assumed to be flowing from the back of the paper towards the reader. The problem on the left hand side has been computed with a lower C than the one in the right hand side. We can therefore see that higher C values lead to an increase in diffusivity highlighted by the apparent rising of the pollution cloud - we are dealing with a Péclet number of 3500, which indicates a very strong advective effect, and the expected solution is therefore a cloud of pollutant which is very flat and stays close to the ground, much more similar to the term on the left.

Crosswind diffusion iterative tolerance

To compute the crosswind artificial diffusion term, an iterative procedure is carried out. Within this algorithm, a tolerance is specified to understand when iterations should stop. The determination of this tolerance impacts both the accuracy of the solution (smaller tolerances result in more stable, but also more diffused solutions for a given C) as well as the computation time (smaller tolerances result in more iterations so the procedure will take longer to finish).

As seen in Figure 6.11, lower tolerances improve the stabilization and result in positive concentrations in the entire domain. However, taking very low tolerances results in two problems. First, for very low tolerances, the algorithm will take many steps to converge, and the computational cost will therefore increase. Second, low tolerances usually increase the total diffusivity, resulting in less physical solutions. The increase in diffusivity which is brought about by the decrease in tolerance can be seen in Figure 6.12.

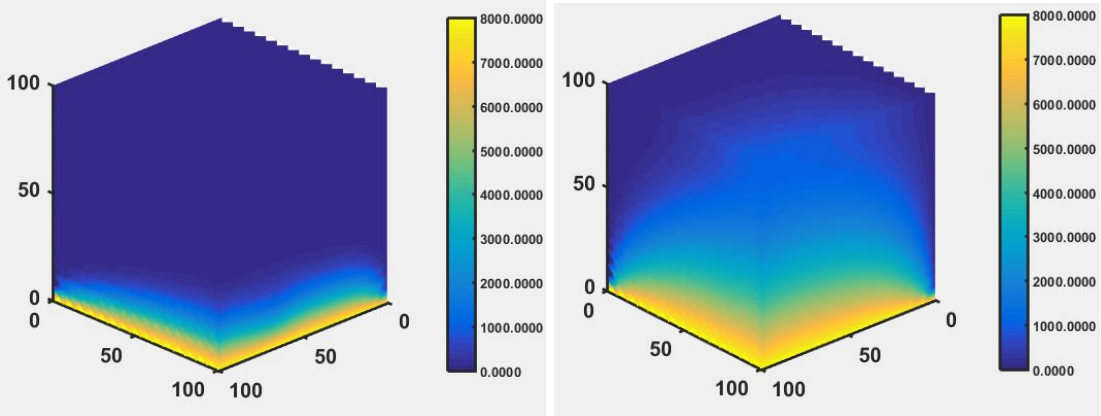

Figure 6.10: Fully stabilized solutions for values of C=0.05 (left) and C=0.5 (right). The rising cloud of pollution with such a high Péclet number indicates overdiffusion for C=0.5

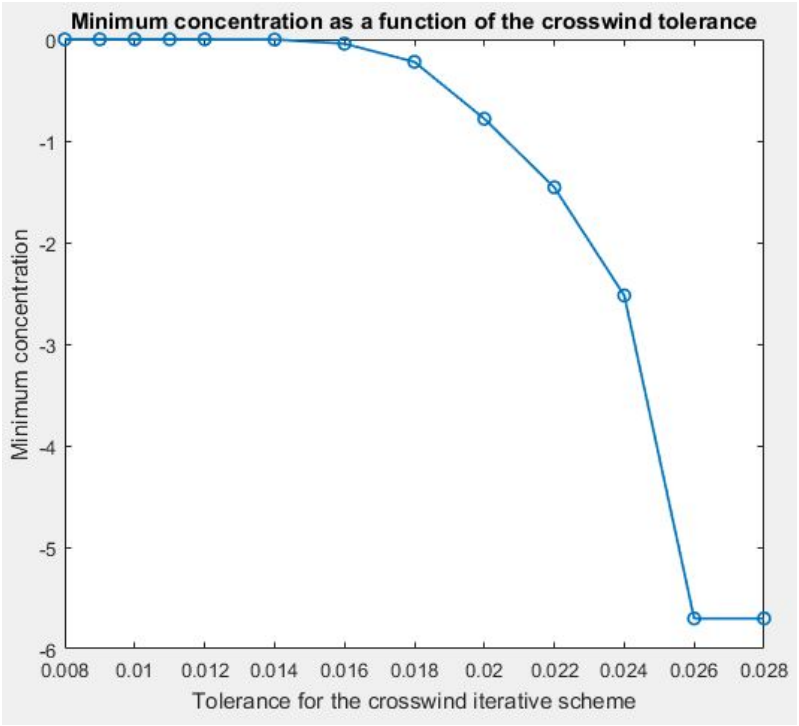

Figure 6.11: Minimum concentration as a function of the tolerance of the crosswind fixed point iteration for a fixed set of parameters

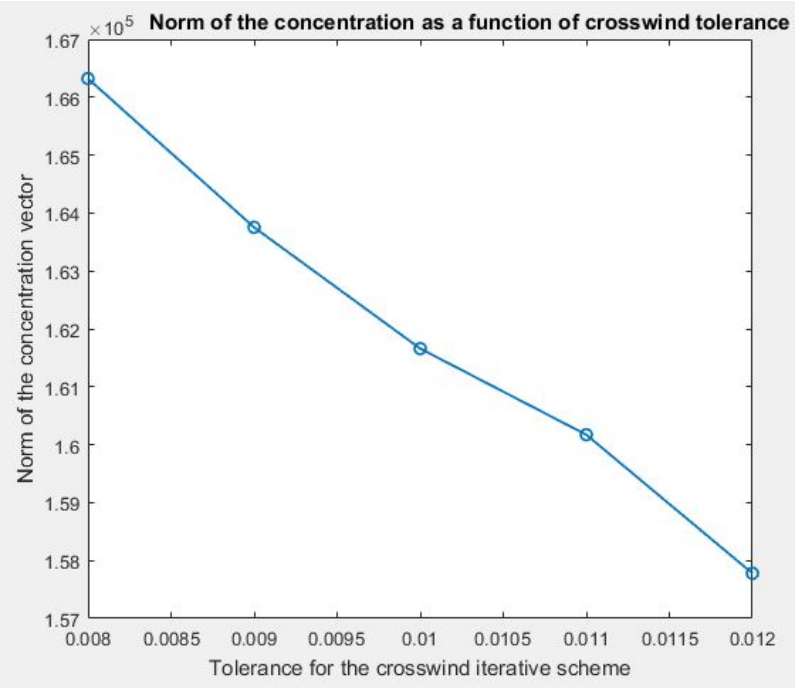

Figure 6.12: Norm of the solution vector for different tolerances. High norms are indicative of excess diffusion

6.2.1 Final parameter choice

It seems evident that the most important choice, rather than the choice of an individual parameter, is the choice of the combination of values for C and the tolerance of the crosswind iteration scheme. While stabilized solutions may be obtained for many different combinations of these two parameters, setting very high values for C and low values for the tolerance may result in stable but overdiffused solutions. Therefore, the idea is to keep the value of C as low as possible and set the tolerance high enough to ensure complete stability but low enough to prevent overdiffusion.

The final combination to be applied has finally chosen to be $C = 0.05$ with a tolerance of 0.012. This fully stabilizes the result and, although the solution obtained for the synthetic mesh is very slightly overdiffused, we can consider it an appropriate approximation of the actual solution.

It should be noted that the final goal of this thesis is to compare the PGD and FEM approaches. What will be important in the following section of the results will not be whether the solution is physically accurate or not, but whether the solution given by the PGD approximation and the one given by the FEM are similar. Of course, having a physically sound approximation is necessary for the final application of the PGD to actual problems in real situations, but at this stage, a slightly (not drastically) overdiffused solution is a reasonable price to pay in exchange for mitigation of the oscillations.

6.3 PGD Results

In this section, the results obtained with PGD are compared to those obtained with the FEM approximation. Three types of data will be monitored to evaluate the results:

- Coefficient β_n: As defined in equation (4.1), β_n measures the amplitude of mode n in the approximation. Therefore, both the absolute amplitude β_n and more frequently the relative amplitude $\frac{\beta_n}{\beta_1}$, is used to assess the convergence of PGD.
- Mean error of the PGD approximation: A very useful measure of the accuracy of the PGD approximation is the average error for each of the rank-n approximations. This means that, for each of the n computed modes, we reconstruct the solution until the n-th mode, and compute the error of the approximation with respect to the FEM solution for all the possible parameter combinations. This yields a "global average error" for each reconstruction, and allows us to check how each additional mode adds accuracy to the approximation.
- Qualitative information: Finally, different data visualizations can also allow us to assess whether the solution behaves according to what we would expect. Visual information can also serve as a diagnostic tool - if there is a problem, measures of accuracy and convergence may not suffice to explain the root cause of the issue. Visual data can help us understand why there are problems in our approximations.

6.3.1 Unstabilized PGD for low Péclet numbers

We begin by considering the PGD approximation to a low local Péclet configuration (Pe<1), which doesn't require stabilization, to consider how good the approximation is when no numerical instabilities need to be accounted for. For this, we consider a very small velocity. The following discretizations are introduced into the model for the parameters:

- Angle: from 0 to $\frac{\pi}{2}$ with 91 points in the discretization
- Velocity: from 10^{-6} to 10^{-4} m/s with 91 points in the discretization. This narrow range is taken to ensure that the Péclet number remains below 1, so that we don't need to use the SUPG matrix as an input to the model. All concentrations are strictly positive, so the minimum concentration is ensured to be 0 so that it is not tracked as part of the quality control. Resulting Péclets are in the [0.008, 0.8] range.

The solution for the low Péclet number was computed on the cubic mesh with boundary conditions corresponding to those of the problem pictured in Figures 3.4 and 6.5. This was done once again to make the problem less symmetrical and make sure that good results were not due to an excessive idealization of the domain.

The evolution of coefficient β lets us see that the approximation is converges towards the solution as we expect it to. As it can be seen in Figure 6.13, amplitudes show a clear downward trend and, for a tolerance of 10^{-4} for the relative mode amplitude $\frac{\beta_i}{\beta_1}$, convergence is reached after 68 modes.

As we can see in Figure 6.14, the PGD offers a very good approximation for problems with low Péclet numbers. The error (measured with the Euclidean norm of the vector of difference in concentrations with respect to the FEM) steadily decreases as we include more terms in the approximation and it is of around 2% with respect to the solution obtained with the regular FEM after the computation of 10 modes.

Given these good results concerning the separation, we can try modifying the discretization of the separated variables to include a wider range of velocities. However, if

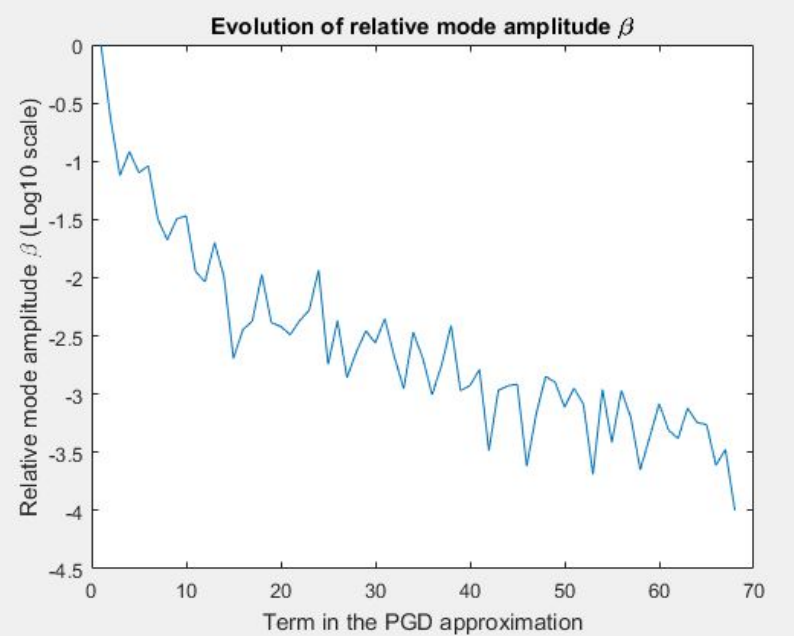

Figure 6.13: Evolution of the relative importance of coefficient $\frac{\beta^i}{\beta^1}$ for the ith iteration for low Péclet numbers (Pe<1), without stabilization

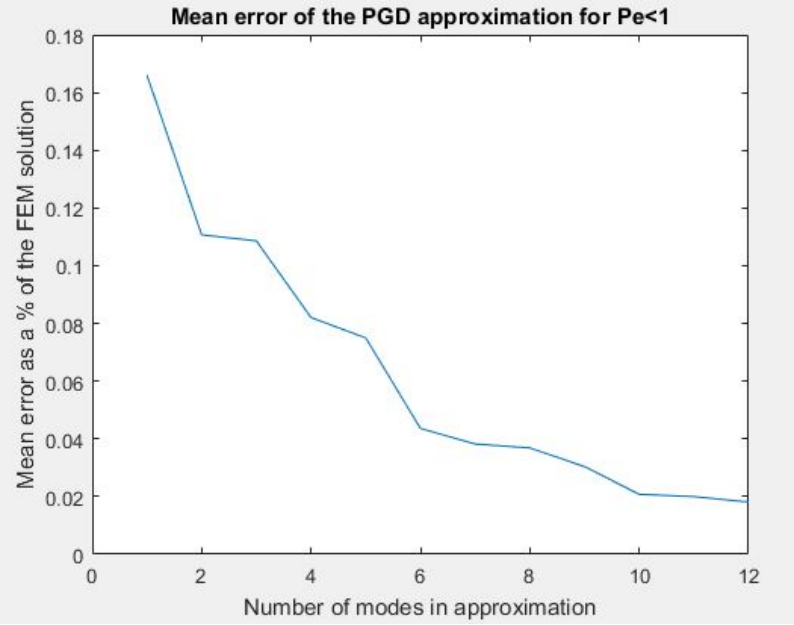

Figure 6.14: Evolution of the global error of the approximation as a function of the number of terms in the iteration

instead of using a velocity range that keeps the Péclet within low range, we input a range of velocities which brings about the instabilities in our solution, the results do not show a good convergence:

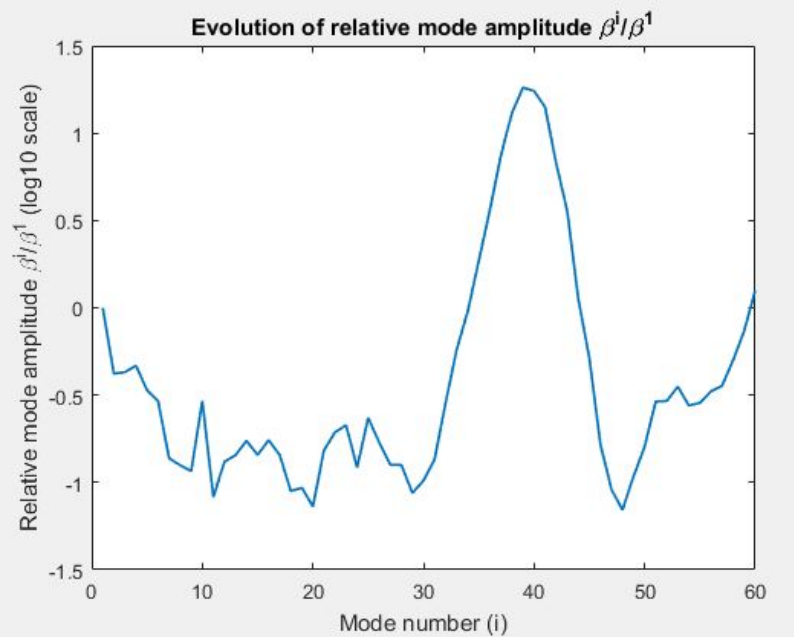

Figure 6.15: Evolution of the relative mode amplitude $\frac{\beta^i}{\beta^1}$ for Péclet $\in$ [1,100] range, without stabilization

As we can see in Figure 6.15, the plot no longer displays a clear descending trend. This means that the underlying separation of variables in the PGD approximation is not adequate for the problem that we are solving. Indeed, there is a value for i at which the relative importance of β_i begins meaning steadily, signaling that the solution is having convergence issues. This seems to be due to the fact that the oscillations are captured by a mode of lower amplitude of the PGD, which subsequently begins trying to separate the spurious oscillations, thus leading the solution to become less physical. Because oscillations cannot be characterized as functions of the parameters, the algorithm will display a non-convergent behaviour.

Therefore, the need to include more terms in the separation to stabilize the solution is justified.

6.3.2 SUPG stabilized formulation

In this section, given that we want to concentrate on issues regarding stability, we initially went back to our original problem formulation with the boundary conditions as seen in Figure 3.3. The evaluation of the global error for the SUPG approximation can be seen in figure 6.17.

We see that the relative mode amplitudes display a decreasing trend, although not as marked as the one for low Péclet numbers, but are not increasing rapidly as in Figure 6.15, which is a good sign and shows that the parametrization is appropriate for this problem.

It may be surprising to realize that the first mode in the PGD already captures more than 94% of the variation. Therefore, with only 1 mode, we are already capturing 94% of the solution for all the parametric combinations. However, this boils down to the modeling

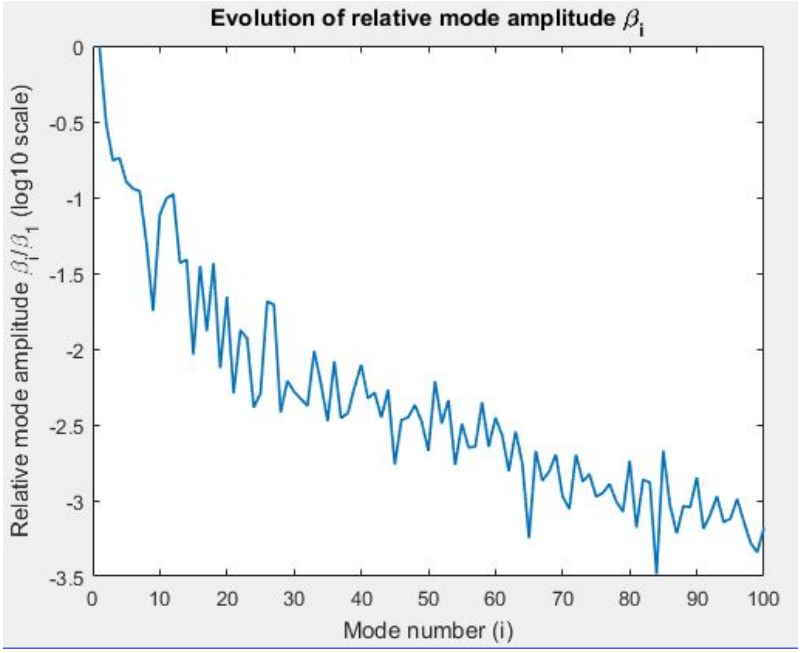

Figure 6.16: Evolution of the relative mode amplitude of the PGD solution for a problem with homogeneous Dirichlet on the bottom boundary

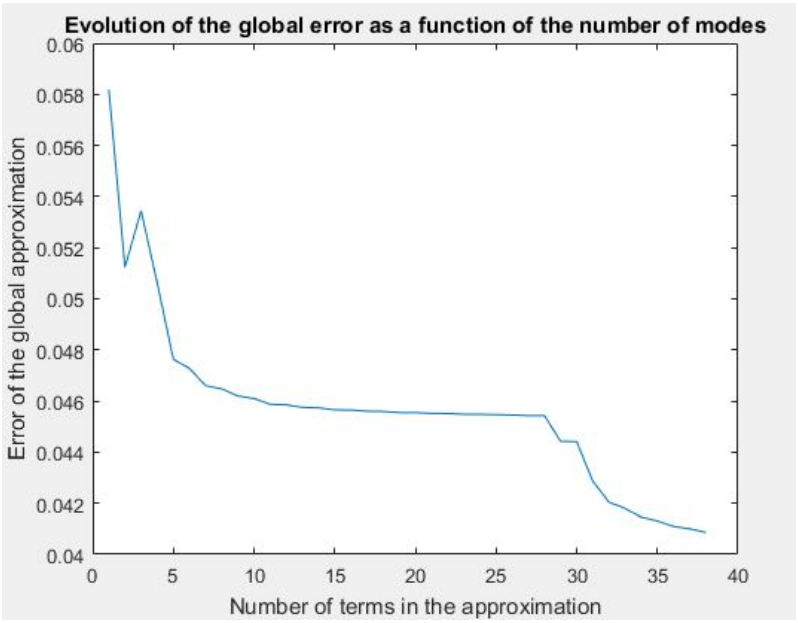

Figure 6.17: Evolution of the global error for a problem with homogeneous Dirichlet on the bottom boundary

choices made. Because we have imposed a Dirichlet boundary condition on the entire bottom boundary, and the wind speed is very strong, we obtain a similar "flat" solution, where the cloud of pollutant remains very close to the bottom boundary and doesn't rise, across the entire domain no matter the velocity or the direction.

Due to this, we once again used the modified boundary conditions introduced in the problem statement and pictured in Figures 6.5 and 3.4.

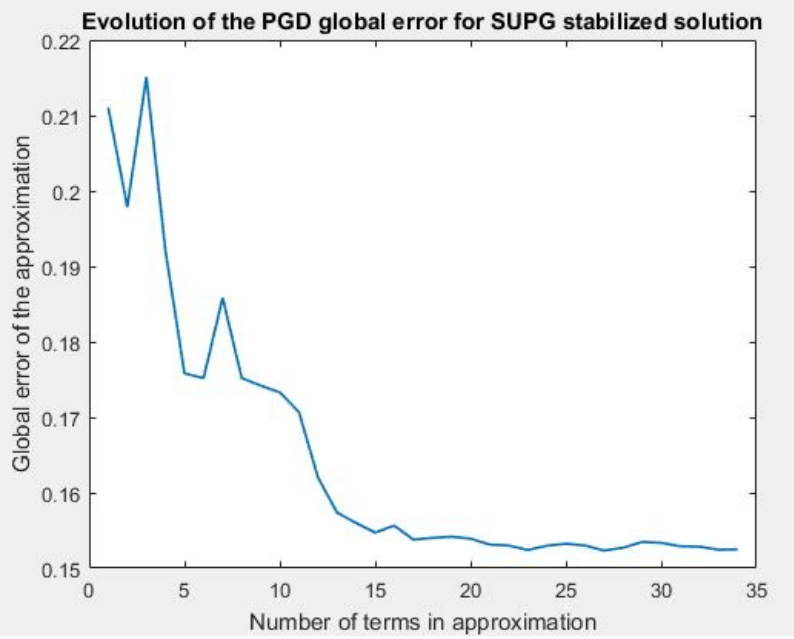

Figure 6.18: Evolution of the global error for a problem with inhomogeneous Dirichlet on the bottom boundary

As shown in Figure 6.18, the error shows a very similar trend to the one in Figure 6.17, but the global error of the approximation is higher because the solution is no longer symmetric and therefore more modes need to be computed to effectively capture all the variability of the solution.

An interesting phenomenon is observed in figures 6.17 and 6.18. While more terms are added to the approximation, the global accuracy of the solution at a certain point stagnates. This means that whatever features the PGD is capturing, they are not contributing to improve the quality of the overall solution. These features are noticeably the oscillations in the solution. Because the solution is partially stabilized, the PGD begins capturing relevant information. However, as it begins characterizing more details, it eventually picks up the instabilities. As these have been damped, β does not increase a lot, but the PGD incorporates them into the solution. Given that these are difficult to separate as a function of the input parameters, many modes are needed to capture these effects, leading to an enrichment without an improvement in accuracy. To improve the results, we now keep on stabilizing the solution.

6.3.3 Fully stabilized formulation

We now fully stabilize the solution with the addition of the crosswind diffusion terms. This was only done for our original problem statement with boundary conditions as seen in Figure 3.3.

The evolution of β for this case is plotted in Figure 6.19, and shows a downward trend, which means that the separation is converging towards a solution.

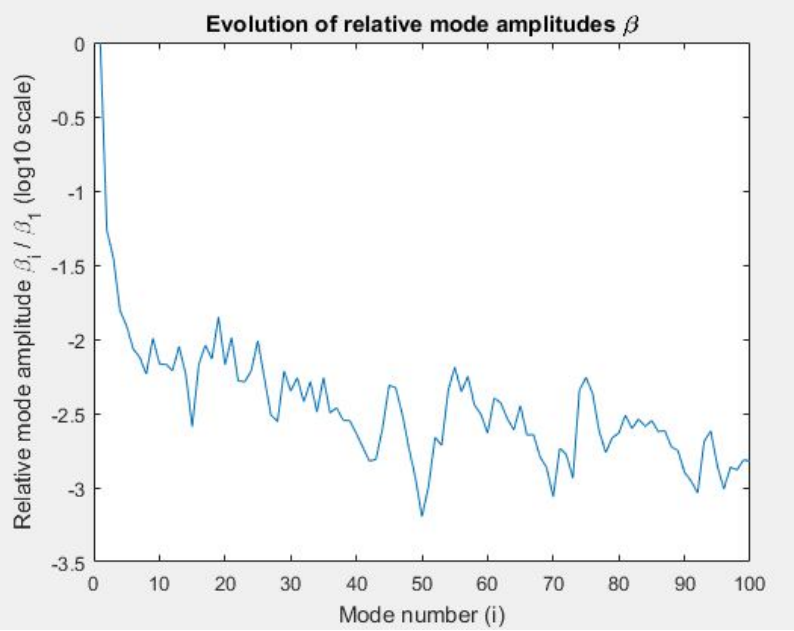

Figure 6.19: Evolution of relative mode amplitude for the fully stabilized problem with homogeneous bottom boundary conditions

However, when we plot the global error trend in Figure 6.20, we see that there is no decrease in the error as we conduct the enrichment even though β seems to be slightly decreasing, signaling a certain convergence. However, the explanation for this is due to the fact that after having linearized the crosswind term, we are no longer solving exactly the same equation. This can be seen in Figure 6.21, where the FEM and the PGD solution (approximated with 50 modes) are being plotted. The error between them remains constant because the PGD algorithm is converging towards a solution which is different from the one in the FEM due to the change in problem formulation. Therefore, in this case, when measuring the error of the approximation with respect to the error of the FEM solution, we must realize that no matter how much we enrich the approximation, a large difference in values will likely remain.

However, a further interesting conclusion can be extracted from Figure 6.21. While the FEM solution appears to be somewhat overdiffused, the new linearization procedure has yielded a stable solution which is much closer to the one we would expect for a very high Péclet number (for this particular example, around 7500). This means that the addition of not consistent stabilization in this particular test case may yield good results, and further research would be useful to figure out if, at a fraction of the cost of solving a nonlinear problem, an alternative linear formulation could be appropriate for this situation.

Results for the synthetic meshes have been mostly in line with what we expected. PGD approximates the solution quite effectively and, despite the fact that it is clear that the linearization of nonlinear terms yields a solution which diverges from the one obtained with the fully stabilized FEM, the PGD approximation yields comparatively good results. The following step will be to implement this algorithm in the computational domain representing an area of the Eixample district in Barcelona which has been obtained following the procedure described in Chapter 5.

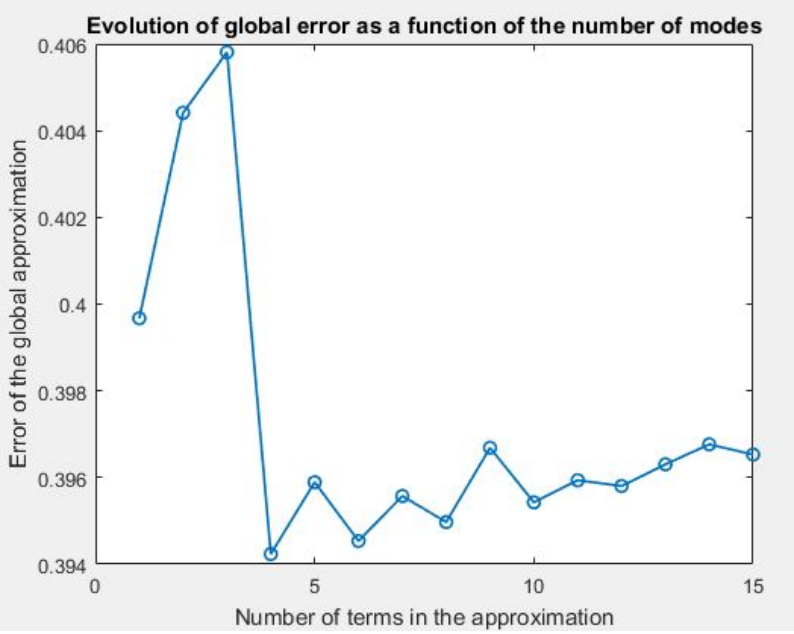

Figure 6.20: Evolution of the global error for the fully stabilized problem with homogeneous bottom boundary conditions

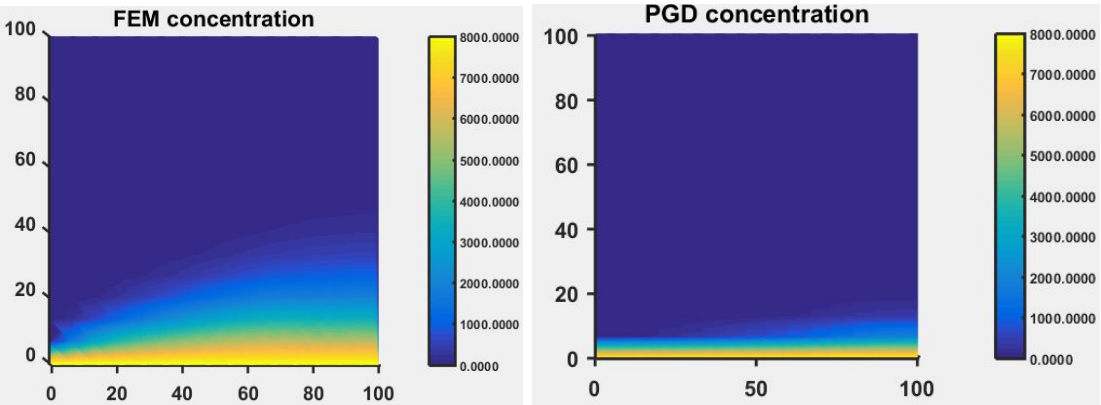

Figure 6.21: Evolution of relative mode amplitude in the non-linear case

6.3.4 Implementation for l'Eixample, Barcelona

In the rest of this section, it is important to take into account that only SUPG stabilization has been implemented, due to the computational constraints of performing iterations on such a large mesh, which rendered non-linear modeling very complicated.

The solution of the transport equations with the FEM yields a baseline solution against which we can compare the PGD approximations. As it can be seen in Figure 6.22, the FEM solution shows the pollutant generated at a park in Barcelona (defined by the nodes shown in the left plot of Figure 5.2) flowing in the direction of the wind, preferably following the directions of the streets. The model behaves like we expect. Pollution is carried in the direction of the wind, which . The strong convection in presence of a velocity field with small vertical components means that the pollutant will tend to stay close to the ground and travel, preferentially, down the streets rather than over the buildings.

MATLAB visualizations were not powerful enough to accurately portray some of the most important features of such a complex solution. Therefore, the results in this section will be visualized using the software Paraview, which offers much more advanced visualization possibilities.

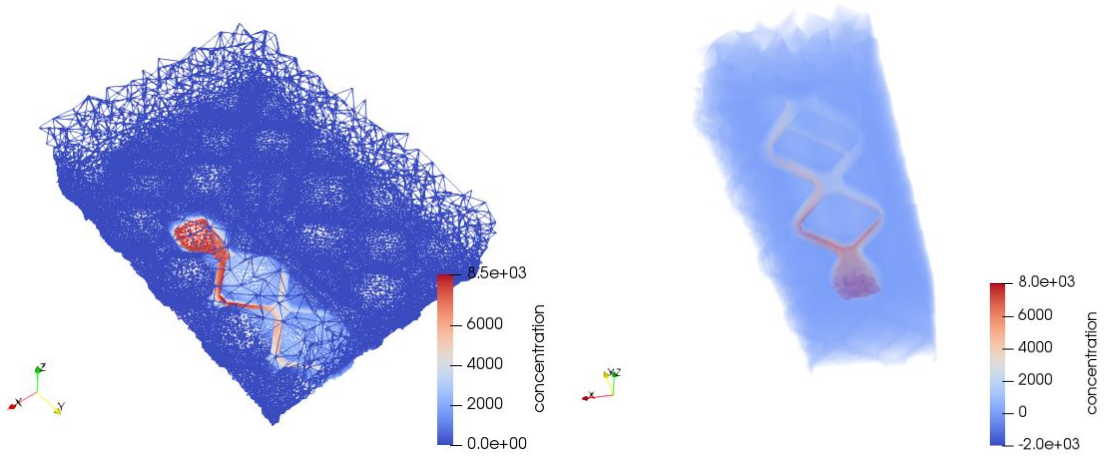

Figure 6.22: FEM solution for a test case in the Barcelona mesh. Visualization includes the wireframe rendering of the mesh and solution (left) and a disperse cloud of pollutant as seen from below (right). Range of legend has been adjusted for improved visualization.

In a first attempt, the SUPG stabilized model with parametric dependencies on both the angle and velocity was implemented for the Eixample in Barcelona. However, it soon became evident that given the computational power available for the project, carrying out a full PGD decomposition with 2 parameters for such a large mesh would be challenging. Even when running the job on the CloneTroop cluster, it was taking more than 24 hours to compute a single mode in the approximation and the initial results were disappointing, showing a rapid increase in β which signaled either that the separation was not being carried out effectively or that instabilities were being captured already in the first modes due to insufficient stabilization with the linearized crosswind diffusion. This can be seen in Figure 6.23.

Due to the high computational burden and the evident lack of good convergence, the parametric dependence of the problem on the velocity was dropped, and we studied a separation of the results based only on the angle of the incoming wind.

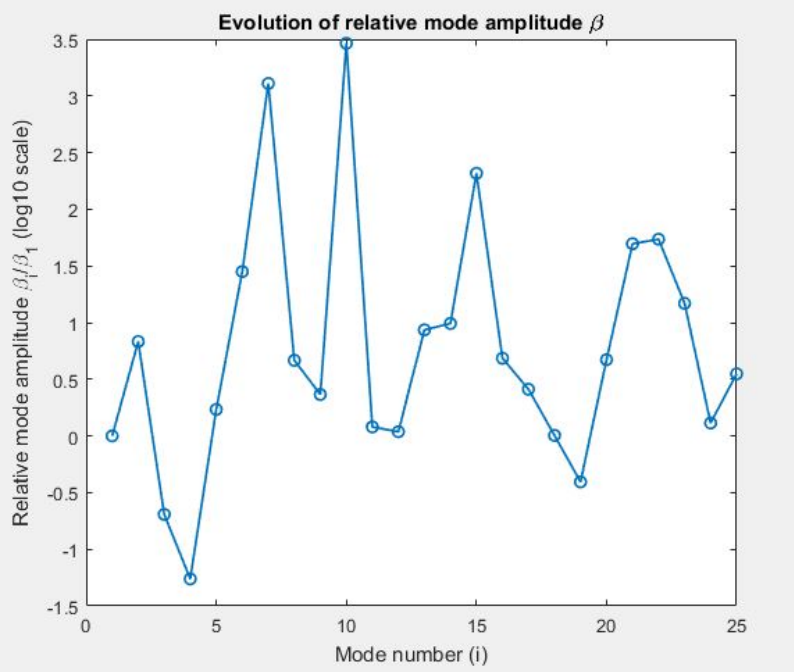

Figure 6.23: Evolution of relative mode amplitude shows no sign of decreasing, showing that the PGD has difficulties capturing the dependency of the most relevant features on the parameters

The evolution of coefficient β for the SUPG-stabilized problem can be seen in Figure 6.23. This shows that the angle parametrization is doing a very bad job at capturing the main features of the solution. By reconstructing the solution and plotting it, we can understand what the modes are actually capturing.

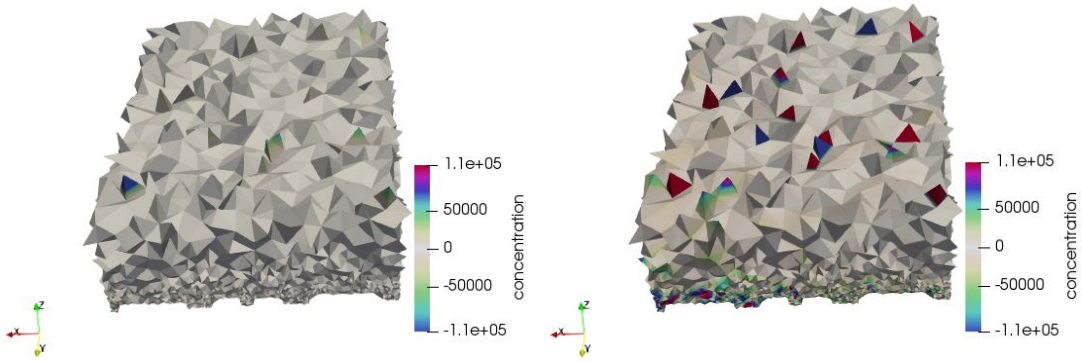

Figure 6.24: Reconstruction of solution for angle $\frac{\pi}{4}$ with 5 modes (left) and 25 modes (right). We see that the enrichment has been capturing instabilities.

As explained in Chapter 5, the computational domain which has finally been used was obtained after pre-processing a larger mesh and extracting a sub-domain from this larger mesh. As such, while the large mesh had very regular boundaries which did not cause numerical instabilities, the boundaries of the extracted mesh are strongly irregular, causing the quality of the solution to worsen. The top boundary proves especially challenging as the obtained elements are strongly irregular. This causes the solution to be very badly behaved at the most irregular elements, arbitrarily under- and overshooting and causing the physical soundness of the solution to greatly decrease.

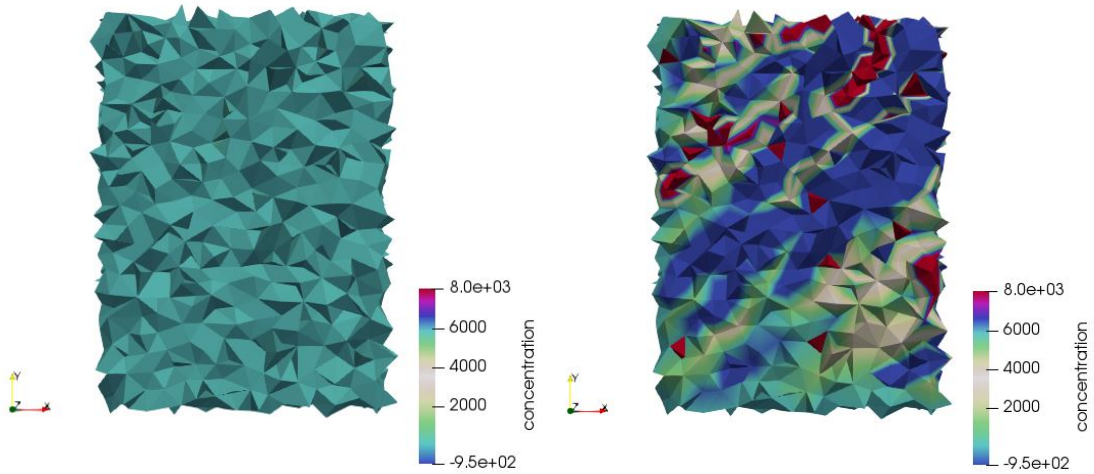

Figure 6.25: Comparison of the reconstructed solution with 25 modes for angles of 0 (left) and $\frac{\pi}{4}$ (left). Bright colors signal extreme concentration values. The PGD approximation has been capturing discontinuities which are characteristic of the intermediate angles, but don't contribute to an angle of 0.

As pictured in Figure 6.24, the increase in the number of terms in the PGD solution has actually led to the capturing of numerical instabilities in the nodes in the upper boundary. This is displayed by the increased number of colored finite elements, which represent extreme concentration values. We can see that the discontinuities being captured are not general for all the angles in Figure 6.25. While for an angle of 0 no oscillations are evident, the upper boundary shows extreme values for an angle of $\frac{\pi}{4}$. This shows that the oscillations cannot be parametrized easily as a function of the angle.

It therefore becomes evident that more stabilization is needed if we want the solution to converge without capturing unnecessary noise. Given that these instabilities happen in the parts of the domain with almost null velocity due to their geometric position, no amount of SUPG can dampen the oscillations generated in these areas. Using other stabilization methods therefore becomes unavoidable.

The parametric dependence of the solution on the angle can therefore be considered to be poor when solving the SUPG-stabilized problem. Oscillations are not angle-dependent, which causes PGD to fail to meaningfully capture the relationship and enrich the solution with many modes characterizing individual oscillations which don't contribute to the improvement of the solution.

Chapter 7

Conclusions and future work

In this thesis, approximated solutions of the advection-diffusion equation in urban areas have been computed using algebraic PGD and introducing the wind speed and angle as parameters. These solutions have been compared to the approximated solutions obtained with the traditional FEM. MATLAB has been the tool of choice to develop the necessary routines to solve the problem, which have been validated by applying them to a simple case. Finally, the performances of the two approaches have been compared for more realistic situations with the objective of understanding if algebraic PGD can be used to overcome certain bottlenecks of the traditional FEM when used to solving the pollution transport equation, therefore making a wider range of applications (real-time modeling, etc.) practical.

The main original contributions of this thesis have been: i) using stabilization procedures in combination with the PGD framework, ii) studying the parametric dependence of the wind module and speed on the solution of the advection-diffusion equation and iii) applying the PGD to study a pollution transport problem in a computational model of an urban area existing in the real world. None of these things have ever been done before to the knowledge of the author or her advisers.

7.1 Conclusions

The PGD approximation shows very promising results for low Péclet number (Pe$<$1) situations. In these cases, where the separation of variables can be computed without any problem, the PGD is capable of capturing the most relevant features of the solution and converges rapidly towards the FEM solution in modes typically in the tenths or low hundreds. While such a low Péclet number is not applicable to pollution transport in urban areas, given that the smallest Péclet number we would be dealing with is of the order of magnitude of 10^4, we see that the use of PGD for other applications where low Péclet are involved seems justified given these initial results.

However, when the Péclet number becomes greater than one and the solution begins oscillating, it becomes harder for the PGD to efficiently converge. This is made evident by the greater number of modes that the solution takes to converge, as well as the large increases in mode amplitude that occur as the enrichment progresses. This is because the oscillations do not depend on the angle nor the velocity of the wind, so it becomes difficult for the PGD to capture the relationships between the parameters and the instabilities. The

number of modes in the approximation therefore increases by a lot of terms as the individual oscillations are characterized by successive modes without resulting in a noticeable improvement in global accuracy.

The addition of SUPG mitigates the most extreme oscillations, decreasing their order of magnitude and thereby helping the PGD capture more meaningful parametric features of the solution once again. However, SUPG stabilization by itself is often insufficient to completely mitigate the instabilities in crosswind directions if the Péclet number is very high. Therefore, when the order of magnitude of the relevant phenomena being captured by the PGD approximations becomes equal to that of the remaining oscillations, the PGD begins characterizing the instabilities, and the solution is enriched with modes which don't contribute much to the global accuracy of the approximation due to the same effect that is observed in formulations of the problem without stabilization.

At this point, discontinuity capturing comes into play by adding diffusion in the crosswind direction to completely stabilize the solution. However, because discontinuity capturing methods are non-linear, they are very difficult to characterize via an algebraic parameter dependence such as the one assumed in PGD. Proxies which add crosswind diffusion linearly have been used instead, which further stabilizes the solution with respect to the SUPG one and allows us to capture more meaningful modes, but questions remain regarding the physical soundness of the resulting solution as well as the appropriate choice of the numerical values which are used to substitute the non-linear terms in the crosswind stabilization scheme.

While results for synthetic meshes were satisfactory, the results of the PGD approximation for advection-diffusion equations in computational domains of actual urban areas was problematic and yielded inconclusive results. Results showed that the approach which worked for idealized urban areas was not appropriate for real ones, likely due to their increased geometric complexity. Furthermore, implementing the PGD in this context poses its own set of challenges other than the generic computational ones. First, the appropriate choice of domain is relevant when working with meshes generated by third parties, as these may not be suited for the resolution of the advection-diffusion equations and may lead to instabilities due to their irregularity, as was the case in this thesis. Second, empirical parameters in the crosswind stabilization scheme may need to be studied on a case-by-case basis as their values may not correspond to those obtained with test cases, which may have led to the divergence in performances. Finally, the large number of PGD modes needed to accurately approximate solutions in complex geometries, where the number of modes needed to appropriate the solution with a high degree of accuracy may be very high, requires efficient storage and successive compression of the approximated solution to make the more extensive testing feasible. Overcoming these difficulties may then allow extensive testing to determine whether using PGD to model pollution transport in real urban areas is feasible at a large scale.

7.2 Future work

Seeing the results obtained with the first test performed on the Barcelona mesh, it becomes evident that adequate mesh creation procedures are necessary. A possible extension of the work performed in this thesis would be to create a protocol to generate suitable meshes with regular boundaries which are not as prone to instabilities as the ones obtained when elements are simply extracted from a larger mesh without considering their geometrical characteristics. Furthermore, a detailed study of the computational bottlenecks associated

with the resolution of the advection-diffusion problems in meshes of relatively large urban areas, as well as the possible introduction of parallelization of the computations, would allow for faster iterations, thereby providing a way to efficiently conduct more detailed numerical analyses to understand the causes of the possible issues.

The approximation of the non-linear terms remains a delicate aspect for the application of parametric PGD to a fully stabilized advection-diffusion problem. This is because the separation of these terms is not straightforward, and their approximation by the means of average values fails to completely stabilize the solution in the most critical parts of the domain. To improve the approximation, a possibility would be to use PGD inside an iterative scheme and update the nonlinear terms with the values resulting from the previous PGD iteration. While problems related to the separation of variables would remain, it would provide a more balanced distribution of additional crosswind diffusivity and result in a less oscillatory solution.

Finally, an interesting feature of PGD that could potentially be exploited is that by setting an appropriate tolerance for stopping the enrichment, PGD can could provide a stabilized solution by itself. For problems with inherent instabilities, lower tolerances for PGD modes are not necessarily always better, as the oscillations will eventually be captured in the approximations if the tolerance is low enough and the algorithm will compute a lot of modes without improving the global accuracy of solution, which leads to high computational costs without much benefit. We could instead be willing to obtain a slightly less accurate solution, but with the main features of the pollution transport solution, by setting the tolerance such that the enrichment stops right before the oscillations are incorporated into the decomposition.

Appendices

Appendix A: Proof of well-posedness

In this appendix, the well-posedness of the weak form (3.6) of the advection-diffusion equation we are trying to solve will be demonstrated. Well-posedness means that [17]:

1. A solution exists
2. The solution is unique within the space where we are looking for the solution (therefore, if v_1 and v_2 are both solutions, it holds that $v_1 = v_2$).
3. The solution depends continuously on the data (parameters, boundary conditions, etc)

Conditions 1 and 2 ensure that if we compute the solution to the weak form various times, it will always yield the same solution, and that this solution will exist. Given that we are looking for the solution in a different solution space as in the original formulation, this is not an obvious corollary. Condition 3 can be seen as a physical constraint, given that continuity is what we would expect from a physically sound solution.

Well-posedness of the weak form of the problem is problem-dependent as the choice of non-Dirichlet boundary conditions will appear in the weak form and thus could affect well-posedness. Therefore, well-posedness is not straightforward. Well-posedness is most often demostrated through the Lax-Milgram lemma, which states that:

Lax-Milgram lemma. *Let V be a Hilbert space with inner product $(\cdot,\cdot)_V$ and associated norm $\|\cdot\|_V$ (i.e. such that $\|w\|_V^2 = (w,w)_V$). Moreover, it is assumed that the following conditions hold:*

- *Continuity of B: $\exists M{>}0$ such that $\forall u, v \in V$, $|B(u,v)| \leq M\|u\|_V\|v\|_V$;*
- *Continuity of F: $\exists C{>}0$ such that $\forall v \in V$, $|F(v)| \leq C\|v\|_V$;*
- *Coercivity of B: $\exists \alpha > 0$ such that $\forall u \in V$, $B(u,u) \geq \alpha\|u\|_V^2$*

Then, the boundary value problem is well-posed, i.e. there exists a solution, this solution is unique and it continuously depends on the data.

We will apply this to the bilinear B and linear C forms defined in equation (3.9):

$$B(u,v) \int_\Omega D\nabla u \cdot \nabla v d\Omega + \int_\Omega \mathbf{v} \cdot \nabla u v d\Omega + \int_{\partial\Omega_{build}} \beta u v ds$$
$$F(v) = 0$$

We now prove the different conditions of the Lax-Milgram lemma. In the following proofs, there are many results and theorems which will be used but not proved given that it is not the main focus of this thesis. For a detailed explanation of all the relevant concepts, the reader can refer to [17] or, given that this reference is academic material not readily accessible nor purchasable, other more easily available materials such as [11] can be used as references.

1) Continuity of B

We can demonstrate that the different terms in B are bounded:

For the diffusion term, we can use the generalization of the Cauchy-Schwarz inequality along with the definition of norms in $H^1(\Omega)$, which allows us to write:

$$\left|D\int_\Omega \nabla u\cdot\nabla v d\Omega\right| \leq |D|\|u\|_V\|v\|_V$$

Concerning the advection term is the test function $v \in H^1$ as defined by the weak form statement of our problem in equation (3.9), the Sobolev embedding theorem for 3D problems concludes that $v \in L^4$. We also know that $\mathbf{v} \in H^1$ as well, given that $\mathbf{v}$ is obtained by the potential flow model and therefore both the terms and their derivatives exist and are square-integrable. By using the same theorem as previously (Sobolev embedding theorem) we reach the conclusion that $\mathbf{v} \in L^4$. Because the product of two L^4 functions is in L^2, ∇u is in L^2 and the product of two L^2 functions is in L^1, we reach the conclusion that the term inside the integral, $\mathbf{v}\cdot\nabla uv$, actually belongs to L^1. This guarantees that the integral exists and is finite by the definition of L^1. Therefore:

$$\left|\int_\Omega \mathbf{v}\cdot\nabla uv d\Omega\right| \leq \|\mathbf{v}\|_{H^1}\|u\|_{H^1}\|v\|_V$$

The Robin term can be bounded by using a combination of the Cauchy-Schwarz inequality and trace inequality. Assuming that β is a positive coefficient:

$$\left|\int_{\partial\Omega_{build}} \beta uv ds\right| \leq \beta\|u\|_{L^2(\partial\Omega,build)}\|v\|_{L^2(\partial\Omega,build)} \leq \beta^*\|u\|_{V(\Omega)}\|v\|_{V(\Omega)}$$

where β^* includes both the contribution of the β as well as the trace inequality constant.

Because we have proved that the three terms in the bilinear form are bounded, the bilinear form itself is bounded due to the property of boundedness which ensures that the finite sum of bounded sets (including vector spaces) is itself bounded. Therefore, $B(u,v)$ is bounded and continuous.

2) Continuity of F

It is trivial to prove that F is continuous given that F=0.

3) Coercivity of B

We need to prove that the term

$$B(v,v) = \int_\Omega D\nabla v \cdot \nabla v d\Omega + \int_\Omega \mathbf{v} \cdot \nabla v v d\Omega + \int_{\partial\Omega_{build}} \beta v v ds =$$
$$D\|\nabla v\|_{L^2}^2 + \frac{1}{2}\int_\Omega \nabla \cdot (\mathbf{v}v^2) d\Omega + \beta \int_{\partial\Omega_{build}} v^2 ds$$

is lower bounded in terms of $\|v\|_V^2$. The equivalence above holds because we are working with a divergence-free flow ($\nabla \cdot \mathbf{v} = 0$) as explained in the problem statement. Remembering that β is a positive constant:

$$B(u,u) = D\|\nabla v\|_{L^2}^2 + \frac{1}{2}\int_\Omega \nabla \cdot (\mathbf{v}v^2) d\Omega + \beta \int_{\partial\Omega_{build}} v^2 ds \geq D\|\nabla v\|_{L^2}^2 + \frac{1}{2}\int_\Omega \nabla \cdot (\mathbf{v}v^2) d\Omega$$

By using the divergence theorem and the definition of the boundary conditions, we can manipulate the advection term to arrive at a convenient expression (excluding the Dirichlet boundary because the test function v is null there):

$$\frac{1}{2}\int_\Omega \nabla \cdot (\mathbf{v}v^2) d\Omega = \frac{1}{2}\int_{\partial\Omega} \mathbf{n} \cdot (\mathbf{v}v^2) ds = \frac{1}{2}\int_{\partial\Omega_N \cup \partial\Omega_R} \mathbf{n} \cdot (\mathbf{v}v^2) ds$$

Because we have used the potential flow theorem to find the velocity field, we know that all boundaries except for the lateral inflow and outflow will result in $\mathbf{v} \cdot \mathbf{n} = 0$ because they are all considered to be non-penetrable boundaries [14]. The inflow is a Dirichlet boundary, so it is not taken into account in the expression, and the only thing left is the outflow where, by definition, $\mathbf{v} \cdot \mathbf{n} > 0$. It should also be noted that D is a positive constant. This leads to the conclusion that:

$$B(u,u) \geq D\|\nabla v\|_{L^2}^2 + \frac{1}{2}\int_\Omega \nabla \cdot (\mathbf{v}v^2) d\Omega \geq D\|\nabla v\|_{L^2}^2 \geq D^*\|v\|_V^2$$

where D^* includes the contributions from both D and the constant in the Poincaré inequality. This proves the coercivity so that the well-posedness of the problem is proved.

Appendix B: Reference elements and numerical integration details

To perform integration on the elements in a practical way, we refer all elements to a reference element which is built in a convenient space as pictured in Figure 3.6. To do so, we define a 3D space of coordinates α, β and γ such that one of the faces of the tetrahedron lies in the plane $\alpha = 0$, another on the plane $\beta = 0$, another on the plane $\gamma = 0$ and the final one on the plane $1-\alpha-\beta-\gamma = 0$. We build the tetrahedral element so that its edges of lengths a, b, and c coincide with the new coordinate axes, and we number the nodes from 1 to 4. We assume node 1 to be the corner at the origin of the coordinate system. We have the following node coordinates in this new system, where we want our element edges to be of unit length:

Node 1: $\alpha = 0, \beta = 0, \gamma = 0$
Node 2: $\alpha = 1, \beta = 0, \gamma = 0$
Node 3: $\alpha = 0, \beta = 1, \gamma = 0$
Node 4: $\alpha = 0, \beta = 0, \gamma = 1$

If we align the coordinates α, β and γ with the usual axes x, y and z, then we can describe the coordinates as follows [20]:

$$\alpha = \frac{x-x_1}{a}, \beta = \frac{y-y_1}{b}, \gamma = \frac{z-z_1}{c}$$

From the expression above, we can easily find that:

$$\frac{d\alpha}{dx} = \frac{1}{a}, \frac{d\beta}{dy} = \frac{1}{b}, \frac{d\gamma}{dz} = \frac{1}{c}$$

and a differential of volume can therefore be expressed as:

$$dx\ dy\ dz = a\ b\ c\ d\alpha\ d\beta\ d\gamma$$

This greatly simplifies the integration of functions $f = f(x, y, z)$ as we can compute everything in this local system of coordinates and then perform a simple change of coordinates to bring the solution back to the real space where our mesh is defined:

$$\int_\Omega f d\Omega = \int\int\int_{V^{(e)}} f(x,y,z)dxdydz = \int_0^1 \int_0^{1-\alpha} \int_0^{1-\beta-\gamma} f(\alpha,\beta,\gamma)abcd\alpha d\beta d\gamma$$

The application of this theory to the advection-diffusion problem is the following. We are working with 4-node linear tetrahedral elements, which means that each element has 4 shape functions, one associated to each of the nodes. Given the coordinates of the reference element, it is therefore easy to express these shape functions as:

$$N_1 = \hat{N}_1 = 1 - \alpha - \beta - \gamma, \quad N_2 = \hat{N}_2 = \alpha, \quad N_3 = \hat{N}_3 = \beta, \quad N_4 = \hat{N}_4 = \gamma$$

which fulfills the conditions stated above, where the shape function for each node has value 1 at the node it refers to and 0 on all the others.

Before going on, it should be noted that the numbering of the nodes cannot be arbitrary. While there isn't any fixed rule regarding how nodes should be numbered, it is necessary to chose an order so that the determinant of the node coordinates when listed in order is positive. This ensures that the integrals will be computed for positive volumes and will result in physically sound solutions given that, if incorrectly listed, the integrals will be computed over negative volumes. In other words, the quantity:

$$det(A) = \begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix}$$

should be strictly positive for all the elements under consideration.

We not only need to know the expression the shape functions N_i in natural coordinates, but also to compute their gradient ∇N_i. We can use the following rule to derive the corresponding terms of the gradient $\hat{\nabla}\hat{N}_i$, which is the gradient of the shape functions in the reference element:

$$\hat{\nabla}\hat{N}_i = \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \mathbf{J}^{(e)} \frac{\partial \mathbf{N}_i}{\partial \mathbf{x}} = \mathbf{J}^{(e)} \nabla N_i^{(e)}$$

Where $\mathbf{J}^{(e)}$ is the Jacobian matrix corresponding to a given element which allows us to change from reference to real coordinates and viceversa. It is therefore trivial to obtain the expression of the gradient in global coordinates, which is what we are looking for:

$$\nabla N_i^{(e)} = (\mathbf{J}^{(e)})^{-1} \hat{\nabla}\hat{N}_i$$

This way of computing the gradients is preferred because we don't need to compute the derivatives of the shape functions with respect to the global coordinates for each element, which would be bothersome for those elements with edges not aligned with the coordinate axes. We can instead just compute the gradient of the shape functions for a generic element with edges aligned with the axes, which makes the computations of gradients trivial, and then multiply the result by the inverse of the Jacobian. This way of computing results is more convenient than obtaining the derivatives of the shape functions for each element.

A volume differential can thus be computed as:

$$dxdydz = |\mathbf{J}^{(e)}| d\xi d\eta d\zeta$$

The same logic can be applied to the computation of the integrals along the boundaries. Boundaries of the 3D domain are two-dimensional: therefore, the restriction of coordinates to x, y, ξ and η are applied instead. The surface differential is therefore expressed as:

$dxdy = |\mathbf{J}_s^{(e)}|d\xi d\eta,$

where $\mathbf{J}_s^{(e)}$ can be computed as follows:

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = \mathbf{J}_s^{(e)} \frac{\partial \mathbf{N}_i}{\partial \mathbf{x}}$$

These changes of coordinates can be incorporated into equation (3.10) to obtain the form which will be actually implemented in the code. For our pollution transport problem, we get:

$$\begin{aligned} &\sum_{i=1}^{M} (\int_{\Omega^{(e)}} (D(\mathbf{J}^{(e)})^{-1} \hat{\nabla} \hat{N}_i \cdot \nabla (\mathbf{J}^{(e)})^{-1} \hat{\nabla} \hat{N}_j + \mathbf{v} \cdot \nabla (\mathbf{J}^{(e)})^{-1} \hat{\nabla} \hat{N}_i \hat{N}_j) |\mathbf{J}^{(e)}| d\xi d\eta d\zeta + \\ &\int_{\Omega^{(e)}_{build,sides}} \beta \hat{N}_i \hat{N}_j |\mathbf{J}_s^{(e)}| d\xi d\eta) c_i = 0 \end{aligned} \tag{1}$$

for every element where the equation holds. This is the entire domain for the volume integrals, and the Robin boundary for the surface integral. Everything is computed in terms of the local elements, and then the Jacobian converts it to the global system.

To solve the finite element form of the problem, we need to compute the integrals of the terms in the local space as noted above. However, a computer cannot solve for integrals analytically. Instead, these must be cast in a form such that they can be computed as weighted sums using the so called quadrature rules. The result we obtain will therefore be a numerical value which will reflect the computation of the integral of each element, and not a generic expression. This is usually referred to as numerical integration.

Let's assume that we want to integrate a function $f = f(x)$ such that $x \in [-1, 1]$. One of the possible approximations that we could obtain is the following:

$\int_{-1}^{1} f(x)dx \approx 2f(0)$

which is often referred to as the midpoint rule. We also know that, if f is linear within its domain, then this yields the exact value instead of just an approximation. If we take for instance, $f(x) = x+2$, the integral between -1 and 1 is 4, which is exactly $2f(0) = 2 \cdot 2 = 4$.

The same kind of approximation can be generalized to polynomials of a higher order. These are usually known as the Gauss-Legendre quadrature rules. For a 1D problem, these are usually defined as weighted sums:

$\int_{-1}^{1} f(x)dx = \sum_{k=1}^{n} w_{n,k} f(x_{n,k}) + \varepsilon_n(f) \approx \sum_{k=1}^{n} w_{n,k} f(x_{n,k})$

where $x_{n,k}$ are usually referred to as Gauss-Legendre points, $w_{n,k}$ are the weights associated to these points (which are found in tables) and $\varepsilon_n(f)$ is the truncation error, which only depends on the $(2n)^{th}$ derivative of f. n refers to the degree of the approximation.

This formula is often generalized to higher dimensions. If the vector in which we find the finite element can be described as a tensor-product of d 1D spaces, then the Gauss quadrature can just be expressed as the tensor product of d 1D Gauss-Legendre rules.

If this is not the case, then we need to come up with specific Gauss quadrature rules, which are usually written in terms of the barycentric coordinates $(\lambda_0, ..., \lambda_d)$. This is the case in this report, given that elements are tetrahedral. For a unit tetrahedron and for a unit triangle K, the formula above therefore becomes:

$\int_K f(\mathbf{x})dK \approx \sum_{k=1}^{n} w_{n,k} f(\lambda_{0,n,k}, ..., \lambda_{d,n,k})$

where the number of points increases as the order of the integrated polynomial becomes higher.

Because we are working with linear tetrahedra, we would have $d = 2$ (because we start numbering at 0) in the sum above. In the case of linear elements, the simplest Gauss quadrature rules applies, where a single point suffices to obtain the exact value of the integrated term inside the finite element. Therefore, $n = 1$. This point is noted by coordinates $\lambda_{0,1,k}, \lambda_{1,1,k}, \lambda_{2,1,k}$ and corresponds to the barycenter of the tetrahedron. The position of the integration points for elements are tabulated: for a linear tetrahedral element, the barycenter corresponds to natural coordinates $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

The same idea that has been explained for tetrahedra can be applied to 2D surfaces, which in our case is necessary to integrate the boundary conditions. In this case, the integration point for single-point integration corresponds to the barycenter of the triangle and we can look in tables for the corresponding Gauss quadrature rule and weights.

The quadrature rule that is finally implemented for an advection-diffusion problem with the FEM form described in equation (3.10) solved by a 1-point integration approximation would be:

$$\sum_{i=1}^{M}(\sum_{k=1}^{4} w_{1,k}(D(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i \cdot (\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_j + \mathbf{v}\cdot(\mathbf{J}^{(e)})^{-1}\hat{\nabla}\hat{N}_i\hat{N}_j)|\mathbf{J}^{(e)}| + \sum_{k=1}^{3} w_{1,k}\beta N_i N_j |\mathbf{J}_s^{(e)}|)c_i = 0$$

Appendix C: Alternating directions scheme for PGD

In this appendix, a particular realization of the alternating directions scheme for an non-stabilized advection-diffusion problem will be presented. For illustration purposes, the PGD applied here is non-algebraic, and is based instead on the weak form of the advection-diffusion equation. Stabilization has been foregone to ease notation.

We start from the weak form of our problem which is stated in eq. (3.6). The weak form is actually a weighted residual form of the equation, and by choosing a specific type of test function, we can arrive at an expression to be easily computed by our algorithm. The weak form of the problem is recast as follows:

$$\int_{\Omega} (D\nabla c \cdot \nabla u^* + \mathbf{v} \cdot \nabla c u^*) d\Omega + \int_{\partial\Omega_{build,sides}} \beta c u^* ds = 0 \tag{2}$$

to distinguish the test functions v of the FEM case from the test functions u^* of the PGD case. This equation evidently holds true for any appropriately chosen test function u^*.

For the purposes of the variable separation, we separate the velocity $\mathbf{v}$ into its component parts: $\mathbf{v} = \nu(\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta))$. θ determines the angle of the velocity, and the module is then controlled by ν. Given that the solution of the velocity results from a potential flow model, this yields the complete range of possibilities that we will encounter. We substitute the expression for the velocity into the equation:

$$\int_{\Omega} D\nabla c \cdot \nabla u^* + \nu(\mathbf{v}_1 \sin(\theta) + \mathbf{v}_2 \cos(\theta)) \cdot \nabla c u^* d\Omega + \int_{\partial\Omega_{build,sides}} \beta c u^* ds \tag{3}$$

We want to find the solution $c = c(\mathbf{x}, \theta, \nu)$, where θ is the angle of the incoming wind and ν is the module of the wind. Our goal is to obtain a PGD approximation of the solution, which is expressed in separated form. The desired expression of the approximated solution would be of the form:

$c^N(\mathbf{x}, \theta, \nu) = \sum_{i=1}^{N} X_i(\mathbf{x}) \times O_i(\theta) \times V_i(\nu)$

Where we want the approximated solution c^N to be as close as possible to the actual solution of equation (3.2).

The solution is computed by successive approximations. Let's assume we have computed the first $n-1$ terms of the PGD approximation such that:

$c^{n-1}(\mathbf{x},\theta,\nu) = \sum_{i=1}^{n-1} X_i(\mathbf{x}) \cdot O_i(\theta) \cdot V_i(\nu)$

We are now interested in enriching this approximation, that is, computing the next term of the sum for this particular problem:

$c^{n}(\mathbf{x},\theta,\nu) = c^{n-1}(\mathbf{x},\theta,\nu) + X_n(\mathbf{x}) \cdot O_n(\theta) \cdot V_n(\nu)$

For each enrichment step, the term which needs to be computed is non-linear given that it is expressed as a product of functions. The unknown functions which are required for the enrichment must be computed from the current known terms. To compute the solution, we must use an iterative solver.

The preferred way is to use an alternating direction strategy which computes $X_n(\mathbf{x})$ from $O_{n-1}(\theta)$ and $V_{n-1}(\nu)$, then $O_n(\theta)$ from $X_n(\mathbf{x})$ and $V_{n-1}(\nu)$, and finally $V_n(\nu)$ is computed from $X_n(\mathbf{x})$ and $O_n(\theta)$. An arbitrary guess is used as an initial approximation. The iterations proceed until they reach a fixed point within a user-determined tolerance ϵ [3]. This fixed point is determined by the difference between successive approximations: if the difference is small, it means that the new solution found by the algorithm is very close to the one from the previous iteration and thus a fixed point has been reached. Therefore, in our case, the algorithm would stop when:

$$\frac{||X_n^p(\mathbf{x}) \cdot O_n^p(\theta) \cdot V_n^p(\nu) - X_n^{p-1}(\mathbf{x}) \cdot O_n^{p-1}(\theta) \cdot V_n^{p-1}(\nu)||}{||X_n^{p-1}(\mathbf{x}) \cdot O_n^{p-1}(\theta) \cdot V_n^{p-1}(\nu)} < \epsilon$$

Each iteration of the alternating directions strategy consists in 3 steps:

1) Calculating $X_n^p(\mathbf{x})$ from $O_n^{p-1}(\theta)$ and $V_n^{p-1}(\nu)$

Assuming we have arrived at step n, the approximation we have is the one given by the expression:

$c^{n}(\mathbf{x},\theta,\nu) = \sum_{i=1}^{n-1} X_i(\mathbf{x}) \cdot O_i(\theta) \cdot V_i(\nu) + X_n^p(\mathbf{x}) \cdot O_n^{p-1}(\theta) \cdot V_n^{p-1}(\nu)$

Where the only unknown is $X_n^p(\mathbf{x})$. To compute the weighted residuals of the advection-diffusion problem (3) we use the weight function:

$u^*(\mathbf{x},\theta,\nu) = X_n^*(\mathbf{x}) \cdot O_n^{p-1}(\theta) \cdot V_n^{p-1}(\nu)$

From now on, for simplification of the notation, we will drop the explicit dependence of the functions on the variables and on the current iteration of the alternating direction scheme. Therefore, for the unknowns at iteration n, $X^n(\mathbf{x}) = R, O^n(\theta) = Q$ and $V^n(\nu) = W$. This means that the unknown becomes:

$c = c^{n-1} + R \cdot Q \cdot W$

We can then introduce the expressions derived above for the unknown and the test functions into the Galerkin formulation of our problem, which we arrive at as described in the section above, and we obtain the following:

$$\int_{\Omega\times\Omega_\theta\times\Omega_\nu}(D\cdot\nabla R\cdot Q\cdot W\cdot\nabla R^*\cdot Q\cdot W+\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla R\cdot Q\cdot W\cdot R^*\cdot Q\cdot W+$$
$$\beta\cdot R\cdot Q\cdot W\cdot R^*\cdot Q\cdot W)d\mathbf{x}d\theta d\nu=$$
$$-\int_{\Omega\times\Omega_\theta\times\Omega_\nu}\sum_{i=1}^{n-1}(D\cdot\nabla X_i\cdot O_i\cdot V_i\cdot\nabla R^*\cdot Q\cdot W+$$
$$\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla X_i\cdot O_i\cdot V_i\cdot R^*\cdot Q\cdot W+\beta\cdot X_i\cdot O_i\cdot V_i\cdot R^*\cdot Q\cdot W)d\mathbf{x}d\theta d\nu \tag{4}$$

We will now introduce the following notation to refer to the integrals of the known quantities at each step of the derivation:

$$\begin{aligned}
r_1&=\int_\Omega\nabla R\cdot\nabla R d\mathbf{x} & q_1&=\int_{\Omega_\theta}Q^2\sin(\theta)d\theta & &\\
r_2&=\int_\Omega R\nabla R d\mathbf{x} & q_2&=\int_{\Omega_\theta}Q^2\cos(\theta)d\theta & w_1&=\int_\nu W^2 d\nu\\
r_3&=\int_\Omega RR d\mathbf{x} & q_3&=\int_{\Omega_\theta}Q\cdot O_i\sin(\theta)d\theta & w_2&=\int_\nu W\cdot V_i d\nu\\
r_4&=\int_\Omega R\cdot X_i d\mathbf{x} & q_4&=\int_{\Omega_\theta}Q\cdot O_i\cos(\theta)d\theta & w_3&=\int_\nu \nu\cdot W^2 d\nu\\
r_5&=\int_\Omega\nabla R\cdot\nabla X_i d\mathbf{x} & q_5&=\int_{\Omega_\theta}Q^2 d\theta & w_4&=\int_\nu \nu\cdot W\cdot V_i d\nu\\
r_6&=\int_\Omega R\cdot\nabla X_i d\mathbf{x} & q_6&=\int_{\Omega_\theta}Q\cdot O_i d\theta & &
\end{aligned} \tag{5}$$

The integral above can therefore be recast as follows under the new notation:

$$\int_\Omega(D\cdot q_5\cdot w_1\cdot\nabla R\cdot\nabla R^*+v_3(\mathbf{v}_1q_1+\mathbf{v}_2q_2)\cdot\nabla R\cdot R^*+\beta q_5 w_1\cdot R\cdot X^*)d\mathbf{x}=$$
$$-\int_\Omega\sum_{i=1}^{n-1}(D\cdot q_6\cdot w_2\cdot\nabla X_i\cdot\nabla R^*+v_4(\mathbf{v}_1q_3+\mathbf{v}_2q_4)\cdot\nabla X_i\cdot R^*+\beta q_6\cdot w_2\cdot X_i\cdot X^*)d\mathbf{x} \tag{6}$$

The equation above is the weighted form residual of a steady-state BVP which can be solved using any appropriate technique, such as the FEM.

2) Computing $O_n^p(\theta)(Q)$ from $X_n^p(\mathbf{x})(R)$ and $V_n^{p-1}(\nu)(W)$

We now change the test function to suit our current unknown O. Given that X and V are known at this step, we can take their variations to be zero, and the test function thus becomes:

$$u^*(\mathbf{x},\theta,\nu)=R(\mathbf{x})\cdot Q^*(\theta)\cdot W(\nu)$$

By injecting this into (3) and introducing the decomposition of the unknown term, we obtain the following equation:

$$\int_{\Omega\times\Omega_\theta\times\Omega_\nu}(D\cdot\nabla R\cdot Q\cdot W\cdot\nabla R\cdot Q^*\cdot W+\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla R\cdot Q\cdot W\cdot R\cdot Q^*\cdot W+$$
$$\beta\cdot R\cdot Q\cdot W\cdot R\cdot Q^*\cdot W)d\mathbf{x}d\theta d\nu=$$
$$-\int_{\Omega\times\Omega_\theta\times\Omega_\nu}\sum_{i=1}^{n-1}(D\cdot\nabla X_i\cdot O_i\cdot V_i\cdot\nabla R\cdot Q^*\cdot W+$$
$$\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla X_i\cdot O_i\cdot V_i\cdot R\cdot Q^*\cdot W+\beta\cdot X_i\cdot O_i\cdot V_i\cdot R\cdot Q^*\cdot W)d\mathbf{x}d\theta d\nu \tag{7}$$

We can now integrate these forms over $\Omega\times\Omega_\nu$ and, by doing appropriate changes of variables with the terms found in (5) and we obtain the following equation:

$$\begin{aligned}&\int_{\Omega_\theta}Q^*\cdot(-D\cdot r_2\cdot w_1\cdot Q+r_3\cdot w_3\cdot(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot Q)d\theta=\\&-\int_{\Omega_\theta}Q^*\cdot(-\sum_{i=1}^{n-1}D\cdot r_5\cdot O_i\cdot w_2+\sum_{i=1}^{n-1}(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot r_6\cdot O_i\cdot w_4)d\theta\end{aligned} \tag{8}$$

Given that the original equation included no derivatives with respect to the velocity, which the angle defines, there are no differential operators inside this new expression. We therefore do not need to solve the weighted residual form of this equation, which can instead be expressed in its strong formulation and solved as an algebraic equation:

$$\begin{aligned}&Q\cdot(-D\cdot r_2\cdot w_1+r_3\cdot v_3\cdot(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta)))=\\&\sum_{i=1}^{n-1}(-D\cdot r_5\cdot w_2+(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot r_6\cdot w_4)\cdot O_i\end{aligned} \tag{9}$$

By solving this equation directly, we can find the expression for $O(\theta)$ for each value of θ we want to compute our solution for.

3) Obtaining $V_n^p(\nu)(W)$ from $X_n^p(\mathbf{x})(R)$ and $O_n^p(\theta)(W)$

Finally, the last step in the alternated directions scheme is to compute the final function in the separated approximation.

The test function is now modified to be:

$u^*(\mathbf{x},\theta,\nu)=R(\mathbf{x})\cdot Q(\theta)\cdot W^*(\nu)$

By substituting this new test function into equation (3) together with the corresponding separated decomposition for the unknown, we obtain the following expression:

$$\int_{\Omega\times\Omega_\theta\times\Omega_\nu}(D\cdot\nabla R\cdot Q\cdot W\cdot\nabla R\cdot Q\cdot W^*+\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla R\cdot Q\cdot W\cdot R\cdot Q\cdot W^*+$$
$$\beta\cdot R\cdot Q\cdot W\cdot R\cdot Q\cdot W^*)d\mathbf{x}d\theta d\nu=$$
$$-\int_{\Omega\times\Omega_\theta\times\Omega_\nu}\sum_{i=1}^{n-1}(D\cdot\nabla X_i\cdot O_i\cdot V_i\cdot\nabla R\cdot Q\cdot W^*+$$
$$\nu(\mathbf{v}_1\sin(\theta)+\mathbf{v}_2\cos(\theta))\cdot\nabla X_i\cdot O_i\cdot V_i\cdot R\cdot Q\cdot W^*+\beta\cdot X_i\cdot O_i\cdot V_i\cdot R\cdot Q\cdot W^*)d\mathbf{x}d\theta d\nu \tag{10}$$

Because our only unknown is related to the module of the velocity, we can integrate over the domain $\Omega\times\Omega_\theta$ and express the resulting terms with the notation defined in (5), so that the equation becomes:

$$\begin{aligned}&\int_{\Omega_\nu}V^*(D\cdot r_1\cdot q_5\cdot W+\nu(\mathbf{v}_1q_1+\mathbf{v}_2q_2)\cdot r_2\cdot W+\beta\cdot r_3\cdot q_5\cdot V)d\nu=\\&-\int_{\Omega_\nu}\sum_{i=1}^{n-1}V^*(D\cdot r_6\cdot q_6\cdot V_i+w_{\|}(\mathbf{v}_1q_3+\mathbf{v}_2q_4)\cdot r_6\cdot V_i+\beta\cdot r_4\cdot q_6\cdot V_i)d\nu\end{aligned} \tag{11}$$

Just like in the previous step, there are no differential operators involved in the weighted residual form of this equation given that the velocity is not affected by this type of operator. Therefore, this equation can be recast in strong form and solved algebraically:

$$V\cdot(D\cdot r_1\cdot q_5+\nu(\mathbf{v}_1\cdot q_1+\mathbf{v}_2\cdot q_2)\cdot r_2+\beta\cdot r_3\cdot q_5)=-\sum_{i=1}^{n-1}(D\cdot r_6\cdot q_6+\nu(\mathbf{v}_1\cdot q_3+\mathbf{v}_2\cdot q_4)\cdot r_6+\beta\cdot r_4\cdot q_6)\cdot V_i \tag{12}$$

We have now obtained the approximations for terms R, O and V in step n. We now check our approximation against a convergence criterion to determine if it is accurate enough or if we should continue with the enrichment. The stopping criterion is based on the relative change in the solution after the enrichment: we validate the iteration of the change with respect to the previous one is below a certain threshold. In particular, for our case, a particular stopping criterion could be to stop iterating when the difference between two successive iterations is small enough [23]:

$||c^n-c^{n-1}||<\epsilon||c^n||$

Where ϵ is a user-determined tolerance.

If the difference between iterations is larger than ϵ, we now know the approximation functions at step n, and begin once again the alternated directions algorithm at step $n+1$.

Given that the evolution of the norm of the difference between successive iterations is unknown and may be complicated to estimate, a maximum number of iterations is also usually specified to ensure that the algorithm finishes within a reasonable time frame.

We carry out the normalization of the solutions once the fixed point iteration has converged.

As noted in [3], the addition of extra coordinates as problem parameters does not alter the overall complexity of the PGD procedure, although their introduction often means that to reach a given accuracy, the number of terms in the separated approximation must increase.

Bibliography

[1] Alfaro I, González D, Zlotnik S, Díez P, Cueto E, Chinesta F. (2015) *An error estimator for real-time simulators based on model order reduction* Advanced Modeling and Simulation in Engineering Sciences, pp.2-30
https://amses-journal.springeropen.com/articles/10.1186/s40323-015-0050-8

[2] Anuario Estadístico de la Ciudad de Barcelona 2018 - Territorio, clima y medio ambiente - Clima - Viento (2018)
http://www.bcn.cat/estadistica/castella/dades/anuari/cap01/C0102050.htm

[3] Chinesta F, Keunings R, Leygue A (2014) *The Proper Generalized Decomposition for Advanced Numerical Simulations - A Primer*. Springer Verlag.

[4] Chinesta F, Leygue A, Bordeu F, Cueto E, Gonzalez D, et al. (2013) *PGD-Based Computational Vademecum for Efficient Design, Optimization and Control*. Archives of Computational Methods in Engineering, Springer Verlag, 20 (1), pp.31 - 59. https://hal.archives-ouvertes.fr/hal-01515083/document

[5] Codina R. (1993) *A discontinuity-capturing crosswind-dissipation for the finite element solution of the convection-diffusion equation*. Computer Methods in Applied Mechanics and Engineering. Vol. 110, pp.325-342.
https://deca.upc.edu/en/people/ramon.codina/publications-1/articles-in-journals/artweb00

[6] Cueto E, González D, Alfaro I. (2016) *Proper Generalized Decompositions: An Introduction to Computer Implementation with Matlab*. SpringerBriefs in Applied Sciences and Technology.

[7] Díez P, Zlotnik S, García-González A, Huerta A. (2018) *Algebraic PGD for tensor separation and compression: An algorithmic approach* Comptes Rendus Mécanique, Volume 346, Issue 7, pp. 501-514

[8] Dixon S.L, Hall C.A. (2010) *Fluid Mechanics and Thermodynamics of Turbomachinery (Sixth Edition* Butterworth-Heinemann, England.

[9] Donea J, Huerta A. (2003) *Finite Element Methods for Flow Problems*. John Wiley and Sons Ltd., West Sussex, England. ´

[10] Falco A, Nouy A. (2011) *A Proper Generalized Decomposition for the solution of elliptic problems in abstract form by using a functional Eckart-Young approach*. Journal of Mathematical Analysis and Applications, Elsevier, 376 (2), pp.469-480.

[11] Formaggia L, Saleri F, Veneziani A. 2012. *Solving Numerical PDEs: Problems, Applications, Exercises*. Spinger Verlag, Milan, Italy.

[12] Ghorani-Azam A, Riahi-Zanjani B, Balali-Mood M. (2016) *Effects of air pollution on human health and practical measures for prevention in Iran*. J Res Med Sci. 2016 Sep 1;21:65.

[13] Henneron T, Clenet S. (2014) *Proper Generalized Decomposition method applied to solve 3D Magneto Quasistatic Field Problems coupling with External Electric Circuits.* IEEE Transactions on Magnetics, Institute of Electrical and Electronics Engineers, 51 (6), pp.1-10.
https://hal.archives-ouvertes.fr/hal-01187425/document

[14] Kersalé E. *Course 2620 - Fluid Dynamics I - Chapter 4 - Lecture notes.*
http://www1.maths.leeds.ac.uk/~kersale/2620/Notes/chapter_4.pdf

[15] Kuzmin D. *A Guide to Numerical Methods for Transport Equations.*
http://www.mathematik.uni-dortmund.de/ kuzmin/Transport.pdf

[16] Labotatori de Cálcul Numèric *Materials for Course "Computational Engineering", chapter 2.5: Nonlinear problems, academic year 2017-2018* Obtained from Atenea

[17] Lafitte P. *Analyse Théorique et Numérique des Équations aux Dérivées Partielles.* Coursebook for course MA1400 of the Centrale Paris Engineering Program. Academic year 2015/2016.

[18] Li J. *Fluid Mechanics 1 Class notes: Boundary Layer Theory - 3A1 - Cambridge University.*
http://www2.eng.cam.ac.uk/ jl305/3A1/LectureNotes.pdf

[19] Magrinyà F. (2010) *El Ensanche de Barcelona y la modernidad de las teorías urbanísticas de Cerdà.* Ingeniería y territorio: revista del Colegio de Ingenieros de Caminos, Canales y Puertos, n. 88, p. 68-75.
http://www.ciccp.es/revistaIT/textos/pdf/09.%20Francesc%20Magriny%C3%A0.pdf

[20] Oñate E. (1995) *Cálculo de Estructuras por el Método de Elementos Finitos* 2nd Edition, CIMNE, Barcelona.

[21] Peden D.B. (2018) *The Unexpected Health Effects of Air Pollution.* North Carolina Medical Journal September-October vol. 79 no. 5 309-311.
http://www.ncmedicaljournal.com/content/79/5/309.full

[22] Puel G, Barbarulo A. *Applications of the Finite Element Method.* Coursebook for course MG2817 of the Centrale Paris Engineering Program. Academic year 2016/2017.

[23] Sibileau A, García-González A, Auricchio F, Morganti S, Díez P. (2018) *Explicit parametric solutions of lattice structures with proper generalized decomposition (PGD)* Computational Mechanics, Volume 62, Number 4, pp.871-891

[24] Spurk J.H, Aksel N. (2008) *Fluid Mechanics - Second Edition.* Springer-Verlag, Berlin, Germany.
http://fma.if.usp.br/ eabdalla/exacta/000m1.pdf

[25] United States Environmental Protection Agency. *United States Code, 2013 Edition.* Title 42, Chapter 85, Subchapter III, Section 7602.
https://www.govinfo.gov/content/pkg/USCODE-2013-title42/html/USCODE-2013-title42-chap85-

[26] United States Environmental Protection Agency. *United States Code, 2010 Edition.* Title 42, Chapter 85, Section 7408.
https://www.govinfo.gov/content/pkg/USCODE-2010-title42/html/USCODE-2010-title42-chap85.

[27] John V, Knobloch P. *A Computational Comparison of Methods Diminishing Spurious Osillations in Finite Element Solutions of Convection-Diffusion Equations.*
http://msekce.karlin.mff.cuni.cz/ knobloch/PAPERS/panm13.pdf

[28] World Health Organization. Health, environment, and sustainable development - Air Pollution
https://www.who.int/sustainable-development/cities/health-risks/air-pollution/en/

[29] Yeoh GH, Tu J. (2010) *Computational Techniques for Multiphase Flows - Chapter 6 - Gas–Liquid Flows*. Butterworth-Heinemann.

[30] Zhang X, Chen X, Zhang X. (2018) *The impact of exposure to air pollution on cognitive performance*. PNAS 115 (37) 9193-9197
https://www.pnas.org/content/115/37/9193

[31] Zhuang RN, Li X, Tu J. (2014) *Should Different Gaseous Contaminants Be Treated Differently In CFD Indoor Simulations?* WIT Transactions on Ecology and the Environment, Vol 183, pp. 353-362.

[32] Zlotnik S, Díez P, Modesto D, Huerta A. (2015) *Proper Generalized Decomposition of a geometrically parametrized heat problem with geophysical applications* International Journal for Numerical Methods in Engineering. 103. https://core.ac.uk/download/pdf/41822447.pdf