UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

CIMNE

TREBALL DE FI DE GRAU

TFG TITLE: Numerical Implementation of a Mixed Finite Element Formulation for Convection-Diffusion Problems

DEGREE: Grau en Enginyeria d'Aeronavegació

AUTHOR: Iván Padilla Montero

ADVISOR: Adeline de Villardi de Montlaur

SUPERVISORS: Jordi Pons Prats, Riccardo Rossi

DATE: July 10, 2014

Title : Numerical Implementation of a Mixed Finite Element Formulation for Convection-Diffusion Problems

Author: Iván Padilla Montero

Advisor: Adeline de Villardi de Montlaur

Supervisors: Jordi Pons Prats, Riccardo Rossi

Date: July 10, 2014

Overview

This document aims to the numerical solution of convection-diffusion problems in a fluid dynamics context by means of the Finite Element Method (FEM). It describes the classical finite element solution of convection-diffusion problems and presents the implementation and validation of a new formulation for improving the accuracy of the standard approach.
On first place, the importance and need of numerical convection-diffusion models for Computational Fluid Dynamics (CFD) is emphasized, highlighting the similarities between the convection-diffusion equation and the governing equations of fluid dynamics for incompressible flow.
The basic aspects of the finite element method needed for the standard solution of general convection-diffusion problems are then explained and applied to the steady state case. These include the weak formulation of the initial boundary value problem for the convection-diffusion equation and the posterior finite element spatial discretization of the weak form based on the Galerkin method. After their application to the steady transport equation a simple numerical test is performed to show that the standard Galerkin formulation is not stable in convection-dominated situations, and the need for stabilization is justified.
Attention is then focused on the analysis of the truncation error provided by the Galerkin formulation, leading to the derivation of a classical stabilization technique based on the addition of artificial diffusion along the flow direction, the so-called streamline-upwind (SU) schemes. Next, a more general and modern stabilization approach known as the Sub-Grid-Scale (SGS) method is described, showing that SU schemes are a particular case of it.
Taking into account all the concepts explained, a new mixed finite element formulation for convection-diffusion problems is presented. It has been proposed by Dr. Riccardo Rossi, a researcher from the International Center for Numerical Methods in Engineering (CIMNE), and consists on extending the original convection-diffusion equation to a system in mixed form in which both the unknown variable and its gradient are computed simultaneously, leading to an increase in the convergence rate of the solution. The formulation, which had not been tested before, is then implemented and validated by means of a multiphysics finite element software called `Kratos`. Eventually, the obtained results are analyzed, showing the improved performance of the mixed formulation in pure diffusion problems.

Título: Implementación Numérica de una Formulación Mixta de Elementos Finitos para Problemas de Convección-Difusión

Autor: Iván Padilla Montero

Director: Adeline de Villardi de Montlaur

Supervisores: Jordi Pons Prats, Riccardo Rossi

Fecha: 10 de julio de 2014

Resumen

Este documento tiene como objetivo la solución numérica de problemas de convección-difusión por medio del método de los elementos finitos (FEM) dentro del contexto de la dinámica de fluidos. En él se describe la solución clásica por elementos finitos de problemas de convección-difusión y se presenta la implementación y validación de una nueva formulación para mejorar la precisión del enfoque estándar.
En primer lugar se resalta la importancia y la necesidad de modelos numéricos de convección-difusión en la Dinámica de Fluidos Computacional (CFD), destacando las similitudes entre la ecuación de convección-difusión y las ecuaciones que gobiernan la dinámica de flujos incompresibles.
A continuación se explican los aspectos básicos del método de elementos finitos necesarios para la solución estándar de problemas generales de convección-difusión, y se aplican al caso estacionario. Estos incluyen la formulación débil del problema inicial y de contorno para la ecuación de convección-difusión y su posterior discretización espacial por elementos finitos basada en el método de Galerkin. Después de su aplicación a la ecuación de transporte estacionaria se lleva a cabo una simple prueba numérica para demostrar que la formulación estándar de Galerkin no es estable en situaciones en las que la convección domina, justificando la necesidad de estabilización.
La atención se centra entonces en el análisis del error de truncamiento proporcionado por la formulación de Galerkin, lo que conduce a la obtención de una técnica de estabilización clásica basada en añadir difusión artificial a lo largo de la dirección de flujo, los llamados esquemas contracorriente (SU). Seguidamente se describe un enfoque más general y moderno de la estabilización conocido como el método de multiescalas (SGS), demostrando que las técnicas contracorriente son un caso particular de la misma.
Teniendo en cuenta todos los conceptos explicados, se presenta una nueva formulación de elementos finitos mixtos para problemas de convección-difusión. Ha sido propuesto por el Dr. Riccardo Rossi, investigador del Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE), y consiste en la ampliación de la ecuación de convección-difusión original a un sistema en forma mixta en el que tanto la variable desconocida como su gradiente se calculan de forma simultánea, lo que como se verá lleva a un aumento en la precisión de la solución. La formulación, que no se había probado antes, es entonces implementada y validada por medio de un software multifísico de elementos finitos llamado `Kratos`. Finalmente, se analizan los resultados obtenidos y se muestra el rendimiento mejorado de la formulación mixta en problemas de difusión puros.

CONTENTS

LIST OF FIGURES

INTRODUCTION

Between 1960 and 1980, the birth of high-speed digital computers changed the engineering world. Such a rapid increase in the computing power available made computers to become an essential tool for any engineer or scientist, giving rise to a new philosophy of solving practical problems: the numerical analysis. Acting as a bridge between pure theory and pure experiment, numerical methods allow obtaining answers to problems with complex physical processes that would be impossible to solve by classical analytical methods. In this way, they have revolutionized design and research processes, reducing the amount of experimentation needed.

Due to the technological challenges that it entails, aerospace engineering was one of the fields which took most advantage of numerical methods. In fact, an entirely new discipline in aerodynamics was born, namely, computational fluid dynamics (CFD). With a numerical approach to the solution of the governing equations of fluid dynamics, the application of CFD to practical aerodynamics problems opened the door to the design of complex aerodynamic shapes, capable of producing high amounts of lift very efficiently, without the need of intensive wind tunnel testing.

Since its beginning, CFD has been constantly evolving, playing a major role in the transition to modern aviation. Nowadays the design of any flying vehicle involves a detailed numerical analysis of the flow field, closely capturing the physical phenomena that is taking place. On the other hand, the experimental research barrier that was encountered when trying to simulate high-speed flight regimes in ground facilities has now been almost overcome. With the possibility of obtaining accurate numerical solutions of such flight conditions, the extreme high-speed end of the flight spectrum can be explored.

A good knowledge in CFD is indispensable for the modern aerospace engineer. This is why one of the purposes of this work is to introduce the student to the numerical solution of physical problems in aerospace engineering.

The governing equations of fluid dynamics are complex, and dealing with the full nonlinear system of the equations of continuity, momentum and energy requires an advanced knowledge of the numerical techniques that CFD involves. As a result, given the introductory character of this work, we will focus on the numerical solution of a simplified model, namely, the convection-diffusion equation. At first, it may appear that such an equation has nothing to do with fluid dynamics, however, the physical processes that it describes are also found in the equations of motion of incompressible flows. This, as we will see, is traduced in that some of the numerical difficulties found in the solution of incompressible flow problems are also encountered when solving the convection-diffusion equation, making the numerical approaches of both types of problems very similar. In this sense, the convection-diffusion model can be viewed as a simplified way of exposing and analyzing most of the key features that characterize the numerical solution of incompressible flow problems, acting as an excellent introduction to CFD.

The numerical solution of a given physical problem can be described mainly in three steps. On first place, the problem has to be well defined, that is, the governing equations that describe the physical processes taking place have to be completed with appropriate initial and boundary conditions. Secondly, a spatial and temporal (if the problem is unsteady) discretization has to be performed, thus obtaining a set of discrete points where the gov-

erning equations are evaluated by means of algebraic expressions. Finally, the governing equation at each point has to be solved. In general, there are three different numerical approaches in CFD, namely, the finite difference method (FDM), the finite element method (FEM) and the finite volume method (FVM). The FDM is based on the discretization of the partial derivatives of the governing equations by means of finite difference expressions evaluated in a grid of points covering the problem domain. Such a method is very straightforward to implement for problems with simple geometries that allow using a regular grid. However, for irregular grids, which is usually the case, the difference expressions become difficult to obtain. Moreover the implementation of boundary conditions is strongly problem-dependent and cannot be easily generalized. A very good introduction to CFD by means of the FDM can be found in [1].

In the FEM the domain of study is divided in a mesh of arbitrary volumes or cells called elements. In contrast to the FDM, the FEM uses an integral formulation as a starting point for the discretization, making it naturally appropriate for unstructured (irregular) meshes. Inside each element, the nodal unknown variables are interpolated with particular functions and the resulting discrete equations are assembled in a global system, which is then solved to obtain the field solution. Another great advantage of this method is that it allows a consistent and systematic treatment of the boundary conditions, making it more suitable for the construction of general solvers. This method, although being more mathematically involved than the FDM and less ”student friendly”, offers a more powerful approach to the solution of physical problems. A complete presentation of the fundamentals of the finite element method is to be found in [5] or [10], and its application to CFD in [3], [6] and [23].

Like in the FEM, the FVM assumes a division of the domain of interest in arbitrary cells. However, the FVM considers such cells as control volumes where the governing equations are solved in integral conservation form, that is, expressing the flux balance inside and across the control domains. Once again, the integral form makes it suitable for any arbitrary mesh. Its major drawback is its limitation to conservation laws, and it is interesting to note that the integral formulation of the FEM can be considered a generalization of the FVM. As a consequence, in some particular cases the discrete equations that result from the FVM are identical to the ones from the FEM. For a presentation of the fundamentals of the FVM in CFD see [4].

In this work the attention has been focused in the numerical solution of convection-diffusion problems by means of the finite element method. As presented above, it provides a general and consistent approach for the solution of the desired physical problems and therefore will satisfy the objectives of this study. For this reason, along a considerable part of this document the classical finite element formulation for convection-diffusion problems is described and studied, emphasizing the numerical difficulties that are encountered in convection-dominated problems and the different techniques that have been developed to overcome them. It is important to clarify that the intention is not to provide a formal mathematical presentation of the FEM concepts, but to reflect the numerical and physical aspects of the solution of such problems.

The close relation between the convection-diffusion model and the governing equations of incompressible fluid dynamics not only allows an analogy for reflecting the numerical aspects that characterize certain physical processes, but also provides a solid base for the development and testing of new techniques that could be extrapolated to more complex problems. This defines the main objective of this project, namely, the testing of a new mixed

finite element formulation for convection-diffusion problems developed by Dr. Riccardo Rossi at the International Center for Numerical Methods in Engineering (CIMNE). It was originally presented in [24].

Generally, fluid dynamics problems are characterized by the presence of sharp gradients near the body surfaces. It is in such regions where most of the important physical phenomena takes place, making them of critical importance when performing a numerical solution. As a result, developing numerical techniques that focus on the accurate representation of these processes is a priority. The usual way of improving the numerical solution where strong gradients exist is by increasing the amount of discrete points in that regions of the domain, what is known as mesh refinement. However, the proposal of Dr. Rossi is based on a different idea. It lies on the fact that the convergence rate for the solution of a given variable decreases when it is subject to a differential operator. Hence, it is desirable to reduce the differentiation order as much as possible.

Following this reasoning, Rossi proposes to define an auxiliary variable to generate a mixed finite element formulation and solve convection-diffusion problems with less error than the classical form for the same mesh size. It is important to notice that the numerical difficulties that were present in the classical formulation are still encountered in the mixed form, so the remedies exposed for the conventional form are also used. The rest of this document is devoted to the implementation and testing of such a new formulation. This is done by means of `MATLAB` and `Kratos`, a simulation software developed by CIMNE defined as a `C++` object-oriented environment for the implementation of finite element methods in different physical problems (see [27] and [28] for more information). As we will see, the mixed formulation presents additional difficulties, but it succeeds on improving the accuracy in determinate cases.

This document is divided in four chapters. They are directly related, written in such a way to progressively build-up the complete classical finite element solution of convection-diffusion problems and then expose the new mixed formulation. The first chapter introduces the standard convection-diffusion model used in this work along with the governing equations of fluid dynamics. They are compared to find the several similarities between them and to highlight the important role of the convection-diffusion equation in CFD.

Chapter 2 describes the classical finite element formulation for convection-diffusion problems. After presenting some basic concepts about the FEM, the Galerkin finite element formulation is discussed and its difficulties in convection dominated problems are rapidly recognized, justifying the need for stabilization.

In chapter 3, the stabilized finite element methods for convection dominated transport problems are discussed. An analysis of the truncation error from the Galerkin formulation is performed, leading to the definition of an artificial diffusion that serves as a base for the development of streamline-upwind (SU) stabilization techniques. The discussion on stabilization techniques is completed by introducing the sub-grid scale (SGS) approach, a more general and modern framework for the development of stabilization methods.

Finally, chapter 4 exposes the new mixed finite element formulation. The motivation that led to its development is clarified, and the originally proposed formulation is described. Next, it is implemented and tested, and some numerical deficiencies encountered are commented, leading to a modification of the original formulation. The scope of the modified version is analyzed and found to be restricted to pure diffusion situations. For validation of these arguments, a pure diffusion problem is solved and compared against the results of the

classical formulation, obtaining a good improvement in the accuracy of the calculations.

The socioeconomic and environmental impacts of this work have been analyzed and are not found to be relevant.

CHAPTER 1. IMPORTANCE OF CONVECTION-DIFFUSION MODELS IN COMPUTATIONAL FLUID DYNAMICS

1.1. Governing Equations of Fluid Dynamics for Incompressible Flow

In order to illustrate the role of convection and diffusion mechanisms in CFD, we first need to present which are the equations that govern fluid motion. In fact, these are the equations in which CFD is based.

On first place we would want to clarify that our intention is to describe the equations only, to later be able to compare them with the convection-diffusion model, without entering into the details of their derivation. An excellent and physically meaningful derivation of the governing equations of fluid mechanics is developed by Anderson in [1] and [2].

For our purpose we will focus on incompressible flows, that is, those flows where the density changes are negligible. Such flows are governed by two equations, namely, the continuity and momentum equations. They are mathematical statements of the fundamental physical principles upon which fluid dynamics is based.

1.1.1. The Continuity Equation

The continuity equation is the result of applying the principle of conservation of mass to a fluid volume.

If we consider a finite control volume of fluid fixed in space, with volume V and control surface S, the continuity equation can be written in integral form as

$$\int_S \rho \mathbf{V} \cdot \mathbf{n} \, dS = -\int_V \frac{\partial \rho}{\partial t} \, dV, \tag{1.1}$$

where ρ and $\mathbf{V}$ are the density and velocity of the fluid in the control volume, respectively, and $\mathbf{n}$ is the unit outward normal to the control surface S. Equation (1.1) is stating the principle of mass conservation in a straightforward way. It says that the net mass flow out of the control volume through surface S equals the time rate of decrease of mass inside the control volume V, or, in other words, simply that mass can be neither created nor destroyed.

The continuity equation in differential form can be readily obtained from the integral form by applying the divergence theorem to the left-hand side of equation (1.1) and noting that the resulting integrand, for the equation to be satisfied in any arbitrary control volume, must be identically zero. The resulting expression is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0, \tag{1.2}$$

that expresses the conservation of mass at a given (infinitesimal) point in the flow field, as opposed to equation (1.1), which deals with a finite space.

The continuity equation formulations already presented hold in general, for the three-dimensional, unsteady flow of any type of fluid, inviscid or viscous, compressible or incompressible. However, in this work we are only interested in incompressible flows. To obtain the incompressible continuity equation we have to consider the physical definition of incompressible flow, that is, ρ is constant. Applying this condition to equation (1.2) and recalling the identity $\nabla \cdot (\rho \mathbf{V}) = \rho \nabla \cdot \mathbf{V} + \mathbf{V} \cdot \nabla \rho$, we obtain

$$\nabla \cdot \mathbf{V} = 0 \tag{1.3}$$

which is the continuity equation for incompressible flows. It states that the divergence of velocity is zero in such flows. In other words, it is imposing a condition on the fluid velocity, i.e. that for a flow to be incompressible its velocity field has to be divergence free. Furthermore, it can be shown (see for instance [1] or [2]) that the divergence of velocity is physically the time rate of change of the volume of a moving fluid element, per unit volume. Hence, for an incompressible flow, this quantity is zero and the volume of a moving fluid element per unit volume remains constant in time.

Equation (1.3) is sometimes called the incompressibility condition. It is key for defining a correspondence between the convection-diffusion and momentum equations, which is discussed next.

1.1.2. The Momentum Equation

Another fundamental physical principle that is satisfied by fluid dynamics is Newton's second law. It's application to a fluid volume expresses that the net force acting on the fluid equals the time rate of change of its momentum. The resulting equation is called the momentum equation.

As before, we consider the model of a finite control volume fixed in space. And once again, the best way of exposing the physical phenomena behind the equation is to write it in integral conservation form, that is

$$\int_V \rho \mathbf{f} \, dV - \int_S p\mathbf{n} \, dS + \mathbf{F}_{\text{viscous}} = \int_V \frac{\partial}{\partial t} (\rho \mathbf{V}) \, dV + \int_S \rho \mathbf{V} (\mathbf{V} \cdot \mathbf{n}) \, dS, \tag{1.4}$$

where p is the pressure of the fluid, f is the net body force per unit mass exerted on the fluid inside V (usually due to gravity or electromagnetic forces) and $\mathbf{F}_{\text{viscous}}$ is the term that contains the surface forces due to the shear stress acting on the control surface S. The actual expression for $\mathbf{F}_{\text{viscous}}$ is not needed for our purpose, such a term will only be of our interest in the differential form of the momentum equation, which will be described further below.

Equation (1.4) clearly states what was enunciated above. On the left-hand side we have the net force acting on the fluid in the control volume, which comes from two sources, namely, the body forces acting inside V and the surface forces acting on S, exerted by pressure and shear stress. Then, on the right-hand side we find the time rate of change of momentum of the fluid in the control volume, that is, the net flow of momentum exiting

the control volume across surface S plus the time rate of change of momentum due to the unsteady fluctuations of flow properties inside V. The fluid momentum is given by $\rho\mathbf{V}$, so it is important to notice that equation (1.4) is a vector equation.

Having reflected the physical meaning of the momentum equation, we can now present it in differential form, which is the form that will satisfy our needs. The transition from the integral to the differential form for the momentum equation is not as direct as for the continuity equation. Moreover, there are several ways of expressing it. In the majority of cases, the fluid studied is assumed to be Newtonian, meaning that the shear stress is proportional to the normal velocity gradient, with the constant of proportionality given by the fluid dynamic viscosity μ.

Here we will get directly to the point and present the momentum equation expressed in the useful way for our analysis, that is, the differential momentum equation for incompressible flows. The common way of expressing it is (see for example [3])

$$\frac{\partial \mathbf{V}}{\partial t} + (\mathbf{V}\cdot\nabla)\,\mathbf{V} - \nu\nabla^2\mathbf{V} + \frac{1}{\rho}\nabla p = \mathbf{f}, \tag{1.5}$$

where ν is the kinematic viscosity of the fluid, equal to the dynamic viscosity divided by density, and $\nabla^2\mathbf{V}$ denotes the Laplacian operator applied to the velocity vector, also known as vector Laplacian, which is nothing more than the Laplacian of each scalar component of the velocity vector.

Equation (1.5) is the momentum equation for incompressible flow in differential form, usually known as the incompressible Navier-Stokes equations (actually, there are three equations, one for each velocity component). They are the equations that govern the motion of incompressible flows, always satisfying the incompressibility condition (1.3), which is a crucial step in their derivation. This is the desired expression, as will be made clear in section 1.3..

1.2. The Convection-Diffusion Equation

The convection-diffusion equation is a partial differential equation that expresses the transport of a given field quantity (scalar of vector) by means of two physical processes: convection (also known as advection) and diffusion. In fluid dynamics, the mechanism of convection is governed by the fluid's motion, that is, the properties of the fluid are transported (convected) in space according to the fluid velocity. The diffusion process, on the other hand, is not related to the motion of the fluid and acts even if it is static. It is a transport mechanism that works at a molecular level and is strongly dependent on the properties of the fluid.

1.2.1. General Form of the Convection-Diffusion Equation

In its general form, the convection-diffusion equation can be written as (see [7] and [3])

$$\frac{\partial c}{\partial t} + \nabla\cdot(\mathbf{a}c) - \nabla\cdot(D\nabla c) = s, \tag{1.6}$$

where c is the transported quantity, $\mathbf{a}$ is the velocity of the convected quantity (also known as convection velocity), $D > 0$ is the diffusion coefficient (or diffusivity) and s is a volumetric source term. The second term on the left-hand side of equation (1.6) is known as the convective term, whereas the third one is called the diffusion term. It is important to notice that if anisotropy is present, then the diffusion coefficient becomes directionally dependent. In this study, an isotropic medium will always be considered.

Here the general convection-diffusion equation has been written for the transport of a scalar quantity, however, it is equally valid for a vector quantity without loss of generality. Equation (1.6) has many applications in physics, such as mass transfer, in which the transported quantity becomes the species concentration, or heat transfer, where c represents the temperature. For our objective in this chapter, we are interested in putting it in a fluid dynamics point of view.

1.2.2. The Convection-Diffusion Equation in a Fluid Dynamics Context

Taking a fluid dynamics approach, equation (1.6) becomes more specific. The transported quantity c can be viewed as a scalar flow field property like temperature, making the equation express the heat transfer in the flow. In the same sense, the convection velocity $\mathbf{a}$ is taken to be the velocity of the fluid, better expressed as $\mathbf{V}$. In cartesian coordinates, the components of $\mathbf{V}$ are usually expressed as

$$\mathbf{V} = u\mathbf{i} + v\mathbf{j} + w\mathbf{k}, \tag{1.7}$$

with each one being a function of space and time, i.e. $u = u(x, y, z, t)$, and the same for v and w.

The diffusion coefficient also depends on which quantity is being transported. For example, for temperature T it becomes the thermal conductivity k. The same happens for the volumetric source term s, which for example can be considered as a heat source, taking again the case of temperature transport.

For the sake of illustrating the relation of the convection-diffusion equation with fluid motion, there are some common simplifications that have to be made to the model described by equation (1.6). On one hand we have to consider an incompressible fluid. Applying the incompressibility condition (1.3) we find that the convective term in equation (1.6) becomes (in terms of the fluid velocity)

$$\nabla \cdot (\mathbf{V}c) = \mathbf{V} \cdot \nabla c + c\nabla \cdot \mathbf{V} = \mathbf{V} \cdot \nabla c. \tag{1.8}$$

On the other hand, another simplification that is usually made is that the diffusion coefficient is constant, and hence the diffusion term of equation (1.6) can be expressed by using the Laplacian operator.

Attending to these arguments, the simplified convection-diffusion equation for a scalar quantity reads

$$\frac{\partial c}{\partial t} + \mathbf{V} \cdot \nabla c - D\nabla^2 c = s, \tag{1.9}$$

and for a vector field $\mathbf{c}$

$$\frac{\partial \mathbf{c}}{\partial t} + (\mathbf{V} \cdot \nabla)\,\mathbf{c} - D\nabla^2 \mathbf{c} = \mathbf{s}, \tag{1.10}$$

where now the source term also adopts a vectorial structure.

The form of the convection-diffusion equation given in equations (1.9) and (1.10) is the suitable form for putting it in a fluid dynamics context. For example, to express heat transfer in the fluid we rewrite equation (1.9) in terms of temperature and thermal conductivity, resulting in

$$\frac{\partial T}{\partial t} + \mathbf{V} \cdot \nabla T - k\nabla^2 T = f, \tag{1.11}$$

with f being a heat source.

1.3. Convection-Diffusion as a Representative Model for Computational Fluid Dynamics

On the previous two sections we have focused on the presentation of the governing equations of motion and the convection-diffusion equation for incompressible flows. Now we are ready to expose what do they have in common, and to justify why convection-diffusion models are important for CFD.

Recalling the momentum equation for incompressible flow, that is, equation (1.5) and observing the structure of its terms we can make the following identification

$$\frac{\partial \mathbf{V}}{\partial t} + \underbrace{(\mathbf{V} \cdot \nabla)\,\mathbf{V}}_{\text{Convection}} - \underbrace{\nu\nabla^2 \mathbf{V}}_{\text{Diffusion}} + \frac{1}{\rho}\nabla p = \mathbf{f}. \tag{1.12}$$

As we can see, both convection and diffusion processes are explicitly encountered in the equation. Furthermore, if we directly compare equation (1.5) with the vector convection-diffusion equation for an incompressible fluid, namely, equation (1.10), we find that they are very similar. In fact, a part from the pressure gradient term, equation (1.5) is equation (1.10) with $\mathbf{c}$ equal to the fluid velocity $\mathbf{V}$, D equal to the kinematic viscosity ν and $\mathbf{s}$ to the net body force $\mathbf{f}$. Thus, there is a strong correlation between both equations.

In order to emphasize even more the similitudes, the Navier-Stokes equations for an incompressible flow can also be written as

$$\frac{\partial (\rho \mathbf{V})}{\partial t} + \mathbf{V} \cdot \nabla (\rho \mathbf{V}) - \nu\nabla^2 (\rho \mathbf{V}) + \frac{1}{\rho}\nabla p = \mathbf{f}, \tag{1.13}$$

where the fluid momentum, i.e. $\rho\mathbf{V}$, appears as the transported quantity in the terms of the equation. In this form, we see that the second and third terms on the left-hand side express the convection and diffusion of the fluid's momentum. Actually, we have that for an incompressible Newtonian fluid viscosity operates as a diffusion of momentum.

Now it is clear that convection and diffusion processes play a strong role in fluid dynamics, and the proper understanding of their numerical behavior is of fundamental importance in CFD. As will be seen in the next chapter, important numerical difficulties are encountered in situations where convection dominates. Unfortunately, this is usually the case in fluid dynamics, specially in aerospace engineering. As a result, knowing how to solve such issues is a crucial step for the solution of physical problems through CFD, and the best way to expose them and their remedies is by bringing them to its simplest form, that is, to the convection-diffusion equation.

A good proof of the importance of convection-diffusion models in CFD is that several numerical techniques that are used nowadays for improving the solutions of fluid dynamics problems have their foundation in the convection-diffusion equation. When a numerical difficulty associated to one of the physical processes is encountered, such as that for convection, the convection-diffusion equation allows the source of the problem to be exposed, and provides information of how it should be circumvented. Then, if a suitable solution is found, it might be extrapolated to more complex physical models that include the process of convection, such as the case of incompressible flow. In this sense, the convection-diffusion equation can also be viewed as a ”testing bench” for the development of numerical techniques aimed to improve the solution of more complex problems.

The arguments given in this section serve to justify why the convection-diffusion equation, although being simpler, is a representative model problem for CFD.

In this text, the convection-diffusion equation for heat transfer, equation (1.11), will be used. It is simpler than equation (1.10), which is the one that mimics the convection and diffusion processes of the momentum equation, but is enough for showing the important numerical aspects of convection-diffusion problems. The main difference is that the heat transfer equation is scalar and linear, whereas the convective transport of momentum introduces a non-linear term (see equation (1.13)). Such a non-linearity can produce discontinuities in the solution. However, the analysis of this phenomenon is out of the scope of this work, see [3] for the details.

As a last thing, we would want to comment that the presence of the pressure gradient term in equations (1.5) and (1.13) introduces some additional restrictions in the numerical solution, but it does not affect the validity of the numerical techniques exposed in along this work.

CHAPTER 2. THE FINITE ELEMENT METHOD IN CONVECTION-DIFFUSION PROBLEMS

2.1. Statement of the Initial Boundary Value Problem

The first step towards the theoretical solution of a given physical problem, both by analytical or numerical methods, is its correct mathematical formulation. Once the governing equations that model the behavior of the physical system have been obtained, proper initial and boundary conditions must be prescribed in order to close problem. Otherwise its solution would be undefined. While boundary conditions are always needed for any problem to be well-posed, initial conditions are only required in transient problems where the time dimension is important.

In our analysis, two types of boundary conditions are used, namely, the Dirichlet and Neumann boundary conditions. Dirichlet conditions are defined as the direct prescription of the unknown values of the problem at the system boundary. Neumann conditions are the imposition of the normal gradient of the unknown function along the boundary.

Focusing on convection-diffusion problems, our model is given by the partial differential equation (1.11). We consider the transport by convection and diffusion of a scalar quantity, in this case temperature, that is a function of both space and time $T = T(x,y,z,t)$, in a domain denoted by Ω with a smooth boundary Γ. The boundary is divided in two complementary portions, Γ_D and Γ_N such that $\Gamma = \Gamma_D \cup \Gamma_N$, respectively referring to that portions of the boundary where Dirichlet and Neumann conditions are prescribed.

With this, the initial and boundary value problem associated with equation (1.11) can be stated as: given the velocity field $\mathbf{V}(x,y,z,t)$, the diffusion coefficient k, the source term $f(x,y,z,t)$, and the necessary initial and boundary conditions, find $T(x,y,z,t)$ in a given time interval provided that

$$\frac{\partial T}{\partial t} + \mathbf{V}\cdot\nabla T - k\nabla^2 T = f \qquad \text{in } \Omega, \tag{2.1a}$$

$$T(x,y,z,0) = T_0(x,y,z) \qquad \text{in } \Omega, \tag{2.1b}$$

$$T = T_D \qquad \text{on } \Gamma_D, \tag{2.1c}$$

$$\mathbf{n}\cdot(k\nabla T) = k\frac{\partial T}{\partial \mathbf{n}} = q_n \qquad \text{on } \Gamma_N, \tag{2.1d}$$

where T_0 refers to the initial conditions, T_D denotes the prescribed values of T on the Dirichlet portion of the boundary, and the function q_n denotes the prescribed normal diffusive flux on the Neumann portion, with $\mathbf{n}$ being the normal vector to Γ_N.

2.2. Weak Form of the Problem

The statement of the initial boundary value problem given by (2.1) in the previous section is known as the strong form. However, the process of numerical solution by the finite element

method rests upon the discrete representation of a weak integral from of the governing partial differential equation. Then, the first step for the finite element solution of convection-diffusion problems is the association of an equivalent weak form to the strong form of the initial boundary value problem. This can be achieved by two different approaches, namely, variational formulations or residual formulations.

Variational formulations are based on finding the solution of the problem through an integral equation that represents a general property of the system, like for example the principle of minimum energy. On the other hand, residual formulations are characterized by the weighting of the residual of the governing equation by means of specific functions, without requiring any physical property to be formulated. Due to their more general applicability, residual formulations are the most usual, and CFD is no exception.

2.2.1. The Weighted Residual Formulation

For obtaining the weak formulation of our model problem we use the standard approach in the FEM, that is, the weighted residual formulation. Basically, it consists on multiplying the governing equation by a weighting (also called test) function and integrating it over the computational domain Ω. The key point in this process is to ensure that the solution of the weak integral form is also a solution of the strong form of the problem. This imposes some requirements on the choice of the weighting and the unknown functions (in this case the temperature function), with specific continuity requirements that classify them in determinate mathematical spaces.

Here we will assume the classical definition of the weighting and unknown functions, that is, those functions which are square integrable and have square integrable first derivatives over the domain Ω. Moreover, on the Dirichlet portion of the boundary, Γ_D, the weighting functions are assumed to vanish and the unknown functions are assumed to be equal to their prescribed values (T_D in our case). A formal mathematical presentation of this concepts is given in [3] and [5].

Denoting the weighting function by w, the weighted residual formulation of problem (2.1) is

$$\int_\Omega \mathrm{w}\frac{\partial T}{\partial t}\,d\Omega + \int_\Omega \mathrm{w}\,(\mathbf{V}\cdot\nabla T)\,d\Omega - \int_\Omega \mathrm{w}k\nabla^2 T\,d\Omega = \int_\Omega \mathrm{w}f\,d\Omega, \tag{2.2}$$

where, as described above, $\mathrm{w} = 0$ and $T = T_D$ on Γ_D. By observing equation (2.2) we see that classical solutions of problem (2.1) will also verify this integral equation for all the admissible functions w, i.e. both formulations are equivalent.

The weak form (2.2) accounts for initial and Dirichlet conditions through T_0 and T_D, respectively. However, it does not tell anything about Neumann boundary conditions. In order to account for Neumann conditions also, the diffusion term on the left-hand side of equation (2.2) is integrated by parts, becoming

$$\int_\Omega \mathrm{w}k\nabla^2 T\,d\Omega = -\int_\Omega \nabla\mathrm{w}\cdot(k\nabla T)\,d\Omega + \int_\Gamma \mathrm{w}\mathbf{n}\cdot(k\nabla T)\,d\Gamma. \tag{2.3}$$

Now, if we recall that the weighting function is zero on Γ_D by definition, the resulting boundary integral that appears in equation (2.3) is the weighted formulation of the condition expressed in (2.1d), that is, the Neumann boundary condition. As can be observed, such

term has appeared naturally from a mathematical manipulation of the diffusion term. This is why Neumann conditions are also called natural boundary conditions. Now, the weak formulation of our convection-diffusion problem renders

$$\int_\Omega \mathrm{w}\frac{\partial T}{\partial t}\,d\Omega + \int_\Omega \mathrm{w}\,(\mathbf{V}\cdot\nabla T)\,d\Omega + \int_\Omega \nabla \mathrm{w}\cdot(k\nabla T)\,d\Omega = \int_\Omega \mathrm{w} f\,d\Omega + \int_{\Gamma_N} \mathrm{w} q_n\,d\Gamma. \quad (2.4)$$

This weak form is at the basis of the finite element solution of our convection-diffusion problem. The next step is the time and space discretization, necessary for obtaining a numerical solution of the problem.

2.3. Basics of the Finite Element Spatial Discretization

The process of spatial discretization by the finite element method considers a partition of the computational domain Ω into a given number of non-intersecting subdomains called elements (or finite elements). Each subdomain is denoted by Ω_e and has a piecewise smooth boundary Γ_e. The shape of the elements can be arbitrary, however, to ensure a consistent numerical solution their distribution (the mesh) has to be as regular as possible. In one dimension there is only one possible shape, i.e. a straight line, but for multidimensional problems, which is usually the case in practice, different shapes can be used. The most common are the triangular and the quadrilateral elements, for being the simplest two-dimensional geometries.

In our presentation we will focus on two-dimensional convection-diffusion problems using standard triangular elements. Such elements are represented in figure 2.1. Our intention is only to present the finite element formulation that we use for the resolution of the considered convection-diffusion model. General finite element concepts can be found for example in [5] and [10].

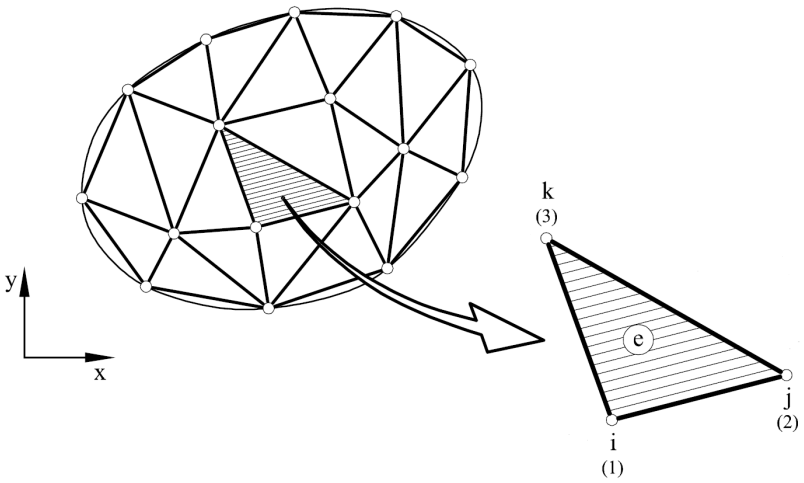


Figure 2.1: A generic domain divided into triangular finite elements. Adapted from [8].

2.3.1. Approximation of the Unknown

It is clear that due to the impossibility of finding exact analytical solutions to the problems under study, what the finite element method finds is an approximate solution. Such an approximation, attending to our model, is denoted by T^h, with $T^h \approx T$ (see [10]). The superscript h is the characteristic mesh size, and consequently gives the precision of the approximation. The weighting functions now also become approximations, expressed as w^h.

This approximation process does not alter any of the function requirements exposed in section 2.2.1., therefore, w^h also vanish on Γ, and T_D is satisfied on the Dirichlet boundary with the precision given by h. Now the weak form of the problem is restricted to the finite element space, and can be written as

$$\int_\Omega \mathrm{w}^h \frac{\partial T^h}{\partial t}\, d\Omega + \int_\Omega \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T^h\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}^h \cdot \left(k\nabla T^h\right)\, d\Omega = \int_\Omega \mathrm{w}^h f\, d\Omega + \int_{\Gamma_N} \mathrm{w}^h q_n\, d\Gamma. \quad (2.5)$$

In the practical implementation of the finite element method, attention is focused on the computations in an individual element. Taking the definition of the integral in a distributional sense, an integral over the complete domain Ω can be viewed as the sum of the integrals over each subdomain Ω_e, that is, over each element. In a similar fashion, the approximation of the unknown can be interpreted as the topological assembly of the individual element contributions, namely, T_e^h. This allows us to write equation (2.5) for a given element as

$$\int_{\Omega_e} \mathrm{w}^h \frac{\partial T_e^h}{\partial t}\, d\Omega + \int_{\Omega_e} \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T_e^h\right)\, d\Omega + \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \left(k\nabla T_e^h\right)\, d\Omega = \int_{\Omega_e} \mathrm{w}^h f\, d\Omega + \int_{\Gamma_N^e} \mathrm{w}^h q_n^e\, d\Gamma, \quad (2.6)$$

with q_n^e being the normal prescribed flux on the portion of the complete Neumann boundary that corresponds to the given element. Obviously, if the element is in the interior of the domain this term becomes zero. It should also be noted that the values of $\mathbf{V}$ and f appearing in (2.6) are the corresponding local values acting on the element domain Ω_e.

Given the previous elementwise definition of the approximated weak form, the problem unknown is interpolated inside each element by means of a linear combination of functions such that, for two dimensions

$$T_e^h(x,y) = \sum_{i=1}^{n} N_i^e(x,y)\, T_i^e, \quad (2.7)$$

where T_i^e is the value of T_e^h at node number i, that is, the nodal unknown, n is the number of nodes of the element and N_i^e is the element shape function of node i, described next.

The shape functions are expressions that interpolate the unknown inside the element according to its nodal values, and they do only depend on the geometry of the element shape chosen. In our case we choose to use triangular elements with three nodes ($n = 3$), also known as linear triangular elements due to the linear order of their shape functions. They are the simplest bidimensional elements, but also the most used.

The order of the shape functions depends on the number of nodes that is used in an element. As a result, several types of elements can be defined with their respective quadratic, cubic, quartic and so on shape functions (see [8] for the details). For linear triangular elements they are defined by

$$N_i^e = \frac{1}{2A_e}\left(a_i^e + b_i^e\, x + c_i^e\, y\right), \tag{2.8}$$

where

$$a_i^e = x_j^e\, y_k^e - x_k^e\, y_j^e, \tag{2.9a}$$
$$b_i^e = x_j^e - x_k^e, \tag{2.9b}$$
$$c_i^e = y_k^e - y_j^e, \tag{2.9c}$$

with A_e being the element area and the subindices i, j, k varying from 1 to 3 (see figure 2.1), depending on the node for which the shape function is being calculated. Looking at equation (2.8), we observe that the shape functions for linear triangular elements are first order polynomials whose coefficients are a function of the nodal coordinates, and thus they are linear. Moreover, it is easy to check that such functions take a value of one in their node and zero on the other two. This is illustrated in figure 2.2, where we see that the shape functions can be viewed as a triangular plane. Hence, if we want the value of the unknown at any point inside the element we only need to substitute for the x and y coordinates of the point in the expression (2.7). This approach provides a local algebraic model that is very powerful for the numerical solution of the problem, as will be made clear later.

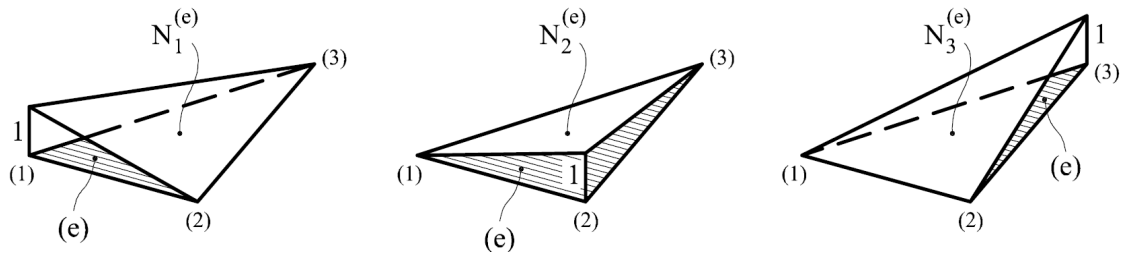


Figure 2.2: Shape functions for the three-noded triangular element. From [8].

With the shape functions already defined, the interpolation (2.7) can be substituted into the elemental weak form. It important to notice, though, that there are differential operators applied to the unknown T_e^h in equation (2.6). In fact, in all the terms where temperature is present. Ignoring the transient term, which will be treated in section 2.6., we see that we have a gradient operator applied to both the convection and diffusion terms. Recalling that T_i^e are the nodal unknowns of the element, which are treated as constants, we find

$$\nabla T_e^h = \sum_{i=1}^{n} \nabla N_i^e T_i^e. \tag{2.10}$$

Thus, the gradient term operates on the shape functions. This is an important feature of the finite element method, and highlights one of its most important differences against the finite difference method, which is that the derivatives of the governing equation, instead of being replaced by finite difference quotients, are algebraically computed through the shape functions. Looking back to expression (2.8), the shape function derivatives are immediately computed to obtain

$$\frac{\partial N_i^e}{\partial x} = \frac{b_i^e}{2A_e} \qquad \text{and} \qquad \frac{\partial N_i^e}{\partial y} = \frac{c_i^e}{2A_e}. \tag{2.11}$$

Due to their linearity, the shape functions gradient is constant all over the element domain. It is obvious that if second derivatives were present in the weak formulation (2.6), such terms would vanish when performing the discretization through linear triangular elements, making the solution to become incorrect. In this case, although the original weak form, i.e. equation (2.2), contains a second-order term (the diffusion term), the process of integration by parts reduces by one the order of the differential operator, making the use of linear elements suitable for the convection-diffusion problem. For this reason, the purpose of the integration by parts is often not only that of introducing the natural boundary conditions, but also that of adequately modifying the continuity requirements of those terms that are subject to non-linear spatial operators. Moreover, this also reflects the importance of the shape functions in the finite element method. The type of element that is used for the spatial discretization shall be carefully chosen, ensuring that it has enough regularity to produce a correct solution of the problem.

2.3.2. The Galerkin Method

Now that the unknown T_e^h is properly approximated for our convection-diffusion model, the last step that remains for completing the finite element spatial discretization of problem (2.1) is the choice of the weighting functions. Almost any set of independent functions could be used for the purpose of weighting and, according to the choice, a different name can be attached to each process. The various common choices include the point collocation method, the subdomain collocation method, the Galerkin method and the least squares method. A description of them can be found in [5] and in [8].

For being the most used in CFD, and in a general context of the FEM, we will focus on the Galerkin method. It simply consists on considering as weighting functions the same functions that are used for interpolating the unknown inside the element, that is, the shape functions. As a result, for a given element node we will have $\mathrm{w}_i^e = N_i^e$, where the superscript h has been dropped in order not to overload the notation.

With this approach, a different weighting function is selected for each node, producing three different equations for a given element. Then, we end up with three equations of the type (2.6), with three unknowns each, namely, T_i^e. When performing the assembly of each individual element contribution, this generates a global system of equations that can be

solved to give the numerical solution to the problem. This process will be made clearer in the following section.

2.4. Discretization of the Steady Transport Problem

With the purpose of illustrating the spatial finite element discretization process for the solution of convection-diffusion transport problems, we will start by considering the steady convection-diffusion equation, which can be derived from equation (1.11) by taking the time derivative equal to zero, obtaining

$$\mathbf{V} \cdot \nabla T - k\nabla^2 T = f. \tag{2.12}$$

This assumption has no impact on the weak formulation of the problem other than removing the time derivative term. As a result, the finite element approximated weak form (2.6) for a given element now reads

$$\int_{\Omega_e} \mathrm{w}^h \left(\mathbf{V} \cdot \nabla T_e^h\right)\, d\Omega + \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \left(k \nabla T_e^h\right)\, d\Omega = \int_{\Omega_e} \mathrm{w}^h f\, d\Omega + \int_{\Gamma_N^e} \mathrm{w}^h q_n^e\, d\Gamma. \tag{2.13}$$

At this point, we have all the necessary ingredients to perform the spatial discretization of the problem. As stated before, we make use of the Galerkin finite element formulation. Taking into consideration the concepts exposed in the previous section, the discrete weak form for the node i of a given element e can be expressed as

$$\int_{\Omega_e} N_i^e \left[\mathbf{V} \cdot \left(\sum_{j=1}^{n} \nabla N_j^e T_j^e\right)\right]\, d\Omega + \int_{\Omega_e} \nabla N_i^e \cdot \left[k \left(\sum_{j=1}^{n} \nabla N_j^e T_j^e\right)\right]\, d\Omega = \int_{\Omega_e} N_i^e f\, d\Omega + \int_{\Gamma_N^e} N_i^e q_n^e\, d\Gamma, \tag{2.14}$$

where both indexes i and j go from 1 to 3. Hence, given a linear triangular element, we have three equations like (2.14), one for each node. Thus, i selects the node for which the equation is being discretized. On the other hand, j gives the interpolation of the unknown, which is the same for the three resulting equations.

Focusing the attention on the integrals of the weak formulation (2.14), we see that although the relative simplicity of linear elements, their analytical calculation can become tedious. In fact, for higher-order elements it becomes intractable. For this reason, in a practical implementation of the finite element method such integrals are evaluated numerically. For our problems, we use a technique known as the Gauss quadrature. However, details will not be provided here. If desired, they can be found for example in [5].

2.4.1. Matrix Form of the Discrete Equation

The indexing used in equation (2.14) can become rather confusing or uncomfortable. It is much more convenient to express the discretized weak form in a matrix structure.

For linear triangular elements, the following matrices are usually defined

$$\mathbf{N}^e = [N_1^e, N_2^e, N_3^e] \tag{2.15}$$

$$\mathbf{T}^e = [T_1^e, T_2^e, T_3^e]^T, \tag{2.16}$$

where the subscripts 1, 2 and 3 denote the corresponding element node and the superscript T denotes the matrix transpose. Recalling matrix algebra, these expressions allow us to rewrite the unknown approximation in the better way

$$T_e^h = [N_1^e, N_2^e, N_3^e] \begin{Bmatrix} T_1^e \\ T_2^e \\ T_3^e \end{Bmatrix} = \mathbf{N}^e \mathbf{T}^e. \tag{2.17}$$

In a similar fashion, the shape functions gradient takes the form

$$\nabla \mathbf{N}^e = [\nabla N_1^e, \nabla N_2^e, \nabla N_3^e] = \begin{bmatrix} \dfrac{\partial N_1^e}{\partial x} & \dfrac{\partial N_2^e}{\partial x} & \dfrac{\partial N_3^e}{\partial x} \\ \dfrac{\partial N_1^e}{\partial y} & \dfrac{\partial N_2^e}{\partial y} & \dfrac{\partial N_3^e}{\partial y} \end{bmatrix}. \tag{2.18}$$

With these definitions, the discrete weak form for our steady transport problem can be cast in matrix form, resulting in the following expression

$$(\mathbf{C}^e + \mathbf{K}^e)\, \mathbf{T}^e = \mathbf{f}^e, \tag{2.19}$$

where $\mathbf{C}^e$ and $\mathbf{K}^e$ are known, respectively, as the element convection and diffusion matrices and $\mathbf{f}^e$ is simply called the right-hand side vector. They have the form

$$\mathbf{C}^e = \int_{\Omega_e} [\mathbf{N}^e]^T \, \mathbf{V} \nabla \mathbf{N}^e \, d\Omega \tag{2.20}$$

$$\mathbf{K}^e = \int_{\Omega_e} [\nabla \mathbf{N}^e]^T \, k \nabla \mathbf{N}^e \, d\Omega \tag{2.21}$$

$$\mathbf{f}^e = \int_{\Omega_e} [\mathbf{N}^e]^T \, f \, d\Omega + \int_{\Gamma_N^e} [\mathbf{N}^e]^T \, q_n^e \, d\Gamma. \tag{2.22}$$

As noted in the previous section, the convection velocity $\mathbf{V}$ and the source term f appearing respectively in equations (2.20) and (2.22) are considered to be the element local values. They result from the interpolation of the element nodal values, in the same way as was done for the unknown function, that is

$$\mathbf{V} = [N_1^e, N_2^e, N_3^e] \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \end{bmatrix} \qquad \text{and} \qquad f = [N_1^e, N_2^e, N_3^e] \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \end{Bmatrix}, \tag{2.23}$$

By direct observation, we see that $\mathbf{C}^e$ and $\mathbf{K}^e$ are 3×3 matrices and $\mathbf{f}^e$ is a 3×1 vector. Therefore (2.19) is a system of three equations with three unknowns.

In order to account for the Dirichlet boundary conditions, if any of the three element nodes is found to be prescribed, its contribution is substracted from the right-hand side of the system. Defining the element Dirichlet values vector by $\mathbf{T}_D^e = \left[T_{D1}^e, T_{D2}^e, T_{D3}^e\right]^T$ the complete system becomes

$$(\mathbf{C}^e + \mathbf{K}^e)\,\mathbf{T}^e = \mathbf{f}^e - (\mathbf{C}^e + \mathbf{K}^e)\,\mathbf{T}_D^e. \tag{2.24}$$

Equation (2.24) represents the element contribution to the overall problem. When all the individual element contributions are assembled, the global system of equations is obtained, allowing us to express the solution of the problem as

$$(\mathbf{C} + \mathbf{K})\,\mathbf{T} = \mathbf{f}, \tag{2.25}$$

where, by introducing the assembly operator $\mathbf{A}^e$

$$\mathbf{C} = \mathbf{A}^e \mathbf{C}^e \qquad \mathbf{K} = \mathbf{A}^e \mathbf{K}^e \qquad \mathbf{f} = \mathbf{A}^e \mathbf{f}^e. \tag{2.26}$$

The solution of system (2.25) delivers the nodal values of the discrete solution of the convection-diffusion problem. It has as many equations as nodal unknowns are in the finite element mesh, and its solution for a linear problem like the one considered in this work can be obtained directly by inverting the left-hand side matrix.

2.5. The Need for Stabilization

With the steady transport problem already discretized, we can now start with the presentation of some finite element solutions. Although our main focus are two-dimensional problems, there are some numerical difficulties that have to be previously addressed, and the best way to expose them is by the solution of unidimensional convection-diffusion problems.

The Galerkin finite element method presents some numerical deficiencies in the solution of convection-dominated problems. To illustrate them, a simplified numerical example is performed, usually consisting on the solution of the one-dimensional steady convection-diffusion equation (see [11] or [3]), that is

$$u\frac{\partial T}{\partial x} - k\frac{\partial^2 T}{\partial x^2} = f. \tag{2.27}$$

In order to characterize the relative importance between convection and diffusion processes in a given flow problem, the following dimensionless number is usually defined

$$Pe = \frac{Vh}{2k}, \tag{2.28}$$

known as the mesh Péclet number, which expresses the ratio of convective to diffusive transport, with V being the average value of the convection velocity norm ($V = u$ in the 1D

case) and h a characteristic mesh size, such as the element length for 1D problems. This number can be considered the convection-diffusion analogue to the Reynolds number, that expresses the ratio of inertia to viscous forces in fluid dynamics, once again reflecting the similitudes between both physical models.

The problem considers solving Equation (2.27) in a domain of length $L = 1$ (dimensionless domain) discretized with a uniform mesh of 10 linear elements, with uniform unit convection velocity and source term, i.e. $u = 1$ and $f = 1$, and with homogeneous Dirichlet boundary conditions imposed at each side, namely, $T = 0$ at $x = 0$ and $x = L$. The value of the diffusion coefficient k is obtained by fixing a value for the Péclet number. Uniform conditions are chosen in order to avoid truncation errors due to the spatial discretization of the velocity and source terms, as given in (2.23), ensuring that the source of the error is purely due to the discrete representation of the convection and diffusion operators produced by the Galerkin finite element formulation.

The exact solution for this model problem with constant convection and diffusion coefficients u and k is not difficult to obtain and is given by

$$T(x) = \frac{1}{u}\left[x - \frac{1-\exp\left(\frac{u}{k}x\right)}{1-\exp\left(\frac{u}{k}\right)}\right]. \tag{2.29}$$

The finite element discretization of this problem follows the same steps described in the previous sections, except that now unidimensional elements have to used. Linear elements in 1D are defined by two nodes, as can be seen in Figure 2.3, where a portion of the problem mesh is shown. Their shape functions are commonly expressed as

$$N_i^e(\xi) = \frac{1}{2}(1+\xi_i\,\xi), \tag{2.30}$$

where ξ is simply the normalized coordinate $\xi = \frac{2(x-x_c)}{h}$, with x_c being the abscissa of the center of the element, and ξ_i its evaluation at the corresponding element node i. Denoting the local left and right nodes of the element by $i = 1$ and $i = 2$, respectively, one finds that $\xi_1 = -1$ and $\xi_2 = 1$, thus giving

$$N_1^e(\xi) = \frac{1}{2}(1-\xi) \qquad \text{and} \qquad N_2^e(\xi) = \frac{1}{2}(1+\xi). \tag{2.31}$$

Similarly, their derivatives can be easily obtained noting that

$$\frac{\partial N_i^e}{\partial x} = \frac{\partial N_i^e}{\partial \xi}\frac{\partial \xi}{\partial x} = \frac{2}{h}\frac{\partial N_i^e}{\partial \xi}, \tag{2.32}$$

so

$$\frac{\partial N_1^e}{\partial x} = -\frac{1}{h} \qquad \text{and} \qquad \frac{\partial N_2^e}{\partial x} = \frac{1}{h}. \tag{2.33}$$

The main reason for using the coordinate transformation given by ξ is to normalize the shape of the element in a transformed space that allows the expression of the shape functions in a more convenient form. Although for linear 1D elements it does not suppose

a noticeable advantage, it becomes very useful for multidimensional elements and its use is standard.

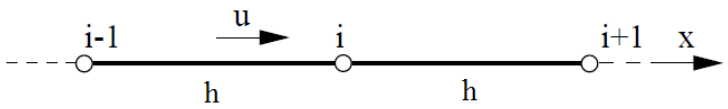


Figure 2.3: Portion of the finite element mesh used for solving problem (2.27). Adapted from [9].

With this definitions, the discretized weak form of the problem can be readily obtained. Recalling the weak formulation of the steady convection-diffusion transport problem for two dimensions, that is, equation (2.19), we see that now the convection and diffusion matrices have a size of 2×2 each, and the right hand side vector of 2×1. Noting that for the one-dimensional case

$$\mathbf{N}^e = [N_1^e, N_2^e] \qquad \text{and} \qquad \nabla \mathbf{N}^e = \begin{bmatrix} \dfrac{\partial N_1^e}{\partial x} & \dfrac{\partial N_2^e}{\partial x} \end{bmatrix} \tag{2.34}$$

and that u, k and f are constant, the element convection and diffusion matrices and the element right-hand side vector become

$$\mathbf{C}^e = u \int_{\Omega_e} \begin{bmatrix} N_1^e \dfrac{\partial N_1^e}{\partial x} & N_1^e \dfrac{\partial N_2^e}{\partial x} \\ N_2^e \dfrac{\partial N_1^e}{\partial x} & N_2^e \dfrac{\partial N_2^e}{\partial x} \end{bmatrix} dx = \frac{u}{2} \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \tag{2.35}$$

$$\mathbf{K}^e = k \int_{\Omega_e} \begin{bmatrix} \dfrac{\partial N_1^e}{\partial x}\dfrac{\partial N_1^e}{\partial x} & \dfrac{\partial N_1^e}{\partial x}\dfrac{\partial N_2^e}{\partial x} \\ \dfrac{\partial N_2^e}{\partial x}\dfrac{\partial N_1^e}{\partial x} & \dfrac{\partial N_2^e}{\partial x}\dfrac{\partial N_2^e}{\partial x} \end{bmatrix} dx = \frac{k}{h} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.36}$$

$$\mathbf{f}^e = f \int_{\Omega_e} \begin{Bmatrix} N_1^e \\ N_2^e \end{Bmatrix} dx = \frac{hf}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}, \tag{2.37}$$

where in this case the solution of the integrals has been obtained analytically, with N_1^e and N_2^e being the shape functions in cartesian coordinates, i.e. expressed as a function of x.

Due to the boundary conditions chosen, the Dirichlet term is zero in this case and the elemental system of equations has the form (2.19), with $\mathbf{T}^e = \left[T_1^e, T_2^e\right]^T$ and with $\mathbf{C}^e$, $\mathbf{K}^e$ and $\mathbf{f}^e$ given by the expressions (2.35) to (2.37). As explained before, the assembly of these individual element contributions already obtained produces the global system of equations, in the form (2.25), that gives the solution to the problem.

In order to achieve the objective of this section, that is, exposing the numerical deficiencies of the Galerkin method in convection-dominated problems, equation (2.27) is solved for two different values of the Péclet number, namely, $Pe = 0.5$ and $Pe = 5$. The numerical results are shown in figure 2.4, compared against the exact solution (2.29).

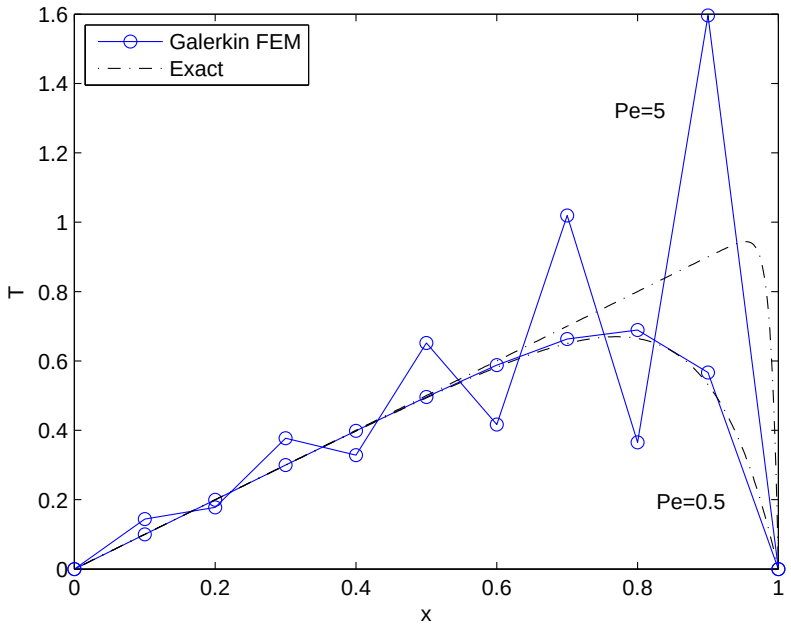


Figure 2.4: Galerkin finite element solution of the 1D steady transport problem with $u = 1$ and $f = 1$ for two different Péclet numbers. The exact solution is also shown for comparison.

As can be clearly seen in the figure, the Galerkin solution is satisfactory when the Péclet number is 0.5, providing an accurate numerical solution to the problem. However, for the case of $Pe = 5$, it presents strong non-physical node-to-node oscillations that make the solution to become unstable and fail. In fact, for values of the Péclet number larger than 1, that is, when convection dominates, the solution obtained by means of the Galerkin finite element method is corrupted by spurious oscillations, also known in CFD as ”wiggles”, that prevent the numerical solution from converging to correct results. Moreover, it can be easily checked the stronger the convective effects, the larger are the oscillations produced.

This deficiency of the Galerkin formulation has been widely studied since the application of the finite element method to fluid dynamics problems. As described by Brooks and Hughes [11], one of the greatest advantages of the Galerkin method, and one of the main reasons of its popularity, is that when applied to most structure or heat conduction problems it leads to symmetric matrices, a situation where the finite element method produces the best results. This created an optimistic point of view and the hope that such significant advantages would again be open to exploitation in the area of fluid flow simulation. Nevertheless, in problems where convection is present, the Galerkin discretization of the convection operator produces a non-symmetric matrix (see equation (2.35)), causing the loss of such ”best approximation” property. As a result, when the convection operator dominates the diffusion operator in the transport equation the oscillations appear. It is important to notice, however, that wiggles are most likely to appear in convection dominated cases where a downstream boundary condition is present, forcing a rapid change (large gradient) in the solution. Such a boundary condition can be observed in figure 2.4, and is the most usual case in practical convection-diffusion and flow problems.

The only way to eliminate the oscillations while not changing the formulation is by a severe

mesh refinement, so that the convection operator no longer dominates on an element level (note in equation (2.36) that the diffusion matrix is inversely proportional to the characteristic element size h). Unfortunately, a global refinement becomes impractical in multidimensional problems, and most of the times detailed features of the solution are not desired in all the domain, only in specific regions. Hence, there is no other choice than that of improving the formulation. This motivated the development of numerical methods that try to eliminate the oscillations while maintaining the accuracy of the solution. Such improved formulations, that can be viewed as a means of stabilizing the solution of the problem, are known as stabilization techniques.

In conclusion, this section serves to justify the fact that stabilization is a necessary condition for the solution of many convection-diffusion problems, and, of course, for the majority of CFD applications. Due to their fundamental importance, in the next chapter the most common stabilization techniques will be described.

2.6. Time Discretization

Along this chapter, we have been dealing only with the finite element solution to the steady convection-diffusion equation. The reason for this, a part from simplifying the presentation of the finite element method in convection-diffusion problems, is that very often the time discretization of unsteady transport problems is not performed through the finite element method but by means of a finite difference approach. Actually, the common philosophy when solving unsteady convection-diffusion problems lies on performing on first place the time discretization of the transient term using a finite difference technique, thus obtaining a modified governing partial differential equation that only contains spatial differential operators, and then applying the standard finite element spatial discretization as exposed in section 2.3. to that time-discretized form of the problem. It has to be said, however, that the finite element time discretization can also be used, leading to the so-called space-time formulations, but will not be considered here.

There are several finite difference formulations suitable for the time discretization, allowing different orders of accuracy to be obtained (refer to [3, 5] for their presentation). In this work, we have focused primarily on the solution of steady problems, and for the unsteady cases, the use of second-order methods has found to be sufficient. Among the second-order algorithms, we have chosen to use the Crank-Nicolson method for being simple in its implementation and because it is the one that has the smallest truncation error.

The Crank-Nicolson algorithm is an implicit single step method, meaning that the value T^{n+1} of the problem unknown at time $t^{n+1} = t^n + \Delta t$ is computed from the value T^n at time t, with Δt denoting the considered time step. In this particular case this is achieved by defining the time derivative as a first-order forward finite difference (see [1] for details) given by

$$\frac{T^{n+1} - T^n}{\Delta t} = \frac{1}{2}\left[\left.\frac{\partial T}{\partial t}\right|_{t=t^{n+1}} + \left.\frac{\partial T}{\partial t}\right|_{t=t^n} \right], \tag{2.38}$$

that is, expressing it as an average between the time derivatives at times t^{n+1} and t^n, which in turn are replaced by using the governing partial differential equation, namely, equation

(1.11), as

$$\left.\frac{\partial T}{\partial t}\right|_{t=t^{n+1}} = f^{n+1} - \mathbf{V}\cdot\nabla T^{n+1} + k\nabla^2 T^{n+1}, \tag{2.39}$$

and the analogue for t^n

$$\left.\frac{\partial T}{\partial t}\right|_{t=t^{n}} = f^{n} - \mathbf{V}\cdot\nabla T^{n} + k\nabla^2 T^{n}. \tag{2.40}$$

Introducing the incremental unknown $\Delta T = T^{n+1} - T^n$ and substituting equations (2.39) and (2.40) into (2.38) we obtain the following form for the Crank-Nicolson time discretization of equation (1.11)

$$\frac{\Delta T}{\Delta t} + \frac{1}{2}\left(\mathbf{V}\cdot\nabla - k\nabla^2\right)\Delta T = \frac{1}{2}\left(f^{n+1} + f^n\right) - \left(\mathbf{V}\cdot\nabla - k\nabla^2\right)T^n. \tag{2.41}$$

Looking at equation (2.41) we see that all the differential operators present are spatial operators. It can be viewed as the new governing equation for the unsteady convection-diffusion problem, and since now the time derivative is not present, the conventional finite element spatial discretization can be applied to obtain the numerical solution at each time step. In other words, although the time discretized equation also has a temporal truncation error, for small time steps such error can be neglected, allowing the equation to be interpreted as a spatial differential operator in strong form that must be solved at each step.

To finish with, it also has to be said in favor of the Crank-Nicolson scheme that it is an unconditionally stable algorithm, that is, it will converge for any value of the time step used. However, large values of Δt are not used in practice because the truncation errors that are produced become important, eliminating the validity of the approach described in this section.

CHAPTER 3. STABILIZED FINITE ELEMENT METHODS FOR CONVECTION-DIFFUSION PROBLEMS

3.1. Stabilization of the Steady One-Dimensional Problem

In the previous chapter we showed how the usual finite element method solution of convection and diffusion problems where convection effects are important leads to unstable results, and that as a consequence numerical stabilization has to be added. In the same way that the solution of the 1D steady transport problem was used to illustrate such need, it is also the best model problem for describing what has to be done for achieving a stable solution, and hence will be again considered in this section.

3.1.1. Analysis of the Galerkin Discrete Equation and the Cause of the Instabilities

The best way of finding a possible remedy for a given numerical difficulty is to identify its cause. The fact of having a problem with a close exact analytical solution like in the case of the 1D steady convection-diffusion equation is a great advantage for that purpose, because by a comparison of the discrete equations provided by both the numerical and the analytical solution the source of the difficulty can be found, and indeed this is what will be done next.

Like any numerical approach, the Galerkin finite element method has a truncation error. In order to expose it, we start by obtaining the discrete equation that the method delivers at a given interior mesh node. Recalling figure 2.3 (known as a tree-node stencil) and the element matrices for the contribution to the solution of the problem under consideration, that is equations (2.35) to (2.37), we can perform the assembly for the two elements that share node i, resulting in the following system of equations

$$\underbrace{\begin{bmatrix} -\frac{u}{2}+\frac{k}{h} & \frac{u}{2}-\frac{k}{h} & 0 \\ -\frac{u}{2}-\frac{k}{h} & \frac{2k}{h} & \frac{u}{2}-\frac{k}{h} \\ 0 & -\frac{u}{2}-\frac{k}{h} & \frac{u}{2}+\frac{k}{h} \end{bmatrix}}_{(\mathbf{C}+\mathbf{K})} \underbrace{\begin{Bmatrix} T_{i-1} \\ T_i \\ T_{i+1} \end{Bmatrix}}_{\mathbf{T}} = \underbrace{\begin{Bmatrix} \frac{hf}{2} \\ hf \\ \frac{hf}{2} \end{Bmatrix}}_{\mathbf{f}}, \tag{3.1}$$

where it has to be emphasized that u, k and f are constants. Now, taking the equation for node i, that is, the second equation of the system, we find

$$u\left(\frac{T_{i+1}-T_{i-1}}{2h}\right)-k\left(\frac{T_{i+1}-2T_i+T_{i-1}}{h^2}\right)=f. \tag{3.2}$$

Equation (3.2) is the discrete equation that the Galerkin method produces at an interior node i for the unidimensional steady convection-diffusion problem. It is easy to observe that it is exactly the same discrete scheme that would be obtained if second-order central differences were used in equation (2.27). This fact, which is not a pure coincidence, reflects the close relation that is found between the Galerkin method based on linear elements and the central-difference finite difference method. Moreover, it also gives evidence that central finite difference solutions are also affected by the same kind of oscillations as the Galerkin formulation, showing that they are not a particular issue of the finite element method. A description of the unstable behavior of convection-dominated problems in a finite difference context can be found in [1].

Having obtained the discrete equation for the numerical scheme, the next step in our way to find the truncation error of the Galerkin method is to get a discrete expression for the exact solution, that is, an equation similar to (3.2) that, when solved, would give the exact solution to the problem at the considered node, no matter which value of the Péclet number is used.

In order to obtain an exact scheme, an equation of the type

$$\alpha_1 T_{i-1}+\alpha_2 T_i+\alpha_3 T_{i+1}=f \tag{3.3}$$

has to be solved, where the nodal values T_{i-1}, T_i and T_{i+1} are expressed by means of the exact solution (2.29), thus obtaining three conditions on the coefficients α_1, α_2 and α_3 that allow solving for them. The details are given by Donea and Huerta in [3]. With this process, the desired exact scheme becomes

$$u\left(\frac{T_{i+1}-T_{i-1}}{2h}\right)-\left(k+\bar{k}\right)\left(\frac{T_{i+1}-2T_i+T_{i-1}}{h^2}\right)=f, \tag{3.4}$$

where $\bar{k}$ can be interpreted as an added numerical diffusion (a non-physical diffusion coefficient) that has the form (see also [11])

$$\bar{k}=\beta\frac{uh}{2} \qquad \text{with } \beta=\coth Pe-\frac{1}{Pe}. \tag{3.5}$$

Equation (3.4) is the one that gives the exact nodal solution to the problem for any configuration of the parameters. In fact, $\bar{k}$ is a function that only depends on the parameters of the governing equation (except the source term) and the element size. With this, we are now ready to discover why the Galerkin method fails to deliver correct results in convection-dominated situations.

By a direct comparison between both the numerical (3.2) and the exact (3.4) difference discrete schemes we see that the Galerkin method introduces a truncation error in the form of a diffusion operator, i.e. it lacks the term

$$-\bar{k}\left(\frac{T_{i+1}-2T_i+T_{i-1}}{h^2}\right)$$

to represent the exact nodal solution. As can be observed, this is a truncation error that depends on the problem parameters, specially on the Péclet number, explaining why the presence and intensity of the spurious oscillations depends on the value of such number.

An analysis of this truncation error allows to find which is the modified partial differential equation that is actually solved exactly at the nodes by the Galerkin finite element method. Such a modified equation is (see [3])

$$u\frac{\partial T}{\partial x} - \left[k - \bar{k}\left(\frac{\sinh^2 Pe}{Pe^2}\right)\right]\frac{\partial^2 T}{\partial x^2} = f, \tag{3.6}$$

where it can be noticed that the diffusion coefficient is now given by the term inside brackets. This modified diffusion coefficient is what the Galerkin method uses as the "physical" diffusion coefficient when solving the problem. However, as pointed in chapter 1, the diffusion coefficient is physically defined as a positive quantity ($D > 0$), and looking at equation (3.6) we see that it may become negative if $\bar{k}\left(\sinh^2 Pe/Pe^2\right) > k$. It can be checked that this condition is satisfied when the Péclet number is larger than one, that is, when convection dominates. Hence, when this happens, a non-physical model is being solved by the Galerkin method, and the corresponding numerical solution is corrupted by the already known node-to-node oscillations.

After these arguments, we see that the negative numerical diffusion introduced by the Galerkin finite element scheme is the cause of the numerical difficulties that arise in the simulation of convective transport problems. But most importantly, now we have an idea of how to develop a remedy for such deficiencies.

3.1.2. The Optimal Formulation

With the source of the numerical instabilities identified, the results of the previous section can be exploited to develop an optimal method for the solution of the one-dimensional steady transport problem. As exposed by Donea and Huerta [3], the behavior of the Galerkin method in convection-dominated problems can be improved by the addition of an artificial diffusion that counteracts the negative dissipation introduced by the method, that is, its truncation error. The advantage of this approach is that the form of such artificial diffusion (also known as balancing diffusion) is given by $\bar{k}$, as expressed in equation (3.5). For illustrating the improved performance of this method, the problem described in section 2.5. is again solved.

Considering the original governing equation of the problem (2.27) and introducing the modified diffusion coefficient we obtain

$$u\frac{\partial T}{\partial x} - \left(k + \bar{k}\right)\frac{\partial^2 T}{\partial x^2} = f, \tag{3.7}$$

which is taken as the new governing partial differential equation. For this equation, the following weighted residual formulation is obtained

$$\int_0^L \left[\mathrm{w}u\frac{\partial T}{\partial x} + \frac{\partial \mathrm{w}}{\partial x}\left(k + \bar{k}\right)\frac{\partial T}{\partial x}\right]dx = \int_0^L \mathrm{w}f\,dx, \tag{3.8}$$

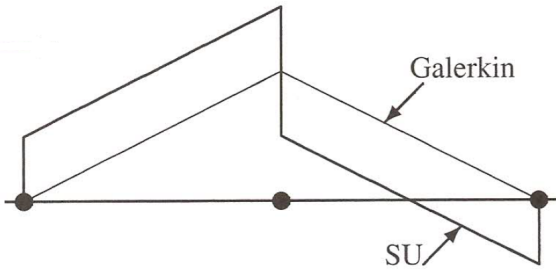


Figure 3.1: Modified weighting function characteristic of the SU method for linear elements, in comparison with that of the Galerkin method. From [3].

which, substituting for the value of $\bar{k}$, can be rewritten in the form

$$\int_0^L \left[\left(\mathrm{w} + \beta \frac{h}{2} \frac{\partial \mathrm{w}}{\partial x} \right) u \frac{\partial T}{\partial x} + \frac{\partial \mathrm{w}}{\partial x} k \frac{\partial T}{\partial x} \right] dx = \int_0^L \mathrm{w} f dx. \tag{3.9}$$

Equation (3.9) shows a very interesting result, namely, that the introduction of the balancing diffusion $\bar{k}$ produces a weak formulation that uses a modified weighting function for the convective term. Such a modified weighting, which is given by $\mathrm{w} + \beta\,(h/2)\,\partial \mathrm{w}/\partial x$, is shown in figure 3.1 for the central node of a three-node stencil, in comparison with the standard Galerkin weighting function, i.e. the shape function of a 1D linear element. As can be observed, the modified function is discontinuous at the inter-elemental boundaries and gives more weight to the element upstream of the central node (the flow is from left to right) than to the element downstream, whereas the Galerkin function weights equally both elements. This kind of modified weighting function is known as an upwind-type weighting function, and the scheme that results from its application to the convective term as in equation (3.9) is referred to (when generalized to multiple spatial dimensions) as the streamline-upwind (SU) method.

The virtues of the 1D SU method will be exposed next, when presenting the solution of the modified equation (3.7). However, a more general description of the method is given in section 3.2.1.. Proceeding as in section 2.5., the spatial discretization of the weak form (3.9) is performed using linear finite elements. The only difference is in the element convection matrix (the diffusion and source terms are not changed by the SU formulation), that now takes the following SU form

$$\mathbf{C}^e_{SU} = \frac{u}{2} \begin{bmatrix} -(1-\beta) & 1-\beta \\ -(1+\beta) & 1+\beta \end{bmatrix}. \tag{3.10}$$

Considering the same mesh of ten linear elements used for the Galerkin solution of the problem, and solving it for the same values of the Péclet number taken before, the SU method delivers the results displayed in figure 3.2. Comparing this figure with the results of the Galerkin method, shown in figure 2.4, the difference is very clear. With the addition of the artificial diffusion that characterizes the streamline-upwind scheme, the exact solution of the problem is obtained at each node, thus completely removing all the spurious oscillations that were present before. Hence, the truncation error due to the SU formulation of the problem is zero, and is the optimal finite element approach for this case. Since the instabilities have disappeared, this formulation is indeed a stabilization technique.

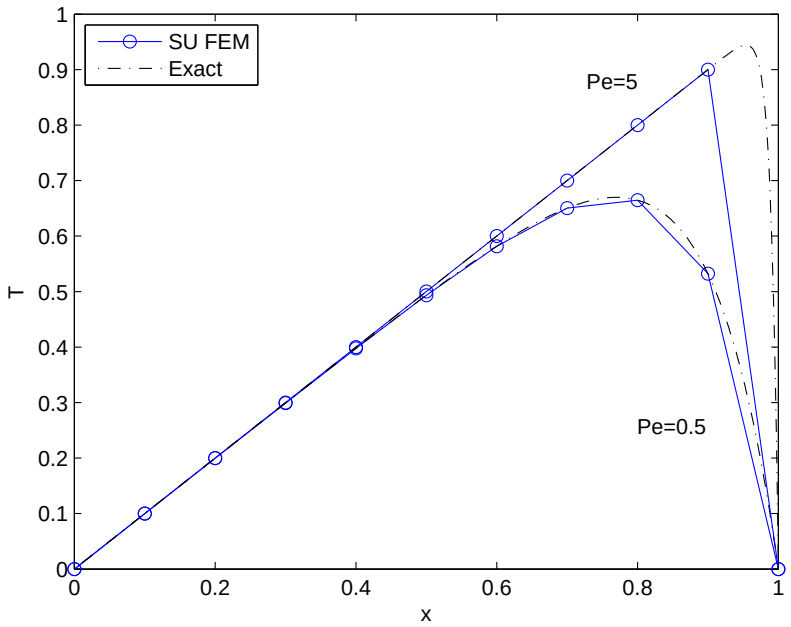


Figure 3.2: Streamline-upwind (SU) finite element solution of the 1D steady transport problem with $u = 1$ and $f = 1$ for two different Péclet numbers. The exact solution is also shown for comparison.

The upwind-type weighting function that has been obtained with the introduction of the balancing diffusion has a numerical meaning that goes beyond the solution of the simple model problem presented here. Actually, it was discovered in a finite difference context that stable but overdiffusive solutions could be obtained by the use of upwind differentiation of the convective term, i.e. approximating the convective derivatives with solution values at the upstream and central nodes of a three-node stencil. However, it was by the combination of central and upwind differences that the optimal (exactly nodal) solution was obtained (consult [11] for more information). This is reflected in the form of the SU weighting function, recall figure 3.1, where, although most of the weight is given to the upstream element, some is still applied to the downstream one. What this means is that the need for upwinding is an inherent property of the numerical discretization of the convective differential operator, given by its mathematical nature, and has nothing to do with whether a finite difference or a finite element approach is used. Moreover, its behavior is generalized for multidimensional problems, steady or transient, and the same stabilization philosophy has to be used for all cases.

3.2. Stabilization Techniques for Multidimensional Problems

Due to the practical importance of obtaining stabilized formulations for problems in multiple dimensions, the development of general optimal formulations has always been a matter of research. Although a method that provides the exact nodal solutions in multidimensional domains has not been achieved yet, the application of the concepts presented in the pre-

vious section to more complex problems has resulted in satisfactory results.

3.2.1. The General Streamline-Upwind Formulation

Convective transport is a physical process that takes place along the streamlines in the flow field. Research studies repeatedly showed that the key for the extrapolation of the SU stabilization to multidimensional problems is that the balancing diffusion has to be added in the flow direction only, and not transversely (the so-called crosswind diffusion). Although the concept of streamlines losses its meaning in 1D (there is only one possible flow direction), this is where the streamline-upwind method takes its name from, being based on the idea of adding diffusion along the streamlines only.

Since an exact solution for the general steady convection-diffusion equation (2.12) is not available, an analysis like the one performed for the one-dimensional case is not possible. Hence, focusing on the directional character of the convective term, and with the arguments given in the previous paragraph, Hughes and Brooks (see [12, 13]) proposed to add the artificial diffusion in a tensorial form that acts only in the flow direction. The resulting SU weak formulation for the general problem is as follows

$$\int_\Omega \left\{ \left[\mathrm{w} + \frac{\bar{k}}{V} (\mathbf{V} \cdot \nabla \mathrm{w}) \right] (\mathbf{V} \cdot \nabla T) + \nabla \mathrm{w} \cdot (k \nabla T) \right\} d\Omega = \int_\Omega \mathrm{w} f \, d\Omega + \int_{\Gamma_N} \mathrm{w} q_n \, d\Gamma. \quad (3.11)$$

Note that this equation has the same structure as the weak form for the 1D problem, namely, equation (3.9). Actually, it is its exact extrapolation to multiple dimensions, i.e. (3.9) can be obtained when particularizing (3.11) for a single dimension. As can be observed, the introduced artificial diffusion is only affecting the convective term, and attending to its structure, it modifies the weighting function just along the streamlines, as given by the term $\mathbf{V} \cdot \nabla \mathrm{w}$. Similarly, now the velocity norm V ($V = ||\mathbf{V}||^2$) is used for computing the artificial diffusivity $\bar{k}$. Recalling that the modified weighting function is discontinuous at the element boundaries, equation (3.11) is more conveniently expressed as

$$\overbrace{\int_\Omega \mathrm{w} (\mathbf{V} \cdot \nabla T) \, d\Omega + \int_\Omega \nabla \mathrm{w} \cdot (k \nabla T) \, d\Omega}^{\text{Galerkin terms}} + \underbrace{\sum_e \int_{\Omega_e} \frac{\bar{k}}{V} (\mathbf{V} \cdot \nabla \mathrm{w}) (\mathbf{V} \cdot \nabla T) \, d\Omega}_{\text{SU stabilization term}} = \int_\Omega \mathrm{w} f \, d\Omega + \int_{\Gamma_N} \mathrm{w} q_n \, d\Gamma, \quad (3.12)$$

which is the general weak form of the SU method. Hence, the SU formulation can be interpreted as the standard formulation delivered by the Galerkin method plus an extra term that comes from the addition of the balancing diffusion and that is the responsible of stabilizing the solution, called the stabilization term.

Although the use of the SU finite element method in multiple dimensions leads to stable results, it is by no means an exact method. It presents accuracy problems in the cases where the velocity field and/or the source term are not uniform (the truncation error analysis performed before assumed uniform conditions). However, it was discovered that such

inaccuracies were due to the fact that modifying the weighting function only for the convective term produces a non-residual formulation, that is, that the solution of the weak form (3.12) is no longer a solution of the original partial differential equation that governs the problem, and thus the desired solution is not guaranteed.

In order to solve the accuracy problems that characterize the SU method, Hughes and Brooks subsequently proposed (see [13]) that the modified weighting function should be applied to all the terms in the governing equation. In this way, it is ensured that the solution of the stabilized weak form is also a solution of the differential equation, producing what is known as a consistent formulation, which is described in the next section.

3.2.2. The Streamline-Upwind Petrov-Galerkin Method

To achieve the objective of stabilizing the convective term in a consistent manner, that is, to perform a consistent stabilization, a residual formulation has to be introduced. This means weighting all the terms of the original weak formulation by using the SU weighting function. The common practice to do this is to add an extra term to the Galerkin formulation, in a similar form to (3.12), but acting on the residual of the governing partial differential equation (2.12), which is defined for the steady case as

$$\mathrm{R}(T) = \mathbf{V}\cdot\nabla T - k\nabla^2 T - f. \tag{3.13}$$

With this, the consistent stabilized formulation for the SU method becomes

$$\begin{aligned}\int_\Omega \mathrm{w}\,(\mathbf{V}\cdot\nabla T)\;d\Omega + \int_\Omega \nabla\mathrm{w}\cdot(k\nabla T)\;d\Omega \\ + \sum_e \int_{\Omega_e} (\mathbf{V}\cdot\nabla\mathrm{w})\,\tau\left(\mathbf{V}\cdot\nabla T - k\nabla^2 T - f\right)\;d\Omega = \int_\Omega \mathrm{w} f\;d\Omega + \int_{\Gamma_N} \mathrm{w} q_n\;d\Gamma,\end{aligned} \tag{3.14}$$

where $\tau = \bar{k}/V$ is defined as the stabilization parameter (also called intrinsic time). Comparing this expression with (3.12), we see that now the stabilization term involves the residual (3.13), and that as a result the upwind-type weighting function is consistently applied to all terms of the original equation. This formulation presented in equation (3.14), where the weighting functions do not coincide with the interpolation functions (the shape functions), is what is known as a Petrov-Galerkin formulation. For this reason, this stabilization technique that results from consistently stabilizing the original weighted residual formulation by means of SU weighting functions is called the streamline-upwind Petrov-Galerkin (SUPG) method. Since its development, it has been successfully applied in CFD. Refer to [20] and [21] for pioneer implementations.

It is important to note that when linear elements are used to discretize equation (3.14), the second-order operator found in the stabilization term vanishes (or is largely underrepresented), since it cannot be further integrated by parts to reduce the continuity requirements. In such a case the SUPG stabilization term reduces to

$$\sum_e \int_{\Omega_e} (\mathbf{V}\cdot\nabla\mathrm{w})\,\tau\,(\mathbf{V}\cdot\nabla T - f)\;d\Omega,$$

and thus the only difference with the SU formulation lies on the source term. This causes a degradation in the consistency of the formulation since the diffusion term is not weighted by the upwind-type function. As commented by Donea and Huerta in [3], this lack of consistency leads to errors of the order of the stabilization parameter ($O(\tau)$).

A part from adding the numerical diffusion in the correct way, i.e. along the flow direction, it is also important to add the right amount of it. If not, both oscillatory or overdiffusive results can be obtained. The responsible for regulating the amount of numerical dissipation that is added is the stabilization parameter τ, and hence it plays a key role in stabilization techniques. Its form was exactly obtained for the case of 1D steady convection-diffusion transport approximated with linear elements, as given by equation (3.5), but for higher spatial dimensions an exact expression is not available. Different studies have been focused on the obtention of a general and optimal definition for τ, but none has succeeded yet. The most satisfactory results are based on algebraic analyses of the truncation error that produce a definition of τ such as the one from Codina (refer to [16, 17])

$$\tau = \left(\frac{2V}{h} + \frac{4k}{h^2}\right)^{-1} = \frac{h}{2V}\left(1 + \frac{1}{Pe}\right)^{-1}, \tag{3.15}$$

which should be evaluated for each element in the mesh if the problem parameters are not uniform. Its structure corroborates the fact that no stabilization is needed for a fine enough mesh, i.e. the stabilization parameter vanishes when h is made very small.

Although along this section we have been focused on the steady convection-diffusion equation, the usual extension of the SUPG stabilization technique to unsteady transport problems is quite straightforward for the case of the Crank-Nicolson method. For the transient case the stabilization term is added to the left-hand side of the weak formulation of equation (2.41), with the same structure as in (3.14) but taking into account that now the residual will be different, namely, that of equation (2.41). The main difference is that the stabilization parameter should be modified to account for the time stepping. Following the analysis of Shakib et al. [18] (see also [19]), Codina's formulation (3.15) adapted for the Crank-Nicolson scheme becomes

$$\tau = \left(\frac{2}{\Delta t} + \frac{2V}{h} + \frac{4k}{h^2}\right)^{-1}, \tag{3.16}$$

which also tends to zero as Δt is made small. Despite this approach is the one that is commonly adopted, the idea of using upwind-type weighting functions that additionally account for the time discretization has also been studied, leading to good results. A presentation of this concept is given by Tezduyar and Ganjoo in [22].

3.2.3. The Sub-Grid Scale Method

Moved by the motivation of obtaining a better understanding of the theoretical foundations of stabilization methods, some research studies (see for instance [14] and [15]) showed that the origins of stabilization techniques emanate from a particular class of what is known as sub-grid scale models. These can be viewed as methods for dealing with the multiscale phenomena that characterizes many physical problems, including fluid dynamics. They are based on the idea that standard finite element approximations such as the Galerkin method

can only numerically resolve the coarse-scale aspects of these problems, and that as a consequence they need a means of incorporating the effects of the unresolvable scales in order to be able to produce accurate and stable results. Otherwise, their solutions may lead to non-physical (unstable) states, producing the already known spurious oscillations.

Attending to this philosophy, the sub-grid scale (SGS) method rests upon the additive decomposition of the solution variable T on a coarse scale component, $\bar{T}$, which can be resolved numerically by the considered finite element mesh (and hence $\bar{T} = T^h$), and a fine-scale component (also called sub-grid scale or subscale), T', which is modeled analytically, that is

$$T = T^h + T', \tag{3.17}$$

where it can be observed that the fine-scale actually represents the error $(T - T^h)$ made by the finite element approximation. In the same way, the weighting function is split as $\mathrm{w} = \mathrm{w}^h + \mathrm{w}'$. With these considerations, the original weak form of the problem, namely equation (2.4), can be equivalently expressed for the steady case by the following two statements

$$\begin{aligned}&\int_\Omega \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T^h\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}^h \cdot \left(k\nabla T^h\right)\, d\Omega \\ &\quad + \int_\Omega \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T'\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}^h \cdot \left(k\nabla T'\right)\, d\Omega = \int_\Omega \mathrm{w}^h f\, d\Omega + \int_{\Gamma_N} \mathrm{w}^h q_n\, d\Gamma,\end{aligned} \tag{3.18}$$

completed by

$$\begin{aligned}&\int_\Omega \mathrm{w}' \left(\mathbf{V}\cdot\nabla T^h\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}' \cdot \left(k\nabla T^h\right)\, d\Omega \\ &\quad + \int_\Omega \mathrm{w}' \left(\mathbf{V}\cdot\nabla T'\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}' \cdot \left(k\nabla T'\right)\, d\Omega = \int_\Omega \mathrm{w}' f\, d\Omega + \int_{\Gamma_N} \mathrm{w}' q_n\, d\Gamma,\end{aligned} \tag{3.19}$$

where the first problem governs the resolvable scales and the second one the sub-grid scale.

Then, as noted above, the objective is to solve problem (3.19) analytically for T', which can be then substituted into (3.18) to obtain a finite element equation for T^h. Among the different ways in which this can be done (refer to the work of Hughes et al. [15] for a general presentation), here we will consider the simplest one, known as the algebraic sub-grid scale (ASGS) approach, which consists on modeling T' as

$$T' = -\tau\, \mathrm{R}\left(T^h\right) = -\tau\left(\mathbf{V}\cdot\nabla T^h - k\nabla^2 T^h - f\right), \tag{3.20}$$

that is, as a function of the residual of the approximation T^h scaled by the stabilization parameter (with the same form exposed in the previous section). As can be noticed, the model provided by equation (3.20) does not make any consideration to the boundary conditions. The reason is that it is standard in the SGS method to impose $T' = 0$ along the finite element edges, in order to localize the sub-grid scale problem in the element interiors only.

In view of the above assumptions, and substituting expression (3.20) into equation (3.18), the terms for the unresolvable scales become

$$\int_\Omega \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T'\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}^h \cdot \left(k\nabla T'\right)\, d\Omega = \sum_e \int_{\Omega_e} \left[\mathbf{V}\cdot\nabla \mathrm{w}^h + \nabla \mathrm{w}^h\cdot(k\nabla)\right] \tau \left(\mathbf{V}\cdot\nabla T^h - k\nabla^2 T^h - f\right)\, d\Omega, \quad (3.21)$$

where the convective term inside the brackets has been integrated by parts to reduce the continuity requirements, with the resulting boundary integral being neglected due to the restriction of definition (3.20) to element interiors Ω_e. With this, the weak form of the finite element approximation that accounts for the fine-scale behavior reads

$$\int_\Omega \mathrm{w}^h \left(\mathbf{V}\cdot\nabla T^h\right)\, d\Omega + \int_\Omega \nabla \mathrm{w}^h \cdot \left(k\nabla T^h\right)\, d\Omega - \int_\Omega \mathrm{w}^h f\, d\Omega - \int_{\Gamma_N} \mathrm{w}^h q_n\, d\Gamma + \underbrace{\sum_e \int_{\Omega_e} \left[\underbrace{\mathbf{V}\cdot\nabla \mathrm{w}^h}_{\text{SUPG}} + \nabla \mathrm{w}^h\cdot(k\nabla)\right] \tau \left(\mathbf{V}\cdot\nabla T^h - k\nabla^2 T^h - f\right)\, d\Omega}_{\text{ASGS stabilization term}} = 0. \quad (3.22)$$

We see that equation (3.22) has the same structure as that resulting from the SUPG technique (3.14), i.e. it is the original weak formulation plus an added term that operates over the residual of the governing equation, labeled here as the ASGS stabilization term. This expression is the result from adding the effect of the sub-grid scales to the standard (Galerkin) finite element formulation of the problem, and, as can be observed, for the model chosen in (3.20) such effect is added in the form of a consistent stabilization term.

An important particular case is once again that of linear finite elements, where, analyzing equation (3.22), we can notice that the term $\nabla \mathrm{w}^h \cdot (k\nabla)\,\tau \left(\mathbf{V}\cdot\nabla T^h - k\nabla^2 T^h - f\right)$ would vanish, producing the classical SUPG formulation. Hence, we conclude that SUPG is ASGS particularized for linear elements.

More details on the SGS stabilization of the steady convection-diffusion equation can be found in the work of Codina [17]. Regarding the unsteady transport case, different SGS techniques can be derived. In a lot of situations it is assumed that the sub-grid scales are static, i.e. $\partial T'/\partial t = 0$, allowing a direct extension of the formulation presented in this section, and maintaining the same structure as the SUPG method for linear elements. On other cases, the transients of the fine-scale components are considered, leading in the ASGS approach to weighting functions that take into account the time step size Δt and that, as a result, produce better solutions than the SUPG method. The reader is referred to [3] for the details.

CHAPTER 4. A NEW MIXED FINITE ELEMENT FORMULATION FOR CONVECTION-DIFFUSION PROBLEMS

4.1. Motivation

In the majority of situations, fluid dynamics problems are characterized by the presence of rapid changes of the flow properties that occur in small regions of space, specially near solid surfaces. Common examples that illustrate this are boundary layers, stagnation points, the flow acceleration or deceleration around aerodynamic shapes like airfoils, or even shock waves in the case of supersonic flows. These are the places where the most important physical phenomena take place, and having the right tools for their analysis is key for the understanding of fluid motion.

Mathematically, such rapid changes are traduced in strong gradients that affect the primitive variables of the problem, and their accurate numerical representation is very important in CFD. For a given numerical technique, which is defined by an inherent truncation error, the most direct way of increasing the numerical precision in the desired regions of space is to locally refine the mesh. However, in order to take all the advantage from this process, the considered numerical method has to properly account for the local discretization of these gradients. Restricting ourselves to a finite element context, this is not the case at all for linear elements.

When standard convergence analyses are performed for the finite element method (see for example [5]), the results show that the convergence rate for an unknown function T interpolated with a polynomial of degree p is of order $O\left(h^{p+1}\right)$, with h being a characteristic mesh size. In the same way, the mth derivative of T should converge with an error of order $O\left(h^{p+1-m}\right)$.

Focusing on the finite element discretization of the convection-diffusion equation that has been exposed in the previous two chapters, that is, equation (1.11) approximated with linear elements, we find in view of the previous argument that the convergence rate of our solution is of order $O\left(h^2\right)$ for the unknown variable T, and of order $O\left(h\right)$ for the gradient ∇T. In other words, that means for example that for a halving of the mesh size h, the error in T is reduced by $1/4$, and for ∇T by $1/2$. However, it must be recalled that since the finite element method is based on a weak integral formulation of the problem, i.e. the weak form produced by the weighting of the residual, these orders refer to the average (global) error of the solution, and that as a consequence there is no guarantee that the nodal (local) errors would also satisfy such convergence rates. This implies that when linear elements are used, the local error in the gradient can no longer be controlled by the mesh size h, that is, it is reduced to the order $O\left(1\right)$, producing inaccurate results specially in those zones where the local gradients are sharp or singular, such as at the stagnation points on the leading and trailing edges of an airfoil.

Although the choice of higher-order elements could solve this problem, the practical interest in using linear elements has motivated the development of a new formulation for circumventing such deficiencies. The idea has been proposed by Dr. Riccardo Rossi

(originally presented in [24]), a researcher from the International Center for Numerical Methods in Engineering (CIMNE). It consists on introducing an auxiliary variable, defined as the gradient of the transported quantity (∇T), to produce a mixed formulation in which ∇T is computed with the same convergence rate as T, namely, $O\left(h^2\right)$. This approach will be made clear in the next section, where the formulation is presented.

4.2. Convection-Diffusion in Mixed Form

4.2.1. Original Formulation

Considering the general scalar convection diffusion problem (1.11), repeated here for convenience

$$\frac{\partial T}{\partial t} + \mathbf{V} \cdot \nabla T - k\nabla^2 T = f,$$

Rossi's proposal is to introduce the new vector variable $\mathbf{q} = \nabla T$ and to express the problem in terms of both T and $\mathbf{q}$, thus extending it to a system of equations. Then, substituting for $\mathbf{q}$ on the original equation we obtain

$$\frac{\partial T}{\partial t} + \mathbf{V} \cdot \mathbf{q} - k\nabla \cdot \mathbf{q} = f \tag{4.1}$$

completed by

$$\nabla T - \mathbf{q} = \mathbf{0}. \tag{4.2}$$

Equations (4.1) and (4.2) constitute what is known as a mixed formulation, i.e. a multiple equation problem where the number of dependent unknowns can be reduced by suitable algebraic operations, still leaving a solvable problem (refer to [5] for a general presentation). It is important to notice that the second equation (4.2) of the system is a vector equation, so the system size varies with the number of spatial dimensions considered. This means that the solution of the problem in mixed form involves dealing with a larger number of unknowns which, obviously, increases the computational cost of the numerical solution. However, such an increase has found to be very reasonable and numerical tests demonstrate that it is not an issue for conventional convection-diffusion problems.

When both equations that constitute the system are enforced weakly, taking into account that the weighting function for each unknown variable can be different, the weak form of the problem becomes

$$\begin{gathered} \int_\Omega \mathrm{w} \left(\frac{\partial T}{\partial t} + \mathbf{V} \cdot \mathbf{q} - k\nabla \cdot \mathbf{q} \right) \, d\Omega = \int_\Omega \mathrm{w} f \, d\Omega \\ \int_\Omega \boldsymbol{\psi} \left(\nabla T - \mathbf{q} \right) \, d\Omega = \mathbf{0}, \end{gathered} \tag{4.3}$$

where w is a test function for the space in which T lives, and $\boldsymbol{\psi}$ is a multidimensional test function for the space in which $\mathbf{q}$ is defined. It shall be noted that now the continuity

requirements have been reduced, and hence there is no need of integrating by parts the diffusion term unless natural boundary conditions should be accounted for.

Observing the weak system (4.3), we can expect the convergence rates of both T and $\mathbf{q}$ to be the same and of order $O\left(h^2\right)$ for linear elements. Attending to the arguments exposed in the previous section, we see that by means of the mixed form (4.3) we have introduced a control over the local convergence of the gradient (the local gradient error is now of order $O(h)$), that is, we now have a guarantee that the local gradients would convergence to what they should. On the other hand, it has to be noted that for the Laplacian, which now becomes $\nabla \cdot \mathbf{q}$, such local control is still not achieved. Despite its global convergence rate has increased one order, i.e. from $O(1)$ in the classical formulation to $O(h)$ in the mixed form, the local error remains on $O(1)$. Nevertheless, the interest of this new approach is not on the Laplacian but on increasing the accuracy with which the gradient is calculated, as justified in the previous section.

While the previous weak form could be directly discretized, the resulting formulation is not stable, not only when convective effects are important, but in diffusion-dominated problems too. The reason is that mixed formulations have extra numerical difficulties that may lead to additional stability problems (refer to [5] for a general discussion), and for this case stabilization is always needed. To stabilize it Rossi proposes to use the algebraic sub-grid scale (ASGS) approach, described in section 3.2.3., with static subscales, which corresponds to assuming that

$$T = T^h + T' \qquad \text{with} \qquad T' = \tau_T \left(f - \frac{\partial T^h}{\partial t} - \mathbf{V} \cdot \mathbf{q}^h + k \nabla \cdot \mathbf{q}^h \right) \tag{4.4}$$

and

$$\mathbf{q} = \mathbf{q}^h + \mathbf{q}' \qquad \text{with} \qquad \mathbf{q}' = \tau_q \left(\nabla T^h - \mathbf{q}^h \right), \tag{4.5}$$

with τ_T and τ_q being the respective stabilization parameters. Basing on an algebraic analysis, Rossi considers using expression (3.15) for τ_T and a constant value for τ_q (he recommends using $\tau_q = 0.1$). With these models, the stabilized weak problem can be written as

$$\begin{aligned}
&\int_\Omega \mathrm{w}^h \left(\frac{\partial T^h}{\partial t} + \mathbf{V} \cdot \mathbf{q}^h - k \nabla \cdot \mathbf{q}^h \right) \, d\Omega + \sum_e \int_{\Omega_e} \mathrm{w}^h \, \mathbf{V} \cdot \left[\tau_q \left(\nabla T^h - \mathbf{q}^h \right) \right] \, d\Omega \\
&\qquad + \sum_e \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \left[k \tau_q \left(\nabla T^h - \mathbf{q}^h \right) \right] \, d\Omega = \int_\Omega \mathrm{w}^h f \, d\Omega \\
&\int_\Omega \psi^h \left(\nabla T^h - \mathbf{q}^h \right) \, d\Omega + \sum_e \int_{\Omega_e} \nabla \psi^h \, \tau_T \left(\frac{\partial T^h}{\partial t} + \mathbf{V} \cdot \mathbf{q}^h - k \nabla \cdot \mathbf{q}^h - f \right) \, d\Omega \\
&\qquad - \sum_e \int_{\Omega_e} \psi^h \, \tau_q \left(\nabla T^h - \mathbf{q}^h \right) \, d\Omega = \mathbf{0},
\end{aligned} \tag{4.6}$$

where the second stabilization term of the first equation and the first stabilization term of the second equation have been consistently integrated by parts under the assumption that subscales are zero on the element boundaries. Analyzing these expressions, we realize that only first derivatives appear. As a result, no terms would vanish when approximating

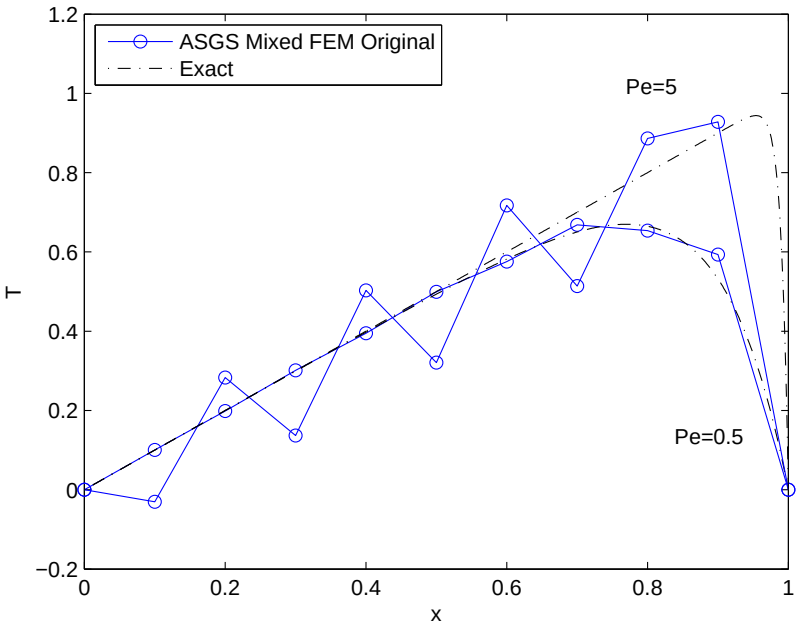


Figure 4.1: ASGS mixed finite element (original formulation) solution of the 1D steady transport problem with $u = 1$ and $f = 1$ for two different Péclet numbers. The exact solution is also shown for comparison.

the problem with linear finite elements, thus preserving the consistency of the stabilization. This is indeed another advantage of the mixed formulation.

Equation (4.6) is the weak form of the problem that is ready for the finite element implementation. For testing its performance, the usual model example has been considered, that is, the solution of the one-dimensional steady convection-diffusion equation. The results using the same configuration of the parameters as in chapters 2 and 3, and a value of $\tau_q = 0.1$, are displayed in figure 4.1. As can be observed, the formulation fails to provide stable results, even in the case of $Pe = 0.5$ some small oscillations are encountered. In fact, numerical experiments we have carried out show that they are present for all values of the Péclet number, becoming more severe as the convective effects are progressively increased but approaching an asymptotic amplitude for values of approximately $Pe > 10$. Moreover, they extend over the entire mesh with the same amplitude, not growing as they approach to the boundary layer. This induces us to think that the oscillations obtained in this case are of different nature than the classical spurious oscillations that characterize the Galerkin formulation (recall figure 2.4). Although we have not been able to identify their source, we think that due to their dependence with convective effects for relatively low values of Pe they should be related with the mixed formulation of the convective term.

4.2.2. Modified Formulation

In view of the of the results obtained in the previous section for the original formulation, Rossi proposed modifying it by leaving the convective term as in the original governing equation, that is

$$\begin{aligned} \frac{\partial T}{\partial t} + \mathbf{V}\cdot\nabla T - k\nabla\cdot\mathbf{q} &= f \\ \nabla T - \mathbf{q} &= \mathbf{0}. \end{aligned} \tag{4.7}$$

Then, proceeding in the same way as before, the stabilized weak form for the modified mixed problem becomes

$$\begin{aligned} &\int_\Omega \mathrm{w}^h \left(\frac{\partial T^h}{\partial t} + \mathbf{V}\cdot\nabla T^h - k\nabla\cdot\mathbf{q}^h \right) d\Omega + \sum_e \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \left[k\tau_q \left(\nabla T^h - \mathbf{q}^h \right) \right] d\Omega \\ &\quad \underbrace{+ \sum_e \int_{\Omega_e} \left(\mathbf{V}\cdot\nabla \mathrm{w}^h \right) \tau_T \left(\frac{\partial T^h}{\partial t} + \mathbf{V}\cdot\nabla T^h - k\nabla\cdot\mathbf{q}^h - f \right) d\Omega}_{\text{SUPG stabilization term}} = \int_\Omega \mathrm{w}^h f \, d\Omega \\ &\int_\Omega \psi^h \left(\nabla T^h - \mathbf{q}^h \right) d\Omega + \sum_e \int_{\Omega_e} \nabla\psi^h \, \tau_T \left(\frac{\partial T^h}{\partial t} + \mathbf{V}\cdot\nabla T^h - k\nabla\cdot\mathbf{q}^h - f \right) d\Omega \\ &\qquad - \sum_e \int_{\Omega_e} \psi^h \, \tau_q \left(\nabla T^h - \mathbf{q}^h \right) d\Omega = \mathbf{0}, \end{aligned} \tag{4.8}$$

where now both stabilization terms of the first equation have been integrated by parts. Observing the system (4.8) we see that an important difference with the original formulation has arisen, namely, that one of the terms of the first equation has the structure of the SUPG stabilization term. Although the stabilization technique applied in the original formulation and in this modified case is the same, i.e. the ASGS method, the change in the convection term of the mixed form (4.7) produces a streamline-upwind consistent stabilization term that was not encountered before.

This difference has indeed an important impact on the performance of the mixed formulation, as illustrated in figure 4.2, where once again the results for the 1D model transport problem are presented. Note that with the modification made on the convective term the instability problems have disappeared, with only residual oscillations remaining concentrated near the boundary layer for the case of $Pe = 5$. This is standard in convection-dominated problems and is an indicator that they have been correctly stabilized (see [3] for more examples). Thus we see that now the mixed formulation is working well, and further numerical tests for higher values of the Péclet number also corroborate these results. Then, it is now clearer that the stability problems of the original formulation are due to the convective term being written in terms of the auxiliary variable $\mathbf{q}$, even though we have not managed to obtain a theoretical explanation for this phenomenon.

The system (4.8) provides a weak formulation for the mixed problem that can be implemented with linear finite elements to obtain good results in general transport problems. However, this modification of the original method has an important drawback in terms of the advantages exposed previously. By leaving the convective term as in the original governing equation, that is, as $\mathbf{V}\cdot\nabla T$, we are introducing a gradient of the unknown variable in the first equation of the system. This, recalling the error concepts described before, causes the convergence rate of ∇T to decrease to the order $O(h)$, and hence eliminates the main advantage of using the mixed formulation in convection-diffusion problems. Despite this, for the case of pure diffusion problems the formulation still preserves the increased convergence rate, and hence it is expected to provide higher accuracy for the calculation of

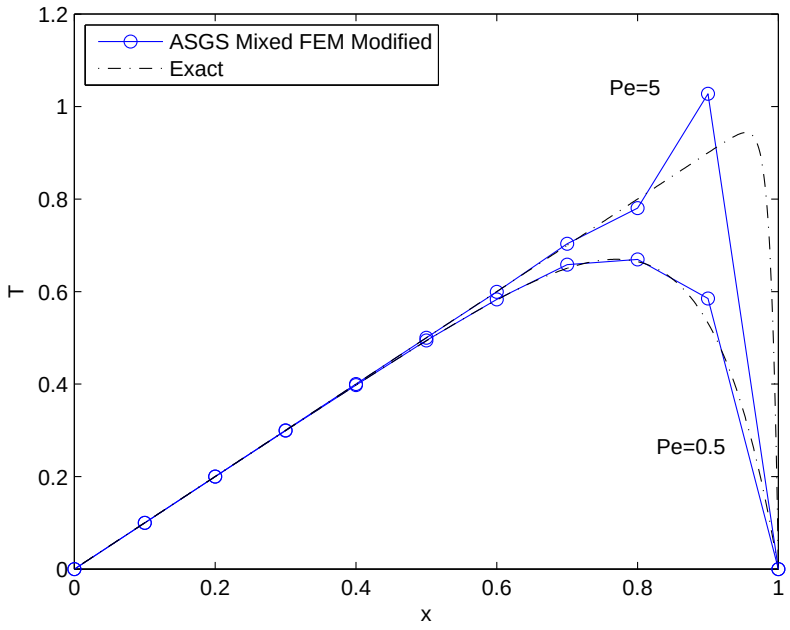


Figure 4.2: ASGS mixed finite element (modified formulation) solution of the 1D steady transport problem with $u = 1$ and $f = 1$ for two different Péclet numbers. The exact solution is also shown for comparison.

the gradients in such cases. In order to validate this argument, a pure diffusion problem is presented in the next section.

4.3. A Pure Diffusion Case: Incompressible Potential Flow

With the purpose of demonstrating the advantages of using the modified mixed formulation in pure diffusion situations, we have considered solving a two-dimensional steady pure diffusion problem. Focusing on a CFD context, there is a particular case in which the governing equations of fluid dynamics reduce to a pure diffusion model, namely, that of irrotational incompressible flow. The author has applied the formulation proposed by Dr. Rossi to this problem and has derived, implemented and validated the resulting finite element solution. The process is described next.

As detailed in any introductory fluid dynamics book (see for example [2]), an irrotational flow is the one that has no vorticity, i.e. that satisfies the condition $\nabla \times \mathbf{V} = 0$. Then, recalling the vector identity which states that the curl of the gradient of a scalar function ϕ is identically zero, that is, $\nabla \times (\nabla \phi) = 0$, we see that

$$\mathbf{V} = \nabla \phi, \tag{4.9}$$

which means that for an irrotational flow there exists a scalar function ϕ, known as the velocity potential, such that the flow velocity is given by the gradient of ϕ. Moreover, we

saw in chapter 1 that the continuity equation for an incompressible flow (equation (1.3)) takes the form of the incompressibility constraint, expressed as $\nabla \cdot \mathbf{V} = 0$. Combining this condition with equation (4.9), we have that for a flow that is both incompressible and irrotational $\nabla \cdot (\nabla \phi) = 0$, better expressed as

$$\nabla^2 \phi = 0. \tag{4.10}$$

Equation (4.10) is the governing equation for incompressible irrotational flow, commonly known as incompressible potential flow, or simply potential flow. It is called Laplace's equation, and is one of the most famous and extensively studied equations in mathematical physics, containing numerous analytical solutions available.

Comparing equation (4.10) with the general convection-diffusion equation (1.11), we see that the governing equation for incompressible potential flow is the scalar convection-diffusion equation particularized for steady pure diffusive transport (no convection) with $k = 1$, $f = 0$, and the velocity potential ϕ as the transported quantity. Hence, we see that incompressible potential flow is a suitable pure diffusion model. Moreover, since analytical solutions exist in some cases, it becomes an excellent problem for testing the modified mixed formulation.

Taking into account the standard notation used for potential flow problems, we now will use ϕ instead of T for denoting the transported quantity, and $\mathbf{V}$ instead of $\mathbf{q}$ for denoting the gradient of the transported quantity. With these considerations, the mixed formulation of the potential flow problem can be written as

$$\begin{aligned} \nabla \cdot \mathbf{V} &= 0 \\ \nabla \phi - \mathbf{V} &= \mathbf{0}. \end{aligned} \tag{4.11}$$

It has to be noted that although the velocity $\mathbf{V}$ appearing in the system is the fluid velocity, with its components defined as in (1.7), now it is not acting as a convection velocity, and the velocity potential is only being transported by a diffusion process.

Except in rare cases, for a flow to be irrotational it also has to be inviscid. The boundary condition for any inviscid flow is the flow tangency condition, namely, that the flow has to be always parallel to the wall. The mathematical way of stating this is to impose that the component of the velocity normal to the body surface must be zero, that is, $\mathbf{V} \cdot \mathbf{n} = 0$, with $\mathbf{n}$ being the unit vector normal to the surface. Then, since $\mathbf{V} = \nabla \phi$, we see that this is a Neumann-type boundary condition, which can be expressed as

$$\mathbf{V} \cdot \mathbf{n} = V_n = 0 \qquad \text{on } \Gamma_N. \tag{4.12}$$

Taking this into account, the weak formulation of the mixed problem reads

$$\begin{aligned} \int_\Omega \nabla \mathrm{w} \cdot \mathbf{V} \, d\Omega &= 0 \\ \int_\Omega \psi (\nabla \phi - \mathbf{V}) \, d\Omega &= \mathbf{0}, \end{aligned} \tag{4.13}$$

where the first equation has been integrated by parts in order to introduce the natural boundary condition (4.12), with $V_n = 0$ being imposed on the resulting boundary integral.

Using once again the ASGS method (as described in section 3.2.3.) for stabilizing the inherently unstable mixed form, the stabilized weak formulation of the problem becomes

$$\begin{aligned}
&\int_\Omega \nabla \mathrm{w}^h \cdot \mathbf{V}^h \, d\Omega + \sum_e \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \left[\tau_V \left(\nabla \phi^h - \mathbf{V}^h \right) \right] \, d\Omega = 0 \\
&\int_\Omega \psi^h \left(\nabla \phi^h - \mathbf{V}^h \right) \, d\Omega \\
&\quad - \sum_e \int_{\Omega_e} \nabla \psi^h \, \tau_\phi \left(\nabla \cdot \mathbf{V}^h \right) \, d\Omega - \sum_e \int_{\Omega_e} \psi^h \, \tau_V \left(\nabla \phi^h - \mathbf{V}^h \right) \, d\Omega = \mathbf{0},
\end{aligned} \tag{4.14}$$

where τ_V and τ_ϕ are the respective nomenclature modifications of τ_q and τ_T, which continue being computed as described in the previous section. Note that the first stabilization term of the second equation has been integrated by parts in the usual way. The system of equations (4.14) gives the weak form of the incompressible potential flow problem in mixed form suitable for a finite element implementation. As the governing equation for incompressible potential flow is a particular case of the general convection-diffusion model, (4.14) can be obtained directly from (4.8), with the corresponding nomenclature changes.

The next step towards the numerical solution of the problem is to perform the finite element spatial discretization. To do so, we follow the steps described in sections 2.3. and 2.4.. As discussed before, our interest is on using linear elements, so we choose to use the previously described linear triangular elements for the 2D potential flow problem. Writing expression (4.14) for a given element, and noting that τ_V is a constant, we find the following compact form of the approximated problem

$$\begin{aligned}
&(1-\tau_V) \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \mathbf{V}_e^h \, d\Omega + \tau_V \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \nabla \phi_e^h \, d\Omega = 0 \\
&(1-\tau_V) \int_{\Omega_e} \psi^h \left(\nabla \phi_e^h - \mathbf{V}_e^h \right) \, d\Omega - \int_{\Omega_e} \nabla \psi^h \, \tau_\phi \left(\nabla \cdot \mathbf{V}_e^h \right) \, d\Omega = \mathbf{0}.
\end{aligned} \tag{4.15}$$

At this point, to be able to express the discrete problem in a convenient matrix form we first need to expand the system. For two dimensions we will have three equations with three different unknowns, namely, the velocity potential ϕ_e^h and the velocity components u_e^h and v_e^h. Thus the system expands as

$$\tau_V \int_{\Omega_e} \nabla \mathrm{w}^h \cdot \nabla \phi_e^h \, d\Omega + (1-\tau_V) \int_{\Omega_e} \frac{\partial \mathrm{w}^h}{\partial x} u_e^h \, d\Omega + (1-\tau_V) \int_{\Omega_e} \frac{\partial \mathrm{w}^h}{\partial y} v_e^h \, d\Omega = 0 \tag{4.16a}$$

$$\begin{aligned}
&(1-\tau_V) \int_{\Omega_e} \psi_u^h \frac{\partial \phi_e^h}{\partial x} \, d\Omega + (\tau_V - 1) \int_{\Omega_e} \psi_u^h u_e^h \, d\Omega \\
&\quad - \int_{\Omega_e} \frac{\partial \psi_u^h}{\partial x} \tau_\phi \frac{\partial u_e^h}{\partial x} \, d\Omega - \int_{\Omega_e} \frac{\partial \psi_u^h}{\partial x} \tau_\phi \frac{\partial v_e^h}{\partial y} \, d\Omega = 0
\end{aligned} \tag{4.16b}$$

$$\begin{aligned}
&(1-\tau_V) \int_{\Omega_e} \psi_v^h \frac{\partial \phi_e^h}{\partial y} \, d\Omega + (\tau_V - 1) \int_{\Omega_e} \psi_v^h v_e^h \, d\Omega \\
&\quad - \int_{\Omega_e} \frac{\partial \psi_v^h}{\partial y} \tau_\phi \frac{\partial u_e^h}{\partial x} \, d\Omega - \int_{\Omega_e} \frac{\partial \psi_v^h}{\partial y} \tau_\phi \frac{\partial v_e^h}{\partial y} \, d\Omega = 0,
\end{aligned} \tag{4.16c}$$

where ψ_u^h and ψ_v^h are, respectively, the tests functions for the x and y velocity components.

We see that we have an equation for each one of the three unknowns, namely, equation (4.16a) for ϕ_e^h, (4.16b) for u_e^h and (4.16c) for v_e^h.

Attending to the approximation of the unknowns, the same shape functions have been chosen for interpolating each one of them inside the element domain. As commented previously, they are the shape functions for the linear triangular element, whose expressions were given in section 2.3.1. Defining the element nodal unknowns vectors as $\phi^e = \left[\phi_1^e, \phi_2^e, \phi_3^e\right]^T$, $\mathbf{u}^e = \left[u_1^e, u_2^e, u_3^e\right]^T$ and $\mathbf{v}^e = \left[v_1^e, v_2^e, v_3^e\right]^T$, the shape functions vector $\mathbf{N}^e$ for each unknown as in (2.15) and the shape functions gradient matrix $\nabla\mathbf{N}^e$ for each unknown as in (2.18), the system (4.16) can be expressed by the following equivalent block matrix form

$$\underbrace{\begin{bmatrix} \mathbf{K}_{\phi\phi}^e & \vdots & \mathbf{K}_{\phi u}^e & \vdots & \mathbf{K}_{\phi v}^e \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{K}_{u\phi}^e & \vdots & \mathbf{K}_{uu}^e & \vdots & \mathbf{K}_{uv}^e \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{K}_{v\phi}^e & \vdots & \mathbf{K}_{vu}^e & \vdots & \mathbf{K}_{vv}^e \end{bmatrix}}_{\mathbf{K}^e} \underbrace{\begin{bmatrix} \phi^e \\ \mathbf{u}^e \\ \mathbf{v}^e \end{bmatrix}}_{\mathbf{a}^e} = \mathbf{0}, \tag{4.17}$$

with each block of $\mathbf{K}^e$ being a 3×3 matrix with two subindexes. The first subindex indicates the row of the block, that is, the equation to which it belongs. The second subindex indicates the column, telling us the unknown variable to which the block corresponds. For example $\mathbf{K}_{\phi u}^e$ refers to the term that contains u_e^h in the equation for ϕ_e^h, and so on. Attending to this convention, the block matrices are given by

$$\begin{aligned} \mathbf{K}_{\phi\phi}^e &= \tau_V \int_{\Omega_e} \left[\nabla\mathbf{N}^e\right]^T \nabla\mathbf{N}^e \, d\Omega \\ \mathbf{K}_{uu}^e &= (\tau_V - 1) \int_{\Omega_e} \left[\mathbf{N}^e\right]^T \mathbf{N}^e \, d\Omega - \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial x} \tau_\phi \frac{\partial \mathbf{N}^e}{\partial x} \, d\Omega \\ \mathbf{K}_{vv}^e &= (\tau_V - 1) \int_{\Omega_e} \left[\mathbf{N}^e\right]^T \mathbf{N}^e \, d\Omega - \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial y} \tau_\phi \frac{\partial \mathbf{N}^e}{\partial y} \, d\Omega \\ \mathbf{K}_{\phi u}^e &= (1 - \tau_V) \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial x} \mathbf{N}^e \, d\Omega \qquad \mathbf{K}_{\phi v}^e = (1 - \tau_V) \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial y} \mathbf{N}^e \, d\Omega \\ \mathbf{K}_{u\phi}^e &= (1 - \tau_V) \int_{\Omega_e} \left[\mathbf{N}^e\right]^T \frac{\partial \mathbf{N}^e}{\partial x} \, d\Omega \qquad \mathbf{K}_{uv}^e = - \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial x} \tau_\phi \frac{\partial \mathbf{N}^e}{\partial y} \, d\Omega \\ \mathbf{K}_{v\phi}^e &= (1 - \tau_V) \int_{\Omega_e} \left[\mathbf{N}^e\right]^T \frac{\partial \mathbf{N}^e}{\partial y} \, d\Omega \qquad \mathbf{K}_{vu}^e = - \int_{\Omega_e} \frac{\partial \left[\mathbf{N}^e\right]^T}{\partial y} \tau_\phi \frac{\partial \mathbf{N}^e}{\partial x} \, d\Omega. \end{aligned} \tag{4.18}$$

With this there is only one thing remaining to complete the formulation of the problem, namely, to account for the Dirichlet boundary conditions. Proceeding as in section 2.4.1., we define the element Dirichlet vector for each variable as $\phi_D^e = \left[\phi_{D1}^e, \phi_{D2}^e, \phi_{D3}^e\right]^T$, $\mathbf{u}_D^e = \left[u_{D1}^e, u_{D2}^e, u_{D3}^e\right]^T$ and $\mathbf{v}_D^e = \left[v_{D1}^e, v_{D2}^e, v_{D3}^e\right]^T$, which can be grouped on a general vector $\mathbf{a}_D^e$ (in the same way as $\mathbf{a}^e$ in equation (4.17)) to obtain

$$\mathbf{K}^e \mathbf{a}^e = -\mathbf{K}^e \mathbf{a}_D^e. \tag{4.19}$$

The system of equations (4.19), when assembled in the usual way, gives the complete discrete form of the mixed finite element formulation for the two-dimensional incompressible potential flow problem.

In order to achieve our objective we need to compare the solution of the mixed problem with the one delivered by the classical formulation, also known as the irreductible form. The discretization of the irreductible problem can be found by the appropriate particularization of expression (2.24). Noting that now $\mathbf{C}^e = 0$, $\mathbf{f}^e = 0$, $k = 1$ and that $\mathbf{T}^e$ is replaced by ϕ^e, the elemental system of equations that governs the finite element discretization of the irreductible form becomes

$$\mathbf{K}^e_{irr}\phi^e = -\mathbf{K}^e_{irr}\phi^e_D \qquad \text{with} \qquad \mathbf{K}^e_{irr} = \int_{\Omega_e} [\nabla \mathbf{N}^e]^T \nabla \mathbf{N}^e \, d\Omega, \tag{4.20}$$

where the subscript irr (denoting irreductible) has been added to the diffusion matrix to differentiate it against the block matrix $\mathbf{K}^e$ of the mixed form.

Comparing both systems, (4.19) and (4.20), we see that for this 2D problem the mixed formulation requires solving a system that is three times larger than the irreductible one. However, in the process the mixed form also solves for the velocity, which in the irreductible case has to be computed as a postprocess to the global solution, where only the velocity potential is obtained. This can be viewed as another advantage of the mixed approach if the gradients of the transported quantity are of interest, which is usually the case.

For calculating the velocity with the classical formulation, a simple method has been considered. Once (4.20) is assembled and solved, the velocity potential is obtained at each mesh node. This allows its gradient, i.e. the velocity, to be approximated in each element by means of the shape functions as (recall equation (2.10)) $\nabla\phi^e(x,y) = \sum_{j=1}^{n} \nabla N^e_j(x,y)\,\phi^e_j$. However, a given node i is usually shared by various elements, depending on the mesh and element shape, and then the actual gradient at the node has to be calculated as an average of the nodal gradient computed in each element, that is,

$$\mathbf{V}_i = \frac{1}{n_{neigh}} \sum_{e=1}^{n_{neigh}} \nabla \phi^e|_i, \tag{4.21}$$

where $\mathbf{V}_i$ is the velocity at the considered node and n_{neigh} denotes the number of elements that share node i, called neighbors. In this expression a simple arithmetic average has been taken. Although an area weighted average should improve the accuracy of the method, the numerical results obtained in our case show that the difference is very small.

We now have all the necessary ingredients to obtain the solution of the problem and compare both formulations. Two different geometries have been tested, namely, a circular cylinder and a symmetric airfoil (model NACA 0012 [26]) at zero degrees of angle of attack (see [25] for more examples). The exact solution is only available for the cylinder geometry (consult reference [2] for the details), so we have focused primarily on the validation of such case. Both problems have been implemented and solved by using the multiphysics finite element software `Kratos` (refer to the introduction of this document for more information). The author source codes can be found in appendix A.

The solution of the incompressible potential flow past a circular cylinder is a well known problem characterized by the presence of strong velocity gradients at the cylinder surface. The important points to consider are shown in figure 4.3. Points 1 and 3 are stagnation

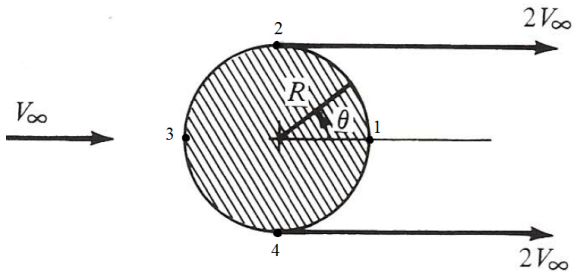


Figure 4.3: Circular cylinder in a freestream. The stagnation and maximum velocity points are identified with numbers, with the corresponding theoretical values of maximum velocity. The cylindrical coordinate θ is also shown.

points, hence, velocity is theoretically zero on them. On the contrary, points 2 and 4 are where the maximum velocity is encountered, which, as exposed in the figure corresponds to a value of $V_{max} = 2V_\infty$, with V_∞ being the freestream velocity. The geometry that has been chosen for the validation test can be seen in figure 4.4. We have considered a circular cylinder of unit radius inside a rectangular domain, with the boundaries chosen enough far away to ensure that they do not interfere with the velocity distribution around the cylinder. As can be observed, a mesh of linear triangular elements is used to discretize the computational domain, with a light refinement near the surface of the cylinder for a proper capture of the desired physics. It has been obtained by means of a pre and postprocessing software called `GiD`, also developed by CIMNE (refer to [29] for further information). In order to produce a freestream, the velocity potential is prescribed on the inlet and outlet boundaries of the domain using the results provided by the exact solution. For simplification, a dimensionless freestream velocity of $V_\infty = 1$ is used.

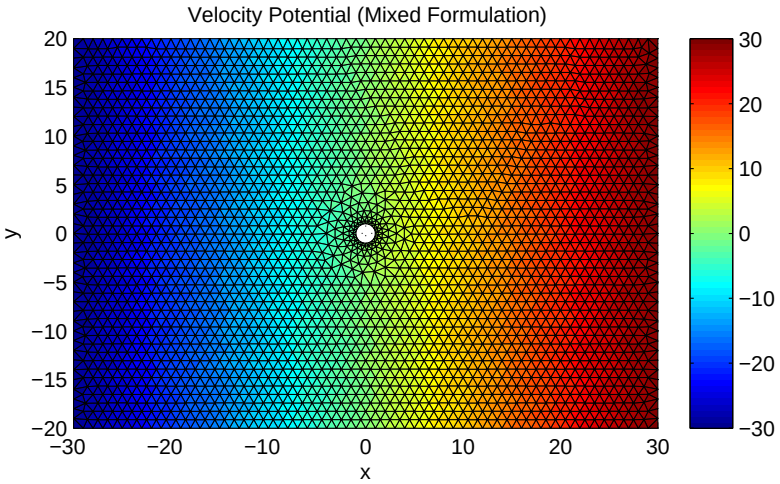


Figure 4.4: Meshed computational domain for the incompressible potential flow problem over a circular cylinder. Colors show the velocity potential solution obtained with the mixed formulation.

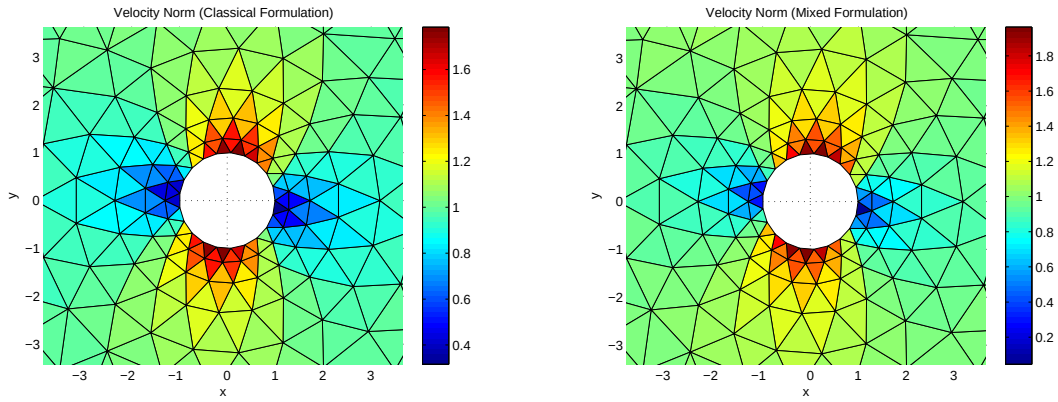
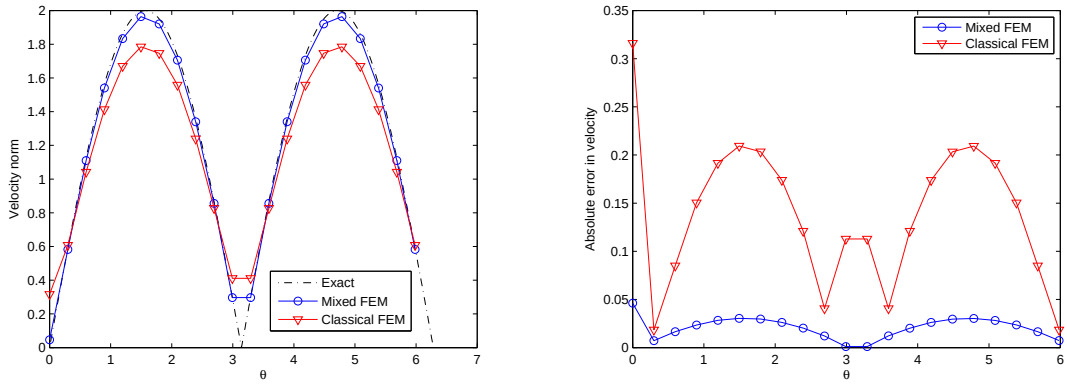


Figure 4.5: Magnitude of the velocity field in the region next to the cylinder surface as obtained by the classical (left) and the mixed (right) finite element solutions. The average value inside each element is represented.



(a) (b)

Figure 4.6: Comparison of the velocity distribution along the cylinder surface (left) and its absolute error (right) against the exact solution for both formulations.

With these conditions, the mixed form produces the velocity potential distribution displayed in figure 4.4. Observe that a very smooth and highly linear field is obtained and that symmetry is present due to the geometry of the problem. Although the disturbances produced by the cylinder are inappreciable, they indeed cause intense velocity gradients. In fact, attending to the velocity results, which are illustrated in figure 4.5 for the region of interest (that is, near the cylinder surface), we see that both the irreductible and the mixed formulations deliver very similar velocity fields. It can be easily noticed that the results agree with the theoretical values expected, approximating to 0 at the stagnation points and to 2 at the maximum velocity points. However, it must be emphasized that for the mixed formulation the values at those key points are closer to the theoretical ones. This is reflected much better in figure 4.6(a), where the numerical velocity distributions along the cylinder surface are compared against the exact one. The θ coordinate on the x-axis defines the angular position along the cylinder surface as represented in figure 4.3. Hence, point 1 is located at $\theta = 0$, point 2 at $\theta = \pi/2$, etc. Note how the mixed solution approximates the exact distribution much better than the classical (irreductible) one. Actually, looking at figure 4.6(b) we find that the absolute error for the mixed form is smaller than the one for the classical

form at all points along the cylinder surface, being approximately an order of magnitude smaller. Moreover, the largest differences in the error are encountered at $\theta = 0,\ \pi/2$ and $3\pi/2$, which respectively correspond to the location of the right stagnation point (where there is a mesh point) and the upper and lower maximum velocity points. This is not the case for the left stagnation point simply because as can be observed in figure 4.5 there is not a mesh point as closer as in the other cases, but it is not an exception as can be shown by using a finer mesh.

In view of the results already described, we can conclude that for this problem the mixed formulation succeeds on improving the accuracy with which the transported quantity gradients are computed, specially at those points where they are stronger. This corroborates the arguments given in the previous section regarding the improvements that should be expected when using the mixed approach in pure diffusion problems.

To finish with, we will present the solution for the other geometry we have considered testing, namely, a NACA 0012 airfoil of unit chord oriented parallel to the freestream (no angle of attack). The airfoil shape can be viewed in figure 4.7, which represents only a portion of the complete domain that has been used in the calculations, whose dimensions are $x = [-10,\ 11]$ and $y = [-5,\ 5]$. As can be noticed, the mesh has been considerably refined on the airfoil surface. This is because this geometry is more streamlined than the circular cylinder and obtaining a good level of accuracy demands smaller elements. In this case we do not have an exact analytical solution for comparison. However, since the airfoil has no angle of attack, we know that there will be two stagnation points located exactly at the airfoil's leading and trailing edges, which by looking at the figure are found to be located respectively at points $(0,\ 0)$ and $(1,\ 0)$.

The Dirichlet conditions on the velocity potential have been chosen to produce a dimensionless freestream velocity of 50. Figure 4.7 illustrates the velocity magnitude obtained with the mixed form at and near the airfoil surface. At the leading edge stagnation point a very small value is obtained, as expected, but at the trailing edge the resulting velocity value is inaccurate. This issue is also obtained with the classical formulation and is not a deficiency of the mixed approach. It is due to the fact that the trailing edge of the airfoil geometry is a singular point, and thus its discretization requires a special treatment in order to achieve accurate results, which has not been taken into account for our purposes. How-

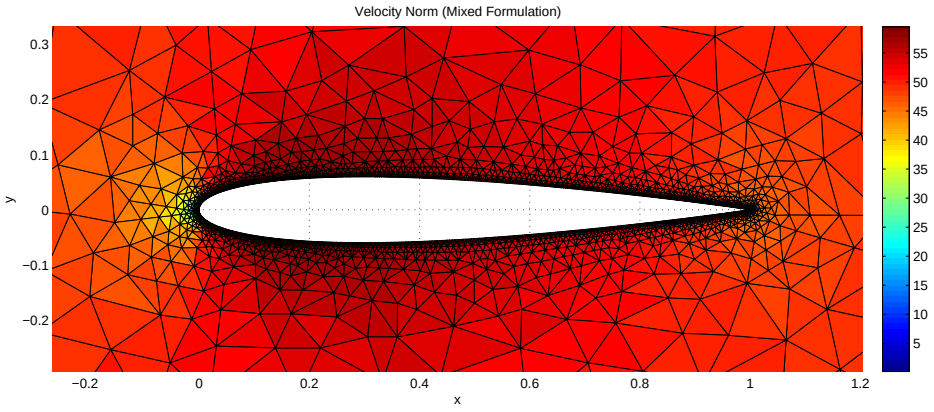


Figure 4.7: Magnitude of the velocity field in the region near the airfoil surface obtained with the mixed finite element formulation.

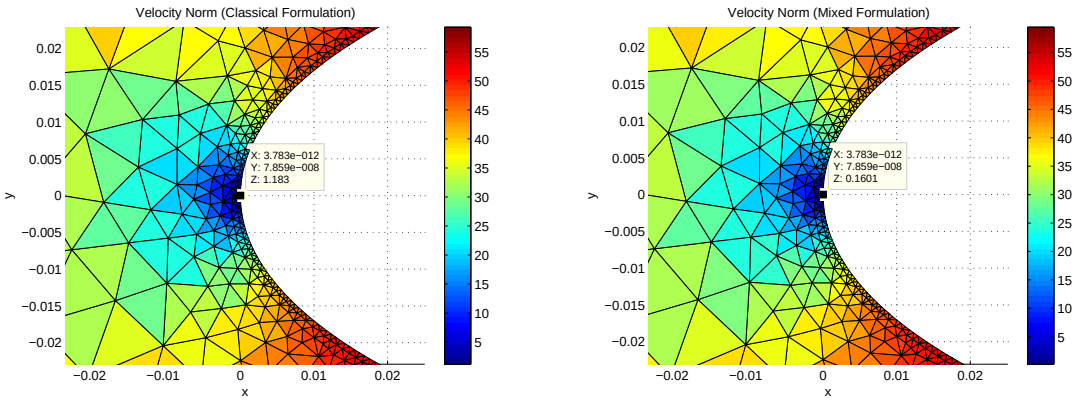


Figure 4.8: Magnitude of the velocity field at the airfoil's leading edge stagnation point calculated with the classical (left) and the mixed (right) finite element methods. The numerical value is given by Z.

ever, numerical results show that with the mixed form a smaller velocity value is obtained than with the irreductible case at such point.

Focusing on the leading edge stagnation point, figure 4.8 displays the velocity value obtained at such point with both formulations. Comparing the values, we see once again that the error of the mixed formulation is much lower than that of the classical approach, i.e. 0.160 in front of 1.18, demonstrating that velocity is computed with increased accuracy for this problem too.

CONCLUSIONS

The most important results in this work have been emphasized in the respective chapters. In summary, the following conclusions have been obtained.

Due to its close relation with the incompressible Navier-Stokes equations, the general convection-diffusion equation is found to be a very good representative model problem for computational fluid dynamics. It has been shown that its numerical solution involves the mathematical treatment of convection and diffusion physical processes, which play a vital role in fluid dynamics problems. Moreover, attending to its simplified structure, the study of the numerical solution to the convection-diffusion equation can be considered the first step in the development of adequate numerical methods for solving fluid flow problems.

When applying the standard finite element method to the solution of the one-dimensional steady transport problem, it is observed that when convective effects are important in the presence of boundary layers the Galerkin formulation lacks sufficient spatial stability and presents non-physical oscillations. An algebraic analysis of its truncation error shows that such instabilities are caused by a not proper discretization of the convective term, which appears as a negative non-physical numerical diffusion introduced in the original governing equation. Then, accounting for such error by means of a balancing diffusion, an upwind-type discretization of the convection term is encountered, where an artificial diffusion is automatically added along the streamlines of the flow. Numerical tests directly show that with this modification the spurious oscillations are completely eliminated, obtaining the exact solution at the nodes. This provides a justification for the formulation and usage of the stabilization techniques known as streamline-upwind (SU) schemes. When consistently applied, these schemes lead to the well-known SUPG stabilization method. Furthermore, it has been shown that the use of the multiscale philosophy introduced by the SGS method produces a more general stabilization framework, from which the SUPG formulation is derived as a particular case.

Attending to the new mixed formulation, the original proposal has been described. It is shown by theoretical arguments that for this new formulation the order of magnitude of the error in the solution and its gradients can be reduced at a reasonable increase in the computational cost. However, numerical tests demonstrate that the original formulation, although being stabilized by the SGS method, presents oscillatory results. By a modification on the formulation of the convective term it is then found that the cause of the instabilities is the mixed discretization of the convection operator, and that as a result the advantages of the proposed method are for the moment restricted to pure diffusion problems. The implementation and testing of the pure diffusion formulation is done by means of the solution of a particular fluid dynamics problem that coincides with a pure diffusive transport equation, namely, the incompressible potential flow. The results in this case show that the mixed formulation is capable of providing an important increase in accuracy, specially in the regions where the gradients are stronger, as demonstrated by the velocity calculations at the stagnation points of a circular cylinder and a symmetric airfoil. In this case, the original expectations regarding the new formulation are satisfied, and provide promising results for considering future research that could lead to an improvement of the formulation to make its advantages applicable to general convection-diffusion systems, and eventually to the governing equations of fluid dynamics, which is its ultimate goal.

BIBLIOGRAPHY

[1] J.D. Anderson Jr. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill, New York, 1995. 2, 5, 6, 23, 26

[2] J.D. Anderson Jr. *Fundamentals of Aerodynamics*. McGraw-Hill, New York, fifth edition, 2011. 5, 6, 40, 44

[3] J. Donea and A. Huerta. *Finite Element Methods for Flow Problems*. John Wiley & Sons, Chichester, 2003. ix, 2, 7, 10, 12, 19, 23, 26, 27, 28, 32, 34, 39

[4] C. Hirsch. *Numerical Computation of Internal and External Flows*, volume 1: "Fundamentals of Computational Fluid Dynamics", chapter 5: "Finite Volume Method and Conservative Discretization with an Introduction to Finite Element Method", pages 203–248. Butterworth-Heinemann, Oxford, second edition, 2007. 2

[5] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*, volume 1: The Basis. Butterworth-Heinemann, Oxford, fifth edition, 2000. 2, 12, 13, 16, 17, 23, 35, 36, 37

[6] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*, volume 3: Fluid Dynamics. Butterworth-Heinemann, Oxford, fifth edition, 2000. 2

[7] C. Farhat. "Representative Model Problems". http://web.stanford.edu/group/frg/course_work/AA214B/CA-AA214B-Ch5.pdf, 2013. Standford University Lecture. Course AA214B: Numerical Methods for Compressible Flows. 7

[8] E. Oñate and F. Zárate. "Introducción al Método de los Elementos Finitos". Curso de Máster en Métodos Numéricos para Cálculo y Diseño en Ingeniería. Apuntes de la asignatura. ix, 13, 15, 16

[9] E. Oñate and F. Zárate. "Transmisión de Calor". Curso de Máster en Métodos Numéricos para Cálculo y Diseño en Ingeniería. Apuntes de la asignatura. ix, 21

[10] T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987. 2, 13, 14

[11] A.N. Brooks and T.J.R. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations". *Comput. Methods Appl. Mech. Eng.*, 32(1–3):199–259, 1982. 19, 22, 26, 29

[12] T.J.R. Hughes and A.N. Brooks. "A multidimensional upwind scheme with no crosswind diffusion". In T.J.R. Hughes, editor, *Finite element methods for convection dominated flows*, volume 34 of *AMD*, pages 19–35. Amer. Soc. Mech. Engrs. (ASME), New York, 1979. 30

[13] T.J.R. Hughes and A.N. Brooks. "A theoretical framework for Petrov-Galerkin methods with discontinuous weighting functions: application to the streamline-upwind procedure". In R.H. Gallagher, D.H. Norrie, J.T. Oden, and O.C. Zienkiewicz, editors, *Finite Elements in Fluids*, volume 4, pages 47–65. John Wiley & Sons, New York, 1892. 30, 31

[14] T.J.R. Hughes. “Multiscale phenomena: Green’s functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods”. *Comput. Methods Appl. Mech. Eng.*, 127(1–4):387–401, 1995. 32

[15] T.J.R. Hughes, G.R. Feijóo, L. Mazzei, and J. Quincy. “The variational multiscale method - a paradigm for computational mechanics”. *Comput. Methods Appl. Mech. Eng.*, 166(1–2):3–24, 1998. 32, 33

[16] R. Codina. “Comparison of some finite element methods for solving the diffusion-convection-reaction equation”. *Comput. Methods Appl. Mech. Eng.*, 156(1–4):185–210, 1998. 32

[17] R. Codina. “On stabilized finite element methods for linear systems of convection-diffusion-reaction equations”. *Comput. Methods Appl. Mech. Eng.*, 188(1–3):61–82, 2000. 32, 34

[18] F. Shakib, T.J.R. Hughes, and Z. Johan. “A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations”. *Comput. Methods Appl. Mech. Eng.*, 89(1–3):141–219, 1991. 32

[19] A. Soulaïmani and M. Fortin. “Finite element solution of compressible viscous flows using conservative variables”. *Comput. Methods Appl. Mech. Eng.*, 118(3–4):319–350, 1994. 32

[20] T.J.R. Hughes and T.E. Tezduyar. “Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations”. *Comput. Methods Appl. Mech. Eng.*, 45(1–3):217–284, 1984. 31

[21] T.E. Tezduyar and T.J.R. Hughes. “Finite element formulations for convection dominated flows with particular emphasis on the compressible Euler equations”. Technical Report 83-0125, AIAA, 1983. Selected paper from the the AIAA 21st Aerospace Sciences Meeting, Reno, Nevada, January 10-13, 1983. 31

[22] T.E. Tezduyar and D.K. Ganjoo. “Petrov-Galerkin formulations with weighting functions dependent upon spatial and temporal discretization: applications to transient convection-diffusion problems”. *Comput. Methods Appl. Mech. Eng.*, 59(1):49–71, 1986. 32

[23] M.K. Esfahani. *A Contribution to the Finite Element Analysis of High-Speed Compressible Flows and Aerodynamic Shape Optimization*. PhD thesis, Universitat Politécnica de Catalunya. Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE), 2013. 2

[24] R. Rossi. “Convection diffusion in mixed form”, 2014. Initial proposal and definition of the new mixed formulation. 3, 36

[25] C.V. Flores et al. “Numerical Simulation of Potential Flow using the Finite Element Method”. California State University, 2007. 44

[26] P. Marzocca. The NACA airfoil series, 2009. Clarkson University. 44

[27] P. Dadvand, R. Rossi, and E. Oñate. “An object-oriented environment for developing finite element codes for multi-disciplinary applications”. *Archives of Computational Methods in Engineering*, 17:253–297, 2010. 3

[28] Kratos Home Page. http://www.cimne.com/kratos/. Accessed: 08-07-2014. 3

[29] GiD Home Page. http://www.gidhome.com/. Accessed: 08-07-2014. 45

[30] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

APPENDICES

APPENDIX A. SOURCE CODES

This appendix collects the source codes from which the numerical results presented along this document have been obtained. They are the finite element implementations of the problems described.

1) MATLAB code for the finite element solution of the one-dimensional steady transport problem:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FINITE ELEMENT SOLUTION OF THE 1D STEADY TRANSPORT PROBLEM %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clc
clear all

% Iván Padilla Montero, June 2014.

L = 1; % Domain length

n_el = 10; % Number of elements in the mesh
n_nod = n_el + 1; % Number of nodes in the mesh

% We use a uniform mesh of linear elements
h = L / n_el; % Element length (size). Also known as mesh size.

nodes = (0 : h : L)'; % Only the x coordinate is stored

elements = zeros(n_el, 2);
for i = 1:n_el
   elements(i, :) = [i, i + 1];
end

source_term = true; % Boolean to now if the problem has a source term
% or not.

if (source_term)
    s = 1;
    dirichlet_nodes = [1; n_nod];
    dirichlet_values = [0; 0];
else
    s = 0;
    dirichlet_nodes = [1; n_nod];
    dirichlet_values = [0; 1];
end

% Since the mesh is uniform, all the elements are equal. As a result,
% the shape functions evaluated at the integration points are all
% equal. For this problem we need integration order 1. We are not
% discretizingvspatially the source term since it is uniform.
```

```
% For integration order 1 the integration point is located at xi = 0:
n_int = 1; % Number of integration points for order 1
N_Lin = [1/2, 1/2]; % Shape functions for each node of the element
% evaluated at the linear integration point.
weight = 2; % Integration weight for order 1
jacobian_det = h/2; % Determinant of the jacobian from the
% transformation of cartesian coordinates into natural coordinates.

% Moreover, the shape function derivatives are defined by:
dN_dx = [-1/h, 1/h];

% Problem parametes
%Pe = 0.9; % Mesh Peclet number
conv_vel = ones(n_nod, 1); % Convection velocity field
conv_vel_norm = 1; % Norm of the convection velocity
Pe = [0.5, 5];
for iter = 1:length(Pe)
k = conv_vel_norm * h / (2 * Pe(iter)); % Diffusivity

% Stabilization parameter. Superconvergence formulation, i.e. the
% optimal stabilization parameter that gives exact nodal results for
% any value of Pe on a uniform mesh in 1D.
tau = h / (2 * conv_vel_norm) * (coth(Pe(iter)) - 1/Pe(iter));

% Now we can compute the system and solve it
% Initialize global system variables
LHS = zeros(n_nod);
RHS = zeros(n_nod, 1);
phi = zeros(n_nod, 1);

% Loop in elements
for el = 1:n_el
    LHS_local = zeros(2);
    RHS_local = zeros(2, 1);
    dirichlet_local = zeros(2, 1);

    % Compute the element (local) contribution to the global system
    conv_vel_local = conv_vel(elements(el, :));
    % Loop in integration points
    for int = 1:n_int
        N = N_Lin(int, :); % Select the row of shape functions
        % corresponding to the current integration point.

        % Double loop in element nodes
        for i = 1:2
            for j = 1:2
                LHS_local(i, j) = LHS_local(i, j) + weight *
                jacobian_det * ((N(i) + tau * conv_vel_local(i) *
                dN_dx(i)) * conv_vel_local(i) * dN_dx(j) + dN_dx(i) *
                k * dN_dx(j));
            end
```

```
            RHS_local(i) = RHS_local(i) + weight * jacobian_det *
            (N(i) + tau * conv_vel_local(i) * dN_dx(i)) * s;
        end
    end

    % Finally, substract the Dirichlet term from the RHS
    for i = 1:2
        if any(dirichlet_nodes == elements(el, i))
            dirichlet_local(i) = dirichlet_values(
            find(dirichlet_nodes == elements(el, i)));
        end
    end
    RHS_local = RHS_local - LHS_local * dirichlet_local;

    % Assemble the element contribution to the global system
    for i = 1:2
        for j = 1:2
            LHS(elements(el, i), elements(el, j)) =
            LHS(elements(el, i), elements(el, j)) + LHS_local(i, j);
        end
        RHS(elements(el, i)) = RHS(elements(el, i)) + RHS_local(i);
    end
end

% Solve the system of equations to obtain the unknowns
% The rows and columns corresponding to the Dirichlet nodes are
% eliminated from the system
unknown_nodes = setdiff((1:n_nod)', dirichlet_nodes);
phi_unknown = LHS(unknown_nodes, unknown_nodes) \ RHS(unknown_nodes);
phi(unknown_nodes) = phi_unknown;
phi(dirichlet_nodes) = dirichlet_values;

plot(nodes, phi, 'o-')

% Compute exact solution
x = 0:1/999:1;
gamma = conv_vel_norm / k;
if (source_term)
    exact_sol = 1/conv_vel_norm * (x - (1 - exp(gamma * x)) /
    (1 - exp(gamma)));
else
    exact_sol = (1 - exp(gamma * x)) / (1 - exp(gamma));
end
hold on
plot(x, exact_sol, 'k-.')
end
legend('SU FEM', 'Exact', 'Location', 'NorthWest')
xlabel('x')
ylabel('T')
```

2) MATLAB code for the original mixed finite element solution of the one-dimensional steady transport problem:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MIXED FINITE ELEMENT SOLUTION OF THE 1D STEADY TRANSPORT PROBLEM %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clc
clear all

% Ivan Padilla Montero, June 2014.

% Original formulation.

L = 1; % Domain length

n_el = 10; % Number of elements in the mesh
n_nod = n_el + 1; % Number of nodes in the mesh

% We use a uniform mesh of linear elements
h = L / n_el; % Element length (size). Also known as mesh size.

nodes = (0 : h : L)'; % Only the x coordinate is stored

elements = zeros(n_el, 2);
for i = 1:n_el
   elements(i, :) = [i, i + 1];
end

source_term = true; % Boolean to now if the problem has a source term
% or not.

if (source_term)
    s = ones(n_nod, 1);
    dirichlet_nodes_phi = [1; n_nod];
    dirichlet_phi = [0; 0];
else
    s = zeros(n_nod, 1);
    dirichlet_nodes_phi = [1; n_nod];
    dirichlet_phi = [0; 1];
end

% Since the mesh is uniform, all the elements are equal. As a result,
% the shape functions evaluated at the integration points are all
% equal. For this problem we need integration order 2. We are not
% discretizing
% spatially the source term since it is uniform.
% For integration order 2 the integration points are located at xi =
% +-0.5773502692, thus:
n_int = 2; % Number of integration points for order 2
```

```
xi_int = [sqrt(1/3), -sqrt(1/3)]; % Coordinates of the integration
%points for order 2
N_Quad = [1/2 * (1 - xi_int(1)), 1/2 * (1 + xi_int(1)); 1/2 *
    (1 - xi_int(2)), 1/2 * (1 + xi_int(2))]; % Shape functions for
% each element node evaluated at each quadratic integration point.
weight = ones(2, 1); % Integration weights for order 2
jacobian_det = h/2; % Determinant of the jacobian from the
% transformation of natural coordinates into cartesian coordinates.

% Moreover, the shape function derivatives are defined by:
dN_dx = [-1/h, 1/h]; % In this case they are constants and they have
% the same value in all the integration points.

% Problem parametes
%Pe = 5; % Mesh Peclet number
conv_vel = ones(n_nod, 1); % Convection velocity field
conv_vel_norm = 1; % Norm of the convection velocity
Pe = [0.5, 5];
for iter = 1:length(Pe)
k = conv_vel_norm * h / (2 * Pe(iter)); % Diffusivity

tau_xi = 0.1; % Constant stabilization parameter
% Codina stabilization parameter.
tau_phi = 1 / (2 * conv_vel_norm / h + 4 * k / h ^ 2);

% Now we can compute the system and solve it
% Initialize global system variables
LHS = zeros(2 * n_nod);
RHS = zeros(2 * n_nod, 1);
dofs = zeros(2 * n_nod, 1);

% Loop in elements
for el = 1:n_el
    LHS_local = zeros(4);
    RHS_local = zeros(4, 1);
    dirichlet_local = zeros(4, 1);

    % Compute the element (local) contribution to the global system
    conv_vel_local = conv_vel(elements(el, :)); % Values of the
    % convection velocity at the element nodes
    s_local = s(elements(el, :));

    % Loop in integration points
    for i_int = 1:n_int
        N = N_Quad(i_int, :); % Select the row of shape functions
        %corresponding to the current integration point.

        conv_vel_local_i_int = N * conv_vel_local;
        s_local_i_int = N * s_local;

        LHS_local(1:2, 1:2) = LHS_local(1:2, 1:2) + weight(i_int) *
```

```
        jacobian_det * tau_xi * (N' * conv_vel_local_i_int * dN_dx +
        k * (dN_dx' * dN_dx));
        LHS_local(1:2, 3:4) = LHS_local(1:2, 3:4) + weight(i_int) *
        jacobian_det * ((1 - tau_xi) * N' * conv_vel_local_i_int *
        N - k * N' * dN_dx - tau_xi * k * dN_dx' * N);
        LHS_local(3:4, 1:2) = LHS_local(3:4, 1:2) + weight(i_int) *
        jacobian_det * (1 - tau_xi) * N' * dN_dx;
        LHS_local(3:4, 3:4) = LHS_local(3:4, 3:4) + weight(i_int) *
        jacobian_det * ((tau_xi - 1) * (N' * N) + tau_phi * dN_dx' *
        conv_vel_local_i_int * N - tau_phi * k * (dN_dx' * dN_dx));

        RHS_local(1:2) = RHS_local(1:2) + weight(i_int) *
        jacobian_det * N' * s_local_i_int;
        RHS_local(3:4) = RHS_local(3:4) + weight(i_int) *
        jacobian_det * tau_phi * dN_dx' * s_local_i_int;
    end

    % Finally, substract the Dirichlet term from the RHS
    for i = 1:2
        if any(dirichlet_nodes_phi == elements(el, i))
            dirichlet_local(i) = dirichlet_phi(
            find(dirichlet_nodes_phi == elements(el, i)));
        end
    end
    RHS_local = RHS_local - LHS_local * dirichlet_local;

    % Assemble the element contribution to the global system
    for i = 1:2
        for j = 1:2
            LHS(elements(el, i), elements(el, j)) = LHS(elements(el,
            i), elements(el, j)) + LHS_local(i, j); % Block 11
            LHS(elements(el, i), elements(el, j) + n_nod) = LHS(
            elements(el, i), elements(el, j) + n_nod) + LHS_local(i,
            j + 2); % Block 12

            LHS(elements(el, i) + n_nod, elements(el, j)) = LHS(
            elements(el, i) + n_nod, elements(el, j)) + LHS_local(i +
            2, j); % Block 21
            LHS(elements(el, i) + n_nod, elements(el, j) + n_nod) =
            LHS(elements(el, i) + n_nod, elements(el, j) + n_nod) +
            LHS_local(i + 2, j + 2); % Block 22
        end
        RHS(elements(el, i)) = RHS(elements(el, i)) + RHS_local(i);
        % Block 1
        RHS(elements(el, i) + n_nod) = RHS(elements(el, i) + n_nod)
        + RHS_local(i + 2); % Block 2
    end
end

% Solve the system of equations to obtain the unknowns
% The rows and columns corresponding to the Dirichlet nodes are
```

```
% eliminated from the system
unknown_nodes = setdiff((1:2 * n_nod)', dirichlet_nodes_phi);
dofs_unknown = LHS(unknown_nodes, unknown_nodes) \ RHS(unknown_nodes)
dofs(unknown_nodes) = dofs_unknown;
dofs(dirichlet_nodes_phi) = dirichlet_phi;
phi = dofs(1:n_nod);
xi_x = dofs(n_nod + 1:2 * n_nod);

figure(1)
plot(nodes, phi, 'o-')

figure(2)
plot(nodes, xi_x, 'o-')

% Compute exact solution
x = 0:1/999:1;
gamma = conv_vel_norm / k;
if (source_term)
    exact_sol = 1/conv_vel_norm * (x - (1 - exp(gamma * x)) / (1 -
    exp(gamma)));
    dexact_sol_dx = 1/conv_vel_norm * (1 - (1 - exp(gamma * x) *
    gamma) / (1 - exp(gamma)));
else
    exact_sol = (1 - exp(gamma * x)) / (1 - exp(gamma));
    dexact_sol_dx = - exp(gamma * x) * gamma / (1 - exp(gamma));
end
figure(1)
hold on
plot(x, exact_sol, 'k-.')
xlabel('x')
ylabel('T')
legend('ASGS Mixed FEM Original', 'Exact', 'Location', 'NorthWest')

figure(2)
hold on
plot(x, dexact_sol_dx, 'k-.')
xlabel('x')
ylabel('q')
legend('ASGS Mixed FEM Original', 'Exact', 'Location', 'NorthWest')
end
```

3) MATLAB code for the modified mixed finite element solution of the one-dimensional steady transport problem:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MIXED FINITE ELEMENT SOLUTION OF THE 1D STEADY TRANSPORT PROBLEM %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clc
clear all

% Ivan Padilla Montero, June 2014.

% Modified formulation

L = 1; % Domain length

n_el = 10; % Number of elements in the mesh
n_nod = n_el + 1; % Number of nodes in the mesh

% We use a uniform mesh of linear elements
h = L / n_el; % Element length (size). Also known as mesh size.

nodes = (0 : h : L)'; % Only the x coordinate is stored

elements = zeros(n_el, 2);
for i = 1:n_el
   elements(i, :) = [i, i + 1];
end

source_term = true; % Boolean to now if the problem has a source term
% or not.

if (source_term)
    s = ones(n_nod, 1);
    dirichlet_nodes_phi = [1; n_nod];
    dirichlet_phi = [0; 0];
else
    s = zeros(n_nod, 1);
    dirichlet_nodes_phi = [1; n_nod];
    dirichlet_phi = [0; 1];
end

% Since the mesh is uniform, all the elements are equal. As a result,
% the shape functions evaluated at the integration points are all
% equal. For this problem we need integration order 2. We are not
% discretizing spatially the source term since it is uniform.
% For integration order 2 the integration points are located at xi =
% +-0.5773502692, thus:
n_int = 2; % Number of integration points for order 2
xi_int = [sqrt(1/3), -sqrt(1/3)]; % Coordinates of the integration
```

```
% points for order 2
N_Quad = [1/2 * (1 - xi_int(1)), 1/2 * (1 + xi_int(1)); 1/2 *
    (1 - xi_int(2)), 1/2 * (1 + xi_int(2))]; % Shape functions for
%each element node evaluated at each quadratic integration point.
weight = ones(2, 1); % Integration weights for order 2
jacobian_det = h/2; % Determinant of the jacobian from the
% transformation of natural coordinates into cartesian coordinates.

% Moreover, the shape function derivatives are defined by:
dN_dx = [-1/h, 1/h]; % In this case they are constants and they have
% the same value in all the integration points.

% Problem parametes
%Pe = 0.5; % Mesh Peclet number
conv_vel = ones(n_nod, 1); % Convection velocity field
conv_vel_norm = 1; % Norm of the convection velocity
Pe = [0.5, 5];
for iter = 1:length(Pe)
k = conv_vel_norm * h / (2 * Pe(iter)); % Diffusivity

tau_xi = 0.1; % Constant stabilization parameter
% Codina stabilization parameter
tau_phi = 1 / (2 * conv_vel_norm / h + 4 * k / h ^ 2);

% Now we can compute the system and solve it
% Initialize global system variables
LHS = zeros(2 * n_nod);
RHS = zeros(2 * n_nod, 1);
dofs = zeros(2 * n_nod, 1);

% Loop in elements
for el = 1:n_el
    LHS_local = zeros(4);
    RHS_local = zeros(4, 1);
    dirichlet_local = zeros(4, 1);

    % Compute the element (local) contribution to the global system
    conv_vel_local = conv_vel(elements(el, :)); % Values of the
    % convection velocity at the element nodes
    s_local = s(elements(el, :));

    % Loop in integration points
    for i_int = 1:n_int
        N = N_Quad(i_int, :); % Select the row of shape functions
        %corresponding to the current integration point.

        conv_vel_local_i_int = N * conv_vel_local;
        s_local_i_int = N * s_local;

        LHS_local(1:2, 1:2) = LHS_local(1:2, 1:2) + weight(i_int) *
        jacobian_det * ((N' + tau_phi * conv_vel_local_i_int * dN_dx'
```

```
        ) * conv_vel_local_i_int * dN_dx + tau_xi * k * (dN_dx' *
        dN_dx));
        LHS_local(1:2, 3:4) = LHS_local(1:2, 3:4) - weight(i_int) *
        jacobian_det * ((N' + tau_phi * conv_vel_local_i_int * dN_dx'
        ) * k * dN_dx + tau_xi * k * dN_dx' * N);
        LHS_local(3:4, 1:2) = LHS_local(3:4, 1:2) + weight(i_int) *
        jacobian_det * ((1 - tau_xi) * N' * dN_dx + tau_phi * dN_dx'
        * conv_vel_local_i_int * dN_dx);
        LHS_local(3:4, 3:4) = LHS_local(3:4, 3:4) + weight(i_int) *
        jacobian_det * ((tau_xi - 1) * (N' * N) - tau_phi * k *
        (dN_dx' * dN_dx));

        RHS_local(1:2) = RHS_local(1:2) + weight(i_int) *
        jacobian_det * (N' + tau_phi * conv_vel_local_i_int
        * dN_dx') * s_local_i_int;
        RHS_local(3:4) = RHS_local(3:4) + weight(i_int) *
        jacobian_det * tau_phi * dN_dx' * s_local_i_int;
    end

    % Finally, substract the Dirichlet term from the RHS
    for i = 1:2
        if any(dirichlet_nodes_phi == elements(el, i))
            dirichlet_local(i) = dirichlet_phi(find(
            dirichlet_nodes_phi == elements(el, i)));
        end
    end
    RHS_local = RHS_local - LHS_local * dirichlet_local;

    % Assemble the element contribution to the global system
    for i = 1:2
        for j = 1:2
            LHS(elements(el, i), elements(el, j)) = LHS(elements
            (el, i), elements(el, j)) + LHS_local(i, j); % Block 11
            LHS(elements(el, i), elements(el, j) + n_nod) = LHS
            (elements(el, i), elements(el, j) + n_nod) + LHS_local
            (i, j + 2); % Block 12

            LHS(elements(el, i) + n_nod, elements(el, j)) = LHS(
            elements(el, i) + n_nod, elements(el, j)) + LHS_local
            (i + 2, j); % Block 21
            LHS(elements(el, i) + n_nod, elements(el, j) + n_nod) =
            LHS(elements(el, i) + n_nod, elements(el, j) + n_nod) +
            LHS_local(i + 2, j + 2); % Block 22
        end
        RHS(elements(el, i)) = RHS(elements(el, i)) + RHS_local(i);
        % Block 1
        RHS(elements(el, i) + n_nod) = RHS(elements(el, i) + n_nod)
        + RHS_local(i + 2); % Block 2
    end
end
```

```
% Solve the system of equations to obtain the unknowns
% The rows and columns corresponding to the Dirichlet nodes are
% eliminated from the system
unknown_nodes = setdiff((1:2 * n_nod)', dirichlet_nodes_phi);
dofs_unknown = LHS(unknown_nodes, unknown_nodes) \ RHS(unknown_nodes)
dofs(unknown_nodes) = dofs_unknown;
dofs(dirichlet_nodes_phi) = dirichlet_phi;
phi = dofs(1:n_nod);
xi_x = dofs(n_nod + 1:2 * n_nod);

figure(1)
plot(nodes, phi, 'o-')

figure(2)
plot(nodes, xi_x, 'o-')

% Compute exact solution
x = 0:1/999:1;
gamma = conv_vel_norm / k;
if (source_term)
    exact_sol = 1/conv_vel_norm * (x - (1 - exp(gamma * x)) / (1 -
    exp(gamma)));
    dexact_sol_dx = 1/conv_vel_norm * (1 - (1 - exp(gamma * x) *
    gamma) / (1 - exp(gamma)));
else
    exact_sol = (1 - exp(gamma * x)) / (1 - exp(gamma));
    dexact_sol_dx = - exp(gamma * x) * gamma / (1 - exp(gamma));
end
figure(1)
hold on
plot(x, exact_sol, 'k-.')
xlabel('x')
ylabel('T')
legend('ASGS Mixed FEM Modified', 'Exact', 'Location', 'NorthWest')

figure(2)
hold on
plot(x, dexact_sol_dx, 'k-.')
xlabel('x')
ylabel('q')
legend('ASGS Mixed FEM Modified', 'Exact', 'Location', 'NorthWest')
end
```

4) C++ code for the classical finite element solution of the incompressible potential flow using Kratos:

```
//   Project Name:        Convection-Diffusion in Irreductible Form
// with Sub-Grid Scale Stabilization
//   Last Modified by:    $Author: Ivan Padilla $
//   Date:                $Date: 09/06/2014 $
//   Revision:            $Revision: 1.0 $


// System includes

// External includes

// Project includes
#include "convection_diffusion_irreductible_application.h"
#include "utilities/geometry_utilities.h" // Needed for the
// calculation of geometry data

namespace Kratos
{
    // Constructors
    IncompressiblePotentialFlow2D::IncompressiblePotentialFlow2D(
        IndexType NewId, GeometryType::Pointer pGeometry)
        : Element(NewId, pGeometry)
    {}

    IncompressiblePotentialFlow2D::IncompressiblePotentialFlow2D(
        IndexType NewId, GeometryType::Pointer pGeometry, PropertiesType:
        :Pointer pProperties)
        : Element(NewId, pGeometry, pProperties)
    {}

    // Pointer
    Element::Pointer IncompressiblePotentialFlow2D::Create(IndexType
        NewId, NodesArrayType const& ThisNodes, PropertiesType::Pointer
        pProperties) const
    {
        return Element::Pointer(new IncompressiblePotentialFlow2D(NewId,
        GetGeometry().Create(ThisNodes), pProperties));
    }

    // Destructor
    IncompressiblePotentialFlow2D::~IncompressiblePotentialFlow2D()
    {}

    // Methods
    void IncompressiblePotentialFlow2D::CalculateLocalSystem(
        MatrixType& rLeftHandSideMatrix, VectorType& rRightHandSideVector
        , ProcessInfo& rCurrentProcessInfo)
    {
```

```
KRATOS_TRY

unsigned int number_of_nodes = GetGeometry().size(); // Number of
nodes of the element (in this case linear triangular elements).

boost::numeric::ublas::bounded_matrix<double, 3, 2> DN_DX;
// Element gradient matrix (3 rows --> nodes of the element),
// 2 columns (2D --> x and y coordinates) --> [dN1_dx dN1_dy;
// dN2_dx dN2_dy; dN3_dx dN3_dy]. Constant over the element.
array_1d<double, 3> N; // Vector of the shape functions of each
// node evaluated at the linear Gauss point (triangle baricenter).
double Area; // Element area
array_1d<double, 3> Dirichlet_Vector = ZeroVector(3); // Element
// vector for Dirichlet conditions (if any).

// First of all we resize the LHS and RHS as needed
if(rLeftHandSideMatrix.size1() != number_of_nodes)
{
    rLeftHandSideMatrix.resize(number_of_nodes, number_of_nodes,
        false);
}
rLeftHandSideMatrix = ZeroMatrix(number_of_nodes, number_of_nodes);

if(rRightHandSideVector.size() != number_of_nodes)
{
    rRightHandSideVector.resize(number_of_nodes, false);
}
rRightHandSideVector = ZeroVector(number_of_nodes);

// Compute geometry data for the current element
GeometryUtils::CalculateGeometryData(GetGeometry(), DN_DX, N, Area);

// Compute the element LHS matrix (size 3x3)
rLeftHandSideMatrix = prod(DN_DX, trans(DN_DX));
rLeftHandSideMatrix *= Area; // Integration over the element.
// Since DN_DX is constant along the element, the integration
// results in a simple area multiplication.

/*if (this->Id() == 7353)
    {KRATOS_WATCH(this->Id())
    KRATOS_WATCH(2.0*Area)
    KRATOS_WATCH(DN_DX)
    KRATOS_WATCH(rLeftHandSideMatrix)
}*/

// Substract the Dirichlet term from the RHS
// (RHS -= LHS*Dirichlet_Vector)
for (unsigned int i = 0; i < number_of_nodes ; i++)
{
    Dirichlet_Vector[i] = GetGeometry()[i].
        FastGetSolutionStepValue(VELOCITY_POTENTIAL);
```

```
        }

        rRightHandSideVector -= prod(rLeftHandSideMatrix, Dirichlet_Vector);

        KRATOS_CATCH("")
    }

    void IncompressiblePotentialFlow2D::EquationIdVector(
        EquationIdVectorType& rResult, ProcessInfo& rCurrentProcessInfo)
    {
        unsigned int number_of_nodes = GetGeometry().PointsNumber();
        // In this case triangular elements with 3 nodes.

        if(rResult.size() != number_of_nodes)
        {
            rResult.resize(number_of_nodes, false);
        }

        // In this case the element unknowns vector will have the following
        // structure:
        // rResult = [phi_1; phi_2; phi_3];

        for (unsigned int i = 0; i < number_of_nodes; i++)
        {
            rResult[i] = GetGeometry()[i].GetDof(VELOCITY_POTENTIAL).
                EquationId();
        }
    }

    void IncompressiblePotentialFlow2D::GetDofList(DofsVectorType&
        rElementalDofList, ProcessInfo& rCurrentProcessInfo)
    {
        unsigned int number_of_nodes = GetGeometry().PointsNumber();

        if(rElementalDofList.size() != number_of_nodes)
        {
            rElementalDofList.resize(number_of_nodes);
        }

        for (unsigned int i = 0; i < number_of_nodes; i++)
        {
            rElementalDofList[i] = GetGeometry()[i].
                pGetDof(VELOCITY_POTENTIAL);
        }
    }
} // namespace Kratos
```

5) C++ code for the mixed finite element solution of the incompressible potential flow using Kratos:

```
//   Project Name:        Convection-Diffusion in Mixed Form with
// Sub-Grid Scale Stabilization
//   Last Modified by:    $Author: Ivan Padilla $
//   Date:                $Date: 09/06/2014 $
//   Revision:            $Revision: 1.0 $


// System includes

// External includes

// Project includes
#include "convection_diffusion_mixed_application.h"
#include "utilities/geometry_utilities.h" // Needed for the
// calculation of geometry data.

namespace Kratos
{
    // Constructors
    IncompressiblePotentialFlow2DMixed::
        IncompressiblePotentialFlow2DMixed(IndexType NewId, GeometryType:
        :Pointer pGeometry)
        : Element(NewId, pGeometry)
    {}

    IncompressiblePotentialFlow2DMixed::
        IncompressiblePotentialFlow2DMixed(IndexType NewId, GeometryType:
        :Pointer pGeometry, PropertiesType::Pointer pProperties)
        : Element(NewId, pGeometry, pProperties)
    {}

    // Pointer
    Element::Pointer IncompressiblePotentialFlow2DMixed::Create(
        IndexType NewId, NodesArrayType const& ThisNodes, PropertiesType:
        :Pointer pProperties) const
    {
        return Element::Pointer(new IncompressiblePotentialFlow2DMixed(
        NewId, GetGeometry().Create(ThisNodes), pProperties));
    }

    // Destructor
    IncompressiblePotentialFlow2DMixed::
        ~IncompressiblePotentialFlow2DMixed()
    {}

    // Methods
    void IncompressiblePotentialFlow2DMixed::CalculateLocalSystem(
        MatrixType& rLeftHandSideMatrix, VectorType& rRightHandSideVector,
```

```
    ProcessInfo& rCurrentProcessInfo)
{
    KRATOS_TRY

    unsigned int number_of_nodes = GetGeometry().PointsNumber();
    // Number of nodes of the element. In this case we are using linear
    // triangular elements so we will have 3 nodes.
    unsigned int DOFs_per_node = 3; // Number of degrees of freedom
    // (DOFs) of each node, i.e. the number of unknowns:
    // TRANSPORTED_QUANTITY, TRANSPORTED_QUANTITY_GRADIENT_X and _Y.
    unsigned int system_vector_length = number_of_nodes * DOFs_per_node;
    // Length of the local system vectors. It is also the number of
    // rows and columns of the local LHS matrix.

    // Resize the LHS matrix and the RHS vector as needed
    if (rLeftHandSideMatrix.size1() != system_vector_length)
    {
        rLeftHandSideMatrix.resize(system_vector_length,
            system_vector_length, false);
    }
    rLeftHandSideMatrix = ZeroMatrix(system_vector_length,
    system_vector_length);

    if (rRightHandSideVector.size() != system_vector_length)
    {
        rRightHandSideVector.resize(system_vector_length, false);
    }
    rRightHandSideVector = ZeroVector(system_vector_length);

    array_1d <double, 3> N_Gauss_1; // Vector containing the shape
    // function of each node evaluated at the linear Gauss point.
    boost::numeric::ublas::bounded_matrix <double, 3, 2> DN_DX;
    // Matrix containing the gradient of the shape functions.
    // It has 3 rows (nodes) and 2 columns (x and y coordinates) -->
    // [dN1_dx dN1_dy; dN2_dx dN2_dy; dN3_dx dN3_dy].
    double Area; // Element area

    // Compute the necessary geometry data for the current element
    GeometryUtils::CalculateGeometryData(GetGeometry(), DN_DX, N_Gauss_1,
    Area); // N_Gauss_1 is not used for the calculation of either the
    // LHS or the RHS, but it is needed for computing DN_DX.

    // Define and calculate the stabilization parameters
    double h = sqrt(2.0 * Area); // Characteristic length of the element.
    double tau_phi = 1.0 / (4.0 / pow(h, 2.0)); // Stabilization
    // parameter resulting from the subscale of the phi variable.
    double tau_xi = 0.1; // Stabilization parameter resulting from
    // the subscale of the xi variable.

    // Calculate the integration points of the element
    const GeometryType::IntegrationPointsArrayType& integration_points
```

```
= GetGeometry().IntegrationPoints(GeometryData::GI_GAUSS_2);
// Quadratic integration order (the maximum needed in the system).
const Matrix& N_Gauss_2 = GetGeometry().ShapeFunctionsValues
(GeometryData::GI_GAUSS_2); // Matrix containing the shape
// function of each node evaluated at each of the quadratic
//Gauss points. It has 3 rows (integration points) and 3 columns.
//KRATOS_WATCH(N_Gauss_2)
double jacobian_det = 2.0 * Area; // Determinant of the jacobian
//from the transformation of cartesian coordinates into natural
// coordinates. For linear triangular elements it has a constant
// value over all the element.
/*if (this->Id() == 7353)
    {KRATOS_WATCH(this->Id())
    KRATOS_WATCH(jacobian_det)
    KRATOS_WATCH(DN_DX)
}*/

// Compute and assemble the element LHS matrix. It is divided in
// 9 different blocks, with a size of 3x3 each. The full element LHS
// matrix is 9x9

// For the calculation of such matrices and vectors we need to use
// numerical integration. We loop over all the integration points
for (unsigned int point_number = 0; point_number <
integration_points.size(); point_number++)
{
    array_1d <double, 3> N = row(N_Gauss_2, point_number);
        // Select the row of shape functions at the current point.
    double weight = integration_points[point_number].Weight();
        // Integration weight of the current integration point.

    for (unsigned int i = 0; i < number_of_nodes; i++)
    {
        for (unsigned int j = 0; j < number_of_nodes; j++)
        {
            rLeftHandSideMatrix(i, j) += weight * jacobian_det * tau_xi
        * (DN_DX(i, 0) * DN_DX(j, 0) + DN_DX(i, 1) * DN_DX(j, 1));
        // Block 11
            rLeftHandSideMatrix(i, number_of_nodes + j) += weight *
        jacobian_det * (1.0 - tau_xi) * DN_DX(i, 0) * N(j); // Block 12
            rLeftHandSideMatrix(i, 2 * number_of_nodes + j) += weight
        * jacobian_det * (1.0 - tau_xi) * DN_DX(i, 1) * N(j);
        // Block 13

            rLeftHandSideMatrix(number_of_nodes + i, j) += weight *
        jacobian_det * (1.0 - tau_xi) * N(i) * DN_DX(j, 0); // Block 21
            rLeftHandSideMatrix(number_of_nodes + i, number_of_nodes + j)
        += weight * jacobian_det * ((tau_xi - 1.0) * N(i) * N(j) -
        tau_phi * DN_DX(i, 0) * DN_DX(j, 0)); // Block 22
            rLeftHandSideMatrix(number_of_nodes + i, 2 *
        number_of_nodes + j) -= weight * jacobian_det * tau_phi
```

```
            * DN_DX(i, 0) * DN_DX(j, 1); // Block 23

                rLeftHandSideMatrix(2 * number_of_nodes + i, j) += weight
            * jacobian_det * (1.0 - tau_xi) * N(i) * DN_DX(j, 1);
            // Block 31
                rLeftHandSideMatrix(2 * number_of_nodes + i,
                number_of_nodes + j)
            -= weight * jacobian_det * tau_phi * DN_DX(i, 1) * DN_DX(j, 0);
            // Block 32
                rLeftHandSideMatrix(2 * number_of_nodes + i, 2 *
            number_of_nodes + j) += weight * jacobian_det * ((tau_xi -
            1.0) * N(i) * N(j) - tau_phi * DN_DX(i, 1) * DN_DX(j, 1));
            // Block 33
            }
        }
    }

    // Finally, substract the Dirichlet term from the RHS
    array_1d <double, 9> Dirichlet_Vector = ZeroVector(9); // Vector
    // containing the local Dirichlet conditions, i.e. the prescribed
    // values of the unknowns (if any).

    for (unsigned int i = 0; i < number_of_nodes; i++)
    {
        Dirichlet_Vector[i] = GetGeometry()[i].
            FastGetSolutionStepValue(VELOCITY_POTENTIAL);
        Dirichlet_Vector[i + number_of_nodes] = GetGeometry()[i].
            FastGetSolutionStepValue(VELOCITY_X);
        Dirichlet_Vector[i + 2 * number_of_nodes] = GetGeometry()[i].
            FastGetSolutionStepValue(VELOCITY_Y);
    }

    rRightHandSideVector -= prod(rLeftHandSideMatrix,
    Dirichlet_Vector);

    KRATOS_CATCH("")
}

void IncompressiblePotentialFlow2DMixed::EquationIdVector(
    EquationIdVectorType& rResult, ProcessInfo& rCurrentProcessInfo)
{
    unsigned int number_of_nodes = GetGeometry().PointsNumber();
    unsigned int DOFs_per_node = 3;
    unsigned int system_vector_length = number_of_nodes *
    DOFs_per_node;

    if (rResult.size() != system_vector_length)
    {
        rResult.resize(system_vector_length, false);
    }
```

```
        // In this case the element vector of unknowns will have the
        // following structure:
        // rResult = [u_1; u_2; u_3; v_1; v_2; v_3; phi_1; phi_2; phi_3];
        for (unsigned int i = 0; i < number_of_nodes; i++)
        {
            rResult[i] = GetGeometry()[i].GetDof(VELOCITY_POTENTIAL).
                EquationId();
            rResult[i + number_of_nodes] = GetGeometry()[i].GetDof(VELOCITY_X).
                EquationId();
            rResult[i + 2 * number_of_nodes] = GetGeometry()[i].
                GetDof(VELOCITY_Y).EquationId();
        }
    }

    void IncompressiblePotentialFlow2DMixed::GetDofList(DofsVectorType&
        rElementalDofList, ProcessInfo& rCurrentProcessInfo)
    {
        unsigned int number_of_nodes = GetGeometry().PointsNumber();
        unsigned int DOFs_per_node = 3;
        unsigned int system_vector_length = number_of_nodes * DOFs_per_node;

        if (rElementalDofList.size() != system_vector_length)
        {
            rElementalDofList.resize(system_vector_length);
        }

        for (unsigned int i = 0; i < number_of_nodes; i++)
        {
            rElementalDofList[i] = GetGeometry()[i].pGetDof(VELOCITY_POTENTIAL);
            rElementalDofList[i + number_of_nodes] = GetGeometry()[i].
                pGetDof(VELOCITY_X);
            rElementalDofList[i + 2 * number_of_nodes] = GetGeometry()[i].
                pGetDof(VELOCITY_Y);
        }
    }
} // namespace Kratos
```