UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Mecânica

JOÃO LUCAS DE SOUSA ALMEIDA

Implementation and Validation of a Discontinuous Galerkin Approach for Two-Dimensional Reynolds-Averaged Navier-Stokes (RANS) Turbulence Modelling

Implementação e Validação de uma Abordagem Galerkin Descontínuo para Modelagem de Turbulência das Equações de Navier-Stokes com Média de Reynolds (RANS) Bidimensionais

CAMPINAS
2019

JOÃO LUCAS DE SOUSA ALMEIDA

Implementation and Validation of a Discontinuous Galerkin Approach for Two-Dimensional Reynolds-Averaged Navier-Stokes (RANS) Turbulence Modelling

Implementação e Validação de uma Abordagem Galerkin Descontínuo para Modelagem de Turbulência das Equações de Navier-Stokes com Média de Reynolds (RANS) Bidimensionais

Dissertation presented to the School of Mechanical Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Mechanical Engineering, in the area of Solid Mechanics and Mechanical Project.

Dissertação apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Mecânica, na Área de Mecânica dos Sólidos e Projeto Mecânico.

Orientador: Prof. Dr. Marco Lúcio Bittencourt

Co-orientador Prof. Dr. Alberto Costa Nogueira Júnior

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO JOÃO LUCAS DE SOUSA ALMEIDA, E ORIENTADA PELO PROF. DR. MARCO LÚCIO BITTENCOURT.

CAMPINAS

2019

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

AL64i

Almeida, João Lucas de Sousa, 1993-
Implementation and validation of a discontinuous galerkin approach for two-dimensional reynolds-averaged navier-stokes (rans) turbulence modelling / João Lucas de Sousa Almeida. – Campinas, SP : [s.n.], 2019.

Orientador: Marco Lucio Bittencourt.
Coorientador: Alberto Costa Nogueira Junior.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Galerkin, Métodos de. 2. Jacobi, Polinômios de. 3. Navier-Stokes, Equações de. 4. Turbulência. I. Bittencourt, Marco Lucio, 1964-. II. Nogueira Junior, Alberto Costa. III. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Implementação e validação de uma abordagem galerkin descontínuo para modelagem de turbulância das equações de navier-stokes com média de reynolds (rans) bidimensionais
Palavras-chave em inglês:
Galerkin Methods
Jacobi Polynomials
Navier-Stokes Equations
Turbulence
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Marco Lucio Bittencourt
Philippe Remy Bernard Devloo
Maicon Ribeiro Correa
Data de defesa: 17-10-2019
Programa de Pós-Graduação: Engenharia Mecânica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-5872-064X
- Currículo Lattes do autor: http://lattes.cnpq.br/4877054354684937

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE SISTEMAS INTEGRADOS

DISSERTAÇÃO DE MESTRADO ACADÊMICO

Implementation and Validation of a Discontinuous Galerkin Approach for Two-Dimensional Reynolds-Averaged Navier-Stokes (RANS) Turbulence Modelling

Implementação e Validação de uma Abordagem Galerkin Descontínuo para Modelagem de Turbulência das Equações de Navier-Stokes com Média de Reynolds (RANS) Bidimensionais

Autor: João Lucas de Sousa Almeida
Orientador: Prof. Dr. Marco Lúcio Bittencourt
Co-orientador: Prof. Dr. Alberto Costa Nogueira Júnior

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Marco Lucio Bittencourt
Faculdade de Engenharia Mecânica/UNICAMP

Prof. Dr. Philippe Remy Bernard Devloo
Faculdade de Engenharia Civil, Arquitetura e Urbanismo/UNICAMP

Prof. Dr. Maicon Ribeiro Correa
Instituto de Matemática, Estatística e Computação Científica/UNICAMP

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa de Engenharia Mecânica da Faculdade de Engenharia Mecânica.

Campinas, 17 de outubro de 2019.

For my parents.

ACKNOWLEDGEMENTS

I should like to express my thanks to my advisor Marco Lucio Bittencourt and my co-advisor Alberto Costa Nogueira Jr. for the continuous support, guidance, monitoring and text proofreading, without which this work could not be properly performed. I cannot express my gratitude to Cláudio Alessandro de Carvalho Silva and Renato Fernandes Cantão, the main developers of the project Manticore, of which I am just a minor colaborator, their large expertise in numerical analysis and software architecture was essential for the construction of the base Manticore's engine. I also would like to thank the IBM Brazil Research Lab, my workplace, for the flexibility and support that gave me the necessary time to accomplish my work. In addition, I would like to specially thank Leonardo Silveira de Albuquerque Martins, my manager, who has encouraged me and given me the support to accomplish this work. Finally, I must to thank all my family, for its attention and patience during the last uncertain, difficult and mentally unstable times of reading, coding and writing for this work.

RESUMO

Uma abordagem de Elementos Finitos Galerkin Descontínuo (DG) para as equações de Navier-Stokes com Média de Reynolds (RANS) complementadas pelo modelo Spalart-Allmaras (SA) é implementada e validada para alguns casos de teste básicos. As variáveis de campo do problema são interpoladas usando expansões modais com polinômios de Jacobi. A comunicação entre os elementos é garantida pelo uso dos fluxos numéricos de Roe e HLLC para os termos convectivos e BR1 para os dissipativos. A integração temporal é realizada usando-se um esquema implícito Standard-Newton GMRES Backward Euler. O software desenvolvido neste trabalho tem intensivamente usado e expandido o pacote de aplicações para fluidodinâmica da plataforma de código aberto Manticore, de forma a permitir a construção da infraestrutura dos modelos RANS.

Keywords: RANS; DG-FEM, Spalart-Allmaras, Soluções Manufaturadas.

ABSTRACT

A Discontinuous Galerkin Finite Elements (DG) approach for the Reynolds-Averaged Navier-Stokes Equations (RANS) complemented by the closure model Spalart-Allmaras (SA) is implemented and validated for some basic test cases. The problem field variables are interpolated using modal expansions of Jacobi polynomials. The communication between the elements is enforced by using the numerical fluxes Roe and HLLC for the convective terms and BR1 for the dissipative ones. The time-integration is performed by using an implicit Standard-Newton GMRES Backward Euler scheme. The software developed in this work has extensively used and expanded the fluid dynamics toolbox of the open-source framework Manticore in order to construct the infra-structure of the RANS models.

Keywords: RANS; DG-FEM, Spalart-Allmaras, Manufactured Solutions.

Theories are nets cast to catch what we call the 'world': to rationalize, to explain, and to master it. We endeavour to make the mesh ever finer and finer.

Karl Popper

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

t	time.
x_i	coordinate of the Cartesian system.
$\mathbf{x}$	coordinates vector of the Cartesian system.
γ	fluid compressibility.
k	thermal conductivity.
Pr	Prandl Number.
Re	Reynolds number.
μ	dynamical viscosity.
ρ	density.
u, v	components of the velocity field in a two-dimensional Cartesian system.
$\boldsymbol{\tau}$	viscous stress tensor.
p	pressure.
U	internal energy per mass unit.
E	total energy per mass unit.
μ_e	eddy viscosity
$\tilde{\nu}$	auxiliary eddy viscosity
$\mathbf{v}$	field variables vector.
$\mathbf{U}$	velocity field vector.
$\mathbf{q}$	heat transfer vector.
$\boldsymbol{\mathcal{F}_c}$	convective flux vector of the Navier-Stokes equations.
$\boldsymbol{\mathcal{F}_v}$	viscous flux vector of the Navier-Stokes equations.
$\boldsymbol{\mathcal{F}_t}$	turbulent flux vector of the RANS equations.
$\boldsymbol{\mathcal{S}_t}$	turbulent source terms.
$\partial\boldsymbol{\Omega}$	closed boundary of a spatial domain $\boldsymbol{\Omega}$.

$\tilde{u}$	fluctuation of the variable u.
$\widehat{u}$	mean component of the variable u.
$\bigoplus$	operator direct sum.

CONTENTS

1 INTRODUCTION

A noticeable obstacle for the union of the Computational Fluid Dynamics (CFD) and the most traditional high order methods, such as the classical Finite Element Method (FEM), is their difficulty to make compatible the continuity requirements of such methods in the presence of shock propagation, a common phenomenon in the compressible flow cases. In order to circumvent this issue, an alternative way has been developed in the last decades, the Discontinuous Galerkin Finite Element Method (DG-FEM) (HESTHAVEN; WARBURTON, 2008).

The DG-FEM is a variation of the classical FEM, from where it inherits its principal characteristics, such as the use of domain tessellation into sub-domains, weak form relaxation and elementwise interpolation. However it differs of the original method for not directly imposing the continuity element-to-element. The continuity between two neighbours sub-domains is guaranteed by means of a numerical flux, a technique imported from the finite volume method (LEVEQUE, 2004). The numerical flux schemes available in the literature ensure the required robustness and flexibility for detecting both continuous and discontinuous solutions in the element interfaces, allowing to perform cases not covered for the classical FEM due to its natural limitations.

The DG-FEM had important results in a number of applications in the last decades, for linear and non-linear problems. Specifically for compressible flows, DG-FEM has attracted attention due to its basic features, most part of them derived from the high locality of the method, centering the operations in the element and its interfaced neighbourhood. Some of these characteristics enable parallelization capability and flexibility in the interpolation order, making simpler to have different regions of the simulation domain with the necessaries *hp*-adaptivity.

However, in counterpart of these benefits, the DG-FEM also inherits the challenge of dealing with the co-existence of discontinuities and non-linearities in case of higher interpolation orders, which may lead to the propagation of numerical oscillations. In order to ensure stability, it is almost indispensable the application of oscillation control schemes that fatefully imply in interference of the solution. However the high locality of these schemes reduce the global interference and make the effect rather negligible.

Turbulence presents another usual numerical challenge due to its multi-scale structure with a wide range of characteristic scale-lengths, consequently, it has a natural difficulty to be described considering the current capacity of the computers available. In this context, the RANS methods offer an achievable solution by replacing the complete scale description by time-averaged variable fluctuations, quantified by using additional

models (usually named closures). Surely, the scales condensing into mean values implies in an unavoidable physical description loss of local phenomena, but still allows to outline the behaviour of a complex flow and extract important aspects of it. No wonder RANS still is one of the standard approaches in current industrial applications.

Although RANS makes use of assumptions for simplifying the turbulence simulations, the technique has some numerical difficulties, such as the time-integration stability and the emergence of non-physical values in the auxiliary variables. As the time-discretization is a crucial for a accurate evaluation of the fluctuations, the time steps can be something restrictive, mainly considering the conditions at the simulation start up. In addiction, many non-realistic scenarios can arise, leading to non-physical responses of the turbulence models. For controlling the time-instability is imperative the use of well-conditioned implicit time-integration and the non-physical behaviour can be handled by using additional limiting mechanisms.

The software here used has been developed based on Manticore, a free and open-source DG-FEM framework implemented in Python 3 (https://bitbucket.org/cantao/manticore/). The present work has contributed to the project Manticore on three ways: testing the inviscid and laminar Navier-Stokes modules using the benchmarks of the literature, implementing the RANS modules (and the auxiliary infrastructure necessary for that) and implementing the current post-processing tools available on the Manticore's repository.

The scope of the flow simulations is subdivided in three categories in this work: inviscid (also named Euler) validation, laminar Navier-Stokes validation and RANS models validation, all of them in subsonic conditions. Therefore, no shock discontinuity is observed in the tests array. At first glance such approach may seem a contradiction considering the DG-FEM numerical characteristics. However, it is necessary to highlight that the method presents others important features (as previously cited), and the field discontinuities handling is only one of those. Besides, even though this work does not intend to cover the transonic and supersonic cases, it aims to be a basis step, validating and implementing essential modules for enabling a continuous developing of the platform Manticore, allowing others topics be approached in future works.

1.1 Purpose

This work intends to implement and validate a numerical software for Reynolds-Averaged Navier-Stokes (RANS) simulation applied to basic validation cases. For a sake of a complete consistency checking, inviscid and laminar cases are also validated. This work uses the high-order numerical methods, instead of the most traditional methods (based on low order methods, as the Finite Volume Methods) The capacity of the higher order

methods to describe the complexity of the phenomenon can tightly outweigh their higher computational costs.

1.2 Outline

The work is organized as follows. The Chapter 2 is a bibliographic review of the fundamental theory in CFD and DG methods. In Chapter 3 it is introduced and explained the basics physical theory about the phenomena considered in this work. Chapter 4 deals with the numerical formulation employed in the software implementation. In Chapter 6, the validation tests are performed and compared to the benchmarks available in the literature.

2 BIBLIOGRAPHIC REVIEW

2.1 Fundamentals in Fluid Dynamics

A comprehensive explanation about any field of Fluid Dynamics requires the establishment (or revision) of some basic concepts:

- Fluid and Flow characteristics:
  - Continuum Hypothesis
  - Compressibility
  - Transport
  - Viscosity
  - Boundary Layer
  - Flow Regime
- Fundamentals of the Mathematical Modeling :
  - Control Volume
  - Characteristic Variables
  - Conservation Laws
  - Differential Form

2.1.1 Continuum Hypothesis

The fundamental consideration in Fluid Mechanics is to suppose the fluid as a continuous medium. This assumption is valid because in mesoscale, in absence of chemical reactions, the molecular effects on fluid are negligible.

2.1.2 Transport

In the context of the Fluid Dynamics, pure transport is the transfer of mass and energy by means the displacement of the flowing matter along non-intersecting trajectories, named streamlines. The characteristic transport variable are the velocity field and the terms regarding are denominated inertial terms. When the presence of viscosity, the purely transport behaviour can be disturbed by the interaction between the two forces and the consideration about the trajectories is violated.

2.1.3 Viscosity

Viscosity is the measure of internal stickiness of a fluid (ÇENGEL; CIMBALA, 2010). Stickiness is, roughly speaking, the resistance of a fluid do motion. It can be correlated with the shear stress in the flow sections. In a special class of fluids, denominated Newtonian (such as air and water), the shear stress can be modelled as linearly proportional to the rate of deformation of the flow section.

2.1.4 Boundary Layer

The Boundary Layer is the contact region between the fluid and a surface where the viscous effects are significant in comparison with the inertial ones. Thus, there are a transition between the pure transport region, where the inertial terms are prevailing, and the a static condition, where the stickiness show up completely.

2.1.5 Flow Regime

Roughly speaking, the flow regime denotes the fluid motion behaviour in terms of its smoothness level. The regime classification covers three subdivisions, laminar, transition and turbulent The regime is primarily distinguished by visual aspects, that are correlated to quantified indicators. In the laminar case, the fluid particles are disposed in an organized and highly predictable motion whereby is possible to recognize a streamline flow. The turbulent situation occurs when the interaction between the inertial and the viscous forces in the boundary layer becomes unstable and a swirling structures arises along the flow. In this state, the disordered motion becomes predominant. The transition is just a intermediary situation between the laminar and turbulent state where characteristics of both the extreme conditions are observed, but the turbulence still are not completely evolved. The most traditional dimensionless parameter for quantifying the regime is the Reynolds number (Re), that are given as following:

$$\mathrm{Re} = \frac{\rho\, U\, l}{\mu} \tag{2.1}$$

Where ρ is the density of the fluid, U is a characteristic velocity, l is a characteristic length (usually related to the geometry interacting with the flow) and μ is a the dynamic viscosity. The Reynolds number represents the ratio between inertial ($\rho\, U\, l$) and viscous time scales (μ) and gives a notion about the fluid dynamical stability in the flow. As the Reynolds number rises, the role of the viscous effects becomes decreasingly significant and, at the limit case, negligible.

2.1.6 Control Volume

A control volume is a delimited region in the space where a flow are studied. A control volume is different of a closed system, because its boundary can exchange matter and energy. The boundary of a control volume is usually denominated surface control.

2.1.7 Characteristic Variables

The analysis of a control volume is performed by means a set of conservative or extensive variables. The extensive variables are dependent on the size of the control volume, such as, mass , momentum and energy. Locally, it is possible to determine the primitive or intensive variables. The intensive variables are independent of the control volume dimensions, for instance, density, velocity, pressure and temperature.

2.1.8 Conservation Laws

The expressions derived from the global balances of the conservative variables are denominated conservation laws. In a generic way, a conservation law can be enunciated as following: The global balance applied to a extensive variable $\mathcal{B}$ in a static control volume Ω surrounded by a surface control $\partial\Omega$ yields the conservation law of $\mathcal{B}$:

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\left(\int_{\Omega}\rho\bar{\mathcal{B}}\,\mathrm{d}\,\mathcal{V}\right)+\int_{\partial\Omega}\rho\bar{\mathcal{B}}\mathbf{U}\cdot\mathbf{n}\mathrm{d}\,\mathcal{S} \tag{2.2}$$

Where $\bar{\mathcal{B}}$ is the variable $\mathcal{B}$ per mass unit (that corresponds to its intensive variable). The general form of the global balance to conservative variables is named Reynolds Transportation Theorem.

2.1.9 Differential Form

The equations of the global conservation laws written so far are denominated integral form. From the integral form is possible to derive the differential form by means of the Divergence Theorem from Calculus. In the differential form, the conservation laws are rewritten in terms of the intensive variables.

2.2 Turbulence Numerical Modelling

When talking about turbulence numerical issues it is necessary to introduce some basic definitions used in such matter with regard to the approaches employed to model turbulence itself. We can subdivide the turbulence numerical simulation techniques in three major categories:

- DNS (Direct Numerical Simulation) - The Navier-Stokes equations are solved in their original form and all the turbulent length scales are considered in the numerical computation.
- LES (Large Eddy Simulation) - Extra schemes are used to separate the larger scales from the smaller ones. The large length scales are computed and a mathematical model is used to the smaller ones.
- RANS (Reynolds Averaged Navier-Stokes) - Approach based in the time-averaging of the Navier-Stokes equations. The individual scales are condensed into a mean value and computed at a time.

2.3 Turbulence Theory

Reynolds (KÁRMÁN, 1938) gave the fundamental step for constructing the statistical approach of turbulence by introducing the concept of statistical mean values on fluid dynamics. Reynolds also introduced the decomposition of the instantaneous field variables into a time-averaged and a fluctuation components (MCDONOUGH J., 2007). That basic concept, afterwards known as Reynolds decomposition, is the principle of the Reynolds-averaged Navier-Stokes equations (RANS).

Taylor (TAYLOR, 1935) formalized the notion of turbulence eddy scale by defining the scale length concept as an analogous for the turbulence modelling of the free mean path from the kinetic gas theory. Taylor also introduced the concept of isotropic turbulence, as a condition where the mean squares and mean products of the velocity field components are invariant with respect to the coordinate system rotation and reflection. (TAYLOR, 1935), (KÁRMÁN, 1938).

Turbulence is qualitatively characterised as a multi-scale phenomenon (MCDONOUGH J., 2007). The Kolmogorov's work conceives the energy cascade as a model for describing the energy transfer throughout the energy spectrum (MCDONOUGH J., 2007), according to this hypothetical system, the largest scale, named integral scale, takes energy from the free flow and continuously transfers it from the larger to the smaller scales and so on, until the molecular or Kolmogorov's scale, in which the energy is dissipated as the form of heat and vibration.

Kolmogorov models the energy transfer in the intermediary eddies as inviscid (MCDONOUGH J., 2007), with energy dissipation only in the molecular scale. However, in order to not violate the energy cascade chain, it is necessary to consider the complete eddy scale range to accurately describe the turbulence phenomenon. It corresponds to directly solve the Navier-Stokes equations in its non-modified form for high Reynolds

numbers, such approach is usually denominated Direct Numerical Simulation (DNS) on the CFD community.

According to the Kolmogorov's laws, which correlates the flow regime and the eddy scale parameters, the characteristic length of the smallest scale rapidly decays with the increase of the Reynolds number (MCDONOUGH J., 2007) making the DNS domain discretization almost impractical for capturing all the phenomenon features. On the scope of the numerical simulation, this feature can represent an insurmountable barrier considering the current computational resources.

Due to the critical limitation of the DNS method, alternative ways have been developed for achieving a good trade-off between accuracy and computational cost. The alternative analytical and numerical approaches, such as RANS, basically describes all scales together, ignoring their individual aspects. In this scenario, where the structural approaches present limitations concerning to the numerical analysis, the statistical theory becomes relevant as a way to comprehend the main features of the turbulence phenomenon.

The RANS system is obtained by decomposing the flow field into mean and fluctuating components following the Reynolds decomposition and, applying an integral time-averaging operation over the Navier-Stokes equations (LANDMANN, 2008). The final RANS equations describe the field variables in terms of their time-averaged values in addition to a fluctuation component, whose momentum term is commonly referred as Reynolds stress tensor. The RANS technique basically replaces the treatment of the individual scales for an integral time-average of the entire spectrum. Thus, turbulence scales are considered as a condensed entity.

Nonetheless, the RANS model itself does not supply a complete equation system, since there are not an expression for direclty relating the fluctuations and the field variables. This, it requires the use of an additional model, commonly named closure, for the RANS equations (MCDONOUGH J., 2007).

The most difficulty aspect of closuring the RANS equations lies in dealing with the non-linearity of the Reynolds stress tensor, where there are products between fluctuation values. In order to circumvent this feature, Boussinesq proposed to model the Reynolds stress tensor as a linear correlation between the viscous gradient tensor and a parameter denominated turbulent or eddy viscosity (LANDMANN, 2008).

Based on the Boussinesq's approximation, algebraic and differential closure models have been proposed in order to evaluate the eddy viscosity and its derived variables. In this work we follow the developments from the differential models in which the eddy viscosity treatment is satisfied by creating new partial differential equations to be included on the RANS equations system.

Although the Boussinesq's hypothesis had gained a large acceptance in developing the closure models, the validity and generality of such assumption has been contested in more recent works. Schimitt (SCHMITT F, 2007) compared the correspondence between the Reynolds stress tensor and the symmetric part of the viscous gradient tensor using representative cases of DNS, LES and experimental datasets and demonstrated weak correlations between the Boussinesq's hypothesis and the above-mentioned baseline dataset for the most part of the simulation domains.

Jones and Launder proposed a two-equations system for low Reynolds numbers, so-called $k-\epsilon$ models, in which the eddy viscosity is modelled as a function of two auxiliary variables, the turbulent kinetic energy (k) and the energy dissipation rate (ϵ) (JONES; LAUNDER, 1972).

The original $k-\epsilon$ was applied in a large variety of flow situations and improved in subsequent works. Meanwhile, the technique presented a number of shortcomings from the aerodynamic and numeric viewpoints, such as the lack of sensitivity to adverse pressure gradients and the intrinsic numerical stiffness when treating the viscous sub-layer (MENTER, 1994).

Wilcox proposed a two-equations model, referred as $k-\omega$. Wilcox assumed the eddy viscosity as an algebraic function of two extra variables and introduced a new partial differential equation for each one of them together with some calibration constants and functions (WILCOX, 1988).

Menter modified the original $k-\omega$ eddy viscosity evaluation function by embodying a limiter in order to reduce the turbulent shear-stress overprediciton and avoid the instantaneous response of the flow shear-stress to the shear-strain rate observed in the original model. The improved model was called Shear Stress Transport (SST) (MENTER, 1994).

Spalart and Allmaras proposed an one-equation method for indirectly modelling the eddy viscosity by means of an auxiliary viscosity variable. The Spalart and Allmaras's model (SA) makes use a set of parameters in order to calibrate the method for aerodynamical purposes (SPALART; ALLMARAS, 1992)

Spalart and Allmaras subsequently modified the original SA model in order to enhance stability and robustness in dealing with adverse aerodynamical cases. One of these modifications was to introduce a limiting scheme for preventing negative values of the modified viscosity ($\tilde{S}$) in the near-wall region (ALLMARAS *et al.*, 2012a).

2.4 Advances in DG-FEM

DG-FEM was introduced by Reed and Hill (REED; HILL, 1973) to approximate the linear neutron transport equation using regular triangular meshes. In this approach the flux terms are directly interpolated over the element interfaces using one-dimensional Lagrange polynomials with no direct continuity imposition to the field variables. This work started a new research thread for the development of new DG-FEM techniques reaching a broad set of applications.

Cockburn et al. (COCKBURN; SHU, 1991) (COCKBURN *et al.*, 1989) produced the most prominent advances for adapting the DG techniques to non-linear problems by constructing a framework to solve non-linear time-dependent systems of equations using fundamental concepts from high order FEM and FV such as high order polynomial interpolation within elements and exact or approximated Riemann's solvers to evaluate the numerical fluxes at the element boundaries. The approach presented in these works, called RKDG, employs explicit Total Variation Diminishing (TVD) Runge-Kutta in order to perform the time discretization (LANDMANN, 2008).

Bassi and Rebay (BASSI; REBAY, 1997) (BASSI; REBAY, 2000) lead off a new branch of techniques to treat elliptic operators, based on a mixed formulation (OLIVER, 2008), in which the second-order system of equations is converted to a first-order one and discretized via DG method. Bassi and Rebay constructed two versions of their mixed formulation, known as BR1 and BR2. BR1 scheme performs contour integrals along all the element interfaces whereas BR2 performs the same integration using just the shared edge of two neighbour elements. Such feature of BR2 implies in a shorter communicating stencil.

Cockburn and Shu (COCKBURN; SHU, 1998) proposed the local discontinuous Galerkin (LDG) schemes as an extension and generalization of the original RKDG method to convection-diffusion systems. The LDG methodology presented in this work was devised to approach non-linear, time dependent convection-diffusion systems maintaining the high parallelism capability, high order accuracy and easy handling of complex geometries. LDG basically transforms the second-order system to a first-order one by introducing auxiliary equations to approximate gradients. After that, the DG spatial discretization is performed by replacing the non-linear flux terms on the element boundaries by numerical fluxes, that mimics the upwinding scheme which is done always in opposite directions for the main and auxiliary variables (HESTHAVEN; WARBURTON, 2008, pp. 252).

Peraire et al. (PERAIRE; PERSSON, 2007) developed an alternative approach for solving elliptic problems by proposing the compact discontinuous Galerkin (CDG) technique, very similar to the Cockburn and Shu's LDG approach. In this case

the second-order equation is converted to a first-order system, but the new additional auxiliary variables are substituted back in the original equation in a primal form scheme. When dealing with multiple dimensions the CDG method is able to ensure more compactness by eliminating connections between distant elements.

Warburton and Hesthaven (HESTHAVEN; WARBURTON, 2008) synthesized the essence of the DG-FEM theory in their book in order to provide a complete conceptual introduction and implementation guidelines, covering from fundamental ideas to complex modelling problems. Despite of focusing on nodal interpolation methods, the book presents a considerable degree of generality, and can be used as a guide for different approaches.

2.5 DG-FEM for Navier-Stokes and RANS equations

Birken (BIRKEN *et al.*, 2012), studied the time-integration problem in 2D simulation of Navier-Stokes equations using modal DG-FEM discretization. In the Birken's work the authors measured the stiffness of the Navier-Stokes system of equations (for a flow with Reynolds number $Re = 100$) by evaluating the eigenvalues regarding the linearized form of the right-hand side operator for a 4th-order interpolation . The large spectrum of the real parts indicates the difficulty in ensuring the stability during the time-integration process. In this case, the explicit integration showed a severe limitation concerning to the time step choice. The implicit approaches have proved to be the most suitable way for performing the Navier-Stokes and RANS steady-state cases, because they can be contructed to have unbounded stability regions (BIRKEN *et al.*, 2012).

Bassi et al. (BASSI *et al.*, 2004) applied modal DG-FEM and a second-order implicit Runge-Kutta time-integrator approach for solving a 2D incompressible RANS system using the closure model $k - \omega$. The formulation is tested simulating steady-state flow over a zero pressure gradient flat plate with $M = 0.2$ and $Re = 11.1 \times 10^6$, using a polynomial interpolation order up to $p = 2$ using a grid with 110×80 elements, 96 elements in the horizontal direction lying on the plate. The tests are validated by comparing the solution to the reference law of wall available on the literature, highlighting the influence of the near-wall grid resolution and the polynomial order in capturing the viscous sub-layer turbulent behaviour.

Nguyen et al. (NGUYEN *et al.*, 2007) used a modal DG-FEM approach for a modified version of the 2D incompressible RANS-SA equations in which it is introduced an artificial viscosity stabilization scheme aimed at enabling high order approximations even for coarse grids. The validation tests are performed for zero pressure gradient flat plate, NACA0012 airfoil and flow over a cylinder. The flat plate tests use different grid resolutions, the coarsest one being a triangularly split 10×16 mesh with p=1 and the finest one a triangularly split 145×241 mesh with $p = 4$. The values of the skin friction

coefficient obtained from the numerical results are compared with experimental data and the velocity profiles are compared with both experimental and law of wall reference values. The results of coarsest grid presents a good agreement with the experimental and theoretical benchmarks when using higher order ($p = 4$, 37×61 equivalent resolution).

Landmann (LANDMANN, 2008) implemented a modal DG-FEM approach in order to study the Navier-Stokes equations for the inviscid, laminar and RANS-turbulent cases. Employing the RANS models SA and $k - \omega$ the study achieved the interpolation order $p = 3$ on the turbulent flat plate and airfoil-A cases. For the SA flat plate tests it was used a triangularly split 44×13 H-grid with 24 elements along the plate (considered coarse (LANDMANN, 2008, pp. 108))and for the $k - \omega$ tests a 88×38 grid with 48 elements along the plate, the test conditions are $M = 0.3$ and $Re = 3 \times 10^6$. The flat plate results revealed a good agreement with the theoretical skin friction profiles (Blasius and turbulent theory (LANDMANN, 2008, pp. 108)). The A-airfoil tests are performed using the SA model and employ a C-grid with 64×16 quadrilateral elements. The p-refinement showed a good agreement with the expected experimental results for the pressure and skin coefficients (LANDMANN, 2008, pp. 115). Landmann observed that the closure models can violate the positivity of the turbulent variables on part of the domain, leading to non-physical and potentially unstable solutions. In order to avoid this, the author proposed the usage of post-processing limiting techniques (LANDMANN, 2008, pp. 44) .

Oliver (OLIVER, 2008) purposed DG-FEM approach allied to a mesh adaptive algorithm to solve RANS problems closed by the SA model and analyses the dual consistence of the discretization. The work verifies that the mixed formulations are generally asymptotically dual consistent, nevertheless, the fashion of weighting gradient-dependent source terms by trial functions and integrating is revealed as dual inconsistent. The validation of the adaptive scheme is performed by using the zero pressure gradient with $M = 0.25$ and $Re = 1.0 \times 10^7$, the initial mesh has 234 elements and the first layer of elements has a too large spacing for accurate boundary layer evaluations. According to the work, by using this mesh the numerical implementation was not able of obtaining $p \geq 1$ convergence for steady-state problems, notwithstanding the unsteady adaptation algorithm was successful even using a considerable coarse tessellation. Initializing the problem with homogeneous fields, using $p = 3$ and dual consistent discretization, the unsteady adaptations algorithm eventually converges to a steady-state solution that meets the established error criteria of 0.2 drag counts.

3 PHYSICAL MODELLING

Let us to consider Newtonian fluid in motion within a two-dimensional control volume, in which there can be interchanging of mass and energy in the control surfaces. The fluid is single-phase and does not undergo chemical reactions. The scale at which the studied phenomena occurs and the order of magnitude of the the numerical discretization are sufficiently large to assume the continuum hypothesis. Based on the aforementioned assumptions and making use of the fundamental conservation laws, it is possible to determine the governing equations of a general compressible viscous flow for Newtonian fluids, the Navier-Stokes equations, as can be seen next.

3.1 The Navier-Stokes Equations

The homogeneous Navier-Stokes Equations (NSE) in their vector form are given3 by:

$$\begin{aligned} &\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{c}}(\mathbf{v}) = \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}(\mathbf{v}, \nabla \mathbf{v}), \\ &\mathbf{v} = \mathbf{v}(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^2. \end{aligned} \tag{3.1}$$

Each vector term of the previous equations system is given by:

$$\mathbf{v} = \begin{bmatrix} \rho \\ \rho\,\mathbf{U} \\ \rho\,E \end{bmatrix}, \tag{3.2}$$

$$\boldsymbol{\mathcal{F}}_{\boldsymbol{c}} = \begin{bmatrix} (\rho\mathbf{U})^T \\ (\rho\mathbf{U})\mathbf{U}^T + p\mathbf{I} \\ (\rho E + p)\mathbf{U}^T \end{bmatrix}, \tag{3.3}$$

$$\boldsymbol{\mathcal{F}}_{\boldsymbol{v}} = \begin{bmatrix} \mathbf{0}^T \\ \boldsymbol{\tau}^T \\ \mathbf{U}^T\boldsymbol{\tau} + \mathbf{q} \end{bmatrix}, \tag{3.4}$$

$$\boldsymbol{\tau}_{ij} = \mu\left(\frac{\partial u_i}{\partial\, x_j} + \frac{\partial u_j}{\partial\, x_i} - \frac{2}{3}\frac{\partial u_k}{\partial\, x_k}\delta_{ij}\right), \tag{3.5}$$

$$E = \frac{1}{2}\mathbf{U}^T\mathbf{U} + U, \tag{3.6}$$

$$\mathbf{q} = -\lambda \frac{\partial E}{\partial \mathbf{x}}, \tag{3.7}$$

$$\mathbf{x} \in \mathbb{R}^2. \tag{3.8}$$

Where ρ is the fluid density, $\mathbf{U}$ is the velocity field, E is the energy per mass unit, U is the internal energy per mass unit, p is the pressure, $\mathbf{q}$ is the heat flux on the domain and $\boldsymbol{\tau}$ is the viscous stress tensor for the Newtonian fluids. The thermal conductivity (k) is given by

$$k = \frac{\gamma\,\mu}{Pr}, \tag{3.9}$$

Where μ is the dynamical viscosity, γ is the gas compressibility and Pr is the Prandtl number.

The NSE system given above is a general form that can describe any flow in which the fluid can be modelled as an ideal gas or a mixture of such gases and its behavior can be properly modelled using the tensor seen in 3.5. By disregarding the fluid viscosity, the equations above take a purely hyperbolic form, known as inviscid transport equations or Euler equations,

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}}_c(\mathbf{v}) = \mathbf{0}, \tag{3.10}$$

The Euler equations, despite being a non-realistic case, present interesting features from the numerical viewpoint, given that the absence of viscous dissipation can enable the emergence of discontinuities during the problem evolution, characteristic which has a noticeable importance for testing the numerical solvers capability in dealing with severe gradients. Notwithstanding this work does not cover the cases presenting sharp gradients, the inviscid transport equations still provide a fundamental testing case for evaluating the robustness of the numerical implementation in coping with the solution of non-linear problems at high order of field interpolation and geometry.

The NSE are based on the continuum mechanics hypothesis, therefore it does not make sense to consider the flow as composed by individual freely-moving molecules, but as a deformable continuous domain, and the characteristic variables as continuous maps for any point inside such region. For sufficiently low velocities, a two-dimensional viscous flow is similar to a sequence of narrow fluid strips slipping one each other in regular and smooth profiles. Due to such visual aspects, this condition is named laminar (see 2.1.5 for further explanations). In flows over rigid walls, the transition region between the moving fluid and the stationary flow in contact with the wall is denominated laminar boundary layer (see the section 2.1.4). Nonetheless, as the available kinetic energy on the flow is increased, the laminar stability is disrupted and a new state is configured, the turbulent condition.

3.2 The Physics of the Turbulence: A Basic Introduction

The turbulence phenomenon is characterized by the emergence of specific structures in the viscous flow, denominated eddies. An eddy is basically a rotational structure, similar to a swirl. The eddies are usually classified according to the order of magnitude of their action radius. However, even within the action region of an eddy, there can exist other smaller scale eddies, given that these formations can present a wide spectrum of characteristic length, ranging from visible vortex formations to whirls in the molecular scale, compounding a complex flow. Due to that, the turbulence is commonly denominated a multi-scale phenomenon.

Such swirling structures arise from the struggle between the transport and the fluid viscosity on the boundary layer. When there is high kinetic energy availability, this strong interaction is unstabilized, triggering a process where kinetic energy is taken from the mean flow and sequentially transferred in the eddy scales. The most traditional model for understanding this process is the Kolmogorov's theory for turbulence (MCDONOUGH J., 2007), whose basic assumption is known as energy cascade. According to the cascade hypothesis, for high Reynolds numbers the dominant direction of the energy transfer occurs from the largest to the smallest scales (also named molecular scales), the large eddies are responsible for injecting energy in the system and the smallest for dissipating it (JOSSERAND *et al.*, 2017), generating heat and vibration. Kolmogorov hypothesized the intermediary scales transfer energy without dissipation and denominated them inertial scales.

Following this theoretical framework, the eddies scales can be subdivided into four fundamental categories (MCDONOUGH J., 2007). The large scale, whose size has the same order of magnitude as the flow domain, here represented as L. The integral scale, whose size is a first order fraction of the large scale. The Taylor microscale, which corresponds to intermediary inertial scales of the Kolmogorov's assumption. The Kolmogorov scale, the smallest turbulence scales, in which the dissipation effectively occurs according to the Kolmogorov's theory.

Making use of the dimensional analysis, it is possible determine an estimative for the turbulence lengths scale ratio as (MCDONOUGH J., 2007)

$$\frac{\eta}{\ell} \sim Re^{3/4}, \tag{3.11}$$

Where η represents the length of the Kolmogorov scale and ℓ of the integral length scale.

The Kolmogorov's theory reveals another important feature of the turbulence phenomenon, there is not a hierarchy among the scales with regard to their relevancy for

the model, since all of them have a role in the phenomenon description, therefore, it is not not physically consistent simply to disregard some of them in an approximated approach. As the size difference between the largest and the smallest scales can be very significant, in a traditional numerical scheme, for embracing all of the scales, the discretization of a flow domain must be sufficiently refined in order to capture the smallest eddies, whose size can be microscopic.

From the numerical standpoint, let us to consider approaching the turbulence modelling via Direct Numerical Simulation (DNS) 2.2 in which the Navier-Stokes equations are discretized in their natural form. Considering a three-dimensional case and assuming that the integral scale is one order behind the domain scale, the volume of the mesh cell necessary for capturing the smallest scale is comparable to $L^3 Re^{-\frac{9}{4}}$. In addition, given that the smallest scales also have the more elevated frequencies, a refined time discretization is likewise necessary.

As the Reynolds number increases, to comprise all the turbulence spectrum becomes technically impractical for a grid refinement taking into account the computational infrastructure currently available. That way, the DNS approach is suitable only for certain cases in which the Reynolds number is sufficiently low for allowing practical time-spatial discretizations.

On the other hand, the Reynolds-Averaged Navier-Stokes (RANS) methods 2.2, although perform a less detailed description of the phenomenon, are able to ensure a good predictability for the general physical behavior and require less computational effort if compared to the DNS approach. Currently, RANS is the standard choice for the most part of the industrial simulation purposes and still has a great space in the academic research.

3.3 The Reynolds-Averaged Navier-Stokes (RANS)

As aforementioned, the Navier-Stokes equations in their natural form are not suitable for the numerical purposes due to the wide difference between the smallest and the largest eddies scales, that will imply large grid size refinement and small simulation time-steps (LANDMANN, 2008). In order to circumvent these limitations, it was purposed the RANS approach, where the Navier-Stokes equations undergo a time-averaging process in order to condense the information of the different time scales into a mean behavior. Such approach allows to better deal with the wide spectrum of the turbulence scales enabling more realistic mesh and time steps.

The field variables vector can be decomposed into a turbulent ($\sim$) and a lami-

nar ($\wedge$) components according to the Favre decomposition as (LANDMANN, 2008)

$$\mathbf{v} = \widehat{\mathbf{v}} + \widetilde{\mathbf{v}}, \tag{3.12}$$

$$\widehat{\mathbf{v}} = \frac{\overline{\rho\,\mathbf{v}}}{\overline{\rho}}. \tag{3.13}$$

Where the the notation $\overline{\ }$ denotes the Reynolds decomposition.

The Navier-Stokes equations can be written according to the Favre's decomposition as

$$\frac{\partial}{\partial\, t}(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}}) + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}}) = \nabla \cdot \boldsymbol{\mathcal{F}_v}(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}}, \nabla(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}})). \tag{3.14}$$

Time-averaging both the sides.

$$\frac{1}{\mathcal{T}}\int_t^{t+\mathcal{T}} \frac{\partial}{\partial\, t}(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}}) + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}})\,\mathrm{d}\,t = \frac{1}{\mathcal{T}}\int_t^{t+\mathcal{T}} \nabla \cdot \boldsymbol{\mathcal{F}_v}[\widehat{\mathbf{v}} + \widetilde{\mathbf{v}}, \nabla(\widehat{\mathbf{v}} + \widetilde{\mathbf{v}})]\,\mathrm{d}\,t.$$

After simplifications we can write (LANDMANN, 2008, p. 15)

$$\frac{\partial \widehat{\mathbf{v}}}{\partial\, t} + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\widehat{\mathbf{v}}) = \nabla \cdot \boldsymbol{\mathcal{F}_v}(\widehat{\mathbf{v}}, \nabla\widehat{\mathbf{v}}) + \nabla \cdot \boldsymbol{\mathcal{F}_t}(\widetilde{\mathbf{v}}). \tag{3.15}$$

Where

$$\boldsymbol{\mathcal{F}_t} = \begin{bmatrix} 0 \\ \langle \rho \widetilde{\mathbf{U}}\, \widetilde{\mathbf{U}}^T \rangle \\ k\,\langle \rho\, T\, \widetilde{\mathbf{U}} \rangle \end{bmatrix}. \tag{3.16}$$

The term $\langle \rho \widetilde{\mathbf{U}}\, \widetilde{\mathbf{U}}^T \rangle$ is commonly referred as the Reynolds stress tensor and $k\,\langle \rho\, T\, \widetilde{\mathbf{U}} \rangle$ as the turbulent heat transfer.

3.3.1 The Boussinesq's Hypothesis

The fluctuations of the variables are unknown at principle. The way to determine them is to assume the Boussinesq's hypothesis (LANDMANN, 2008, p. 15), which states that the Reynolds stress tensor is linearly related to the mean viscous tensor as

$$\langle \rho \widetilde{\mathbf{U}}\, \widetilde{\mathbf{U}}^T \rangle = \left(\frac{\mu_e}{\mu}\right) \boldsymbol{\tau}\,(\widetilde{\mathbf{U}}, \nabla\, \widetilde{\mathbf{U}}) - \frac{2}{3}\,\rho\, K, \tag{3.17}$$

In which K is denominated turbulent kinetic energy

$$K = \frac{1}{2}\,\langle \widetilde{\mathbf{U}}\, \widetilde{\mathbf{U}}^T \rangle \cdot \mathbf{I}. \tag{3.18}$$

Thus, the viscous turbulent flux can be rewritten as

$$\boldsymbol{\mathcal{F}_t} = \begin{bmatrix} \mathbf{0} \\ \left(\frac{\mu_e}{\mu}\right) \boldsymbol{\tau}\left(\widehat{\mathbf{U}}, \nabla \widehat{\mathbf{U}}\right) + \frac{2}{3}\rho K \\ \mathbf{U}^T \left(\frac{\mu_e}{\mu}\right) \boldsymbol{\tau} \; - k_e \nabla \widehat{U} \end{bmatrix}. \tag{3.19}$$

μ_e is the eddy viscosity and k_e is the turbulent thermal conductivity coefficient, that can be related to the eddy viscosity according to the expression

$$k_e = \frac{\gamma \mu_e}{Pr_e}, \tag{3.20}$$

Where γ is the fluid compressibility, Pr_e is the turbulent Prandtl number, that can be assumed as constant (LANDMANN, 2008).

Based on the RANS general model is possible to derive a set of turbulence models for modelling the undetermined coefficients from the Boussinesq's hypothesis. The RANS complementary models are usually referred as closure models.

The undetermined parameters, μ_e and K are estimated by introducing auxiliary equations in the RANS system. The general RANS structure take the form

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\mathbf{v}) = \nabla \cdot \boldsymbol{\mathcal{F}_v}(\mathbf{v}, \nabla \mathbf{v}) + \nabla \cdot \boldsymbol{\mathcal{F}_t}(\mathbf{v}, \nabla \mathbf{v}) + \boldsymbol{\mathcal{S}_t}(\mathbf{v}, \nabla \mathbf{v}). \tag{3.21}$$

The superscripts $\widehat{\ }$ of the mean values will be omitted in the upcoming sections for a sake of simplicity, thus, all the variables with the notation $\mathbf{v}$ are already considered in their mean components. Each vector term seen in the equation 3.21 is now described. $\mathbf{v}$ is the field variables vector and is given by

$$\mathbf{v} = \begin{bmatrix} \rho \\ \rho \mathbf{U} \\ \rho E \\ \cdots \\ \mathbf{v}_t \end{bmatrix}, \tag{3.22}$$

$\boldsymbol{\mathcal{F}_c}$ is the convective fluxes vector and it is as follows

$$\boldsymbol{\mathcal{F}_c} = \begin{bmatrix} \rho \, \mathbf{U} \\ \rho \, \mathbf{U} \, \mathbf{U^T} + p \, \mathbf{I} \\ (\rho E + p)\mathbf{U} \\ \cdots \\ \boldsymbol{\mathcal{F}_{ct}} \end{bmatrix}, \tag{3.23}$$

$\boldsymbol{\mathcal{F}_v}$ is the viscous flux vector and can be seen below

$$\boldsymbol{\mathcal{F}_v} = \begin{bmatrix} \mathbf{0}^T \\ \boldsymbol{\tau}^T \\ \mathbf{U}^T\boldsymbol{\tau} + \mathbf{q} \\ \cdots \\ \mathbf{0} \end{bmatrix}, \tag{3.24}$$

Finally, the turbulent fluxes vector

$$\boldsymbol{\mathcal{F}_t} = \begin{bmatrix} \mathbf{0} \\ \left(\frac{\mu_e}{\mu}\right)\boldsymbol{\tau} + \frac{2}{3}\rho K \\ \mathbf{U}^T\left(\frac{\mu_e}{\mu}\right)\boldsymbol{\tau} - k_e\nabla\widehat{U} \\ \cdots \\ \boldsymbol{\mathcal{F}}_{tt} \end{bmatrix} \tag{3.25}$$

and the turbulent source vector

$$\boldsymbol{\mathcal{S}_t} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ \boldsymbol{\mathcal{S}}_{tt} \end{bmatrix}. \tag{3.26}$$

In the expressions above $\mathbf{v}_t$, $\boldsymbol{\mathcal{F}}_{ct}$, $\boldsymbol{\mathcal{F}}_{tt}$ and $\boldsymbol{\mathcal{S}}_{tt}$ represents the turbulence auxiliary variables, their convective fluxes, turbulent fluxes and source terms respectively.

3.4 The Spalart-Allmaras Model

The Spalart-Allmaras (SA) approach focuses in modelling the eddy viscosity μ_e by introducing an extra field variable to be determined. Therefore,

$$\mu_e = \rho\,\tilde{\nu}\,f_{\nu\,1}, \tag{3.27}$$

where $f_{\nu\,1}$ is a parameter determined by the method. The variable $\tilde{\nu}$ will be referenced here as the eddy kinematic viscosity and modelled as follows (LANDMANN, 2008):

$$\begin{aligned} \frac{\partial}{\partial t}(\rho\,\tilde{\nu}) + \nabla\cdot(\rho\,\tilde{\nu}\mathbf{U}) = & \\ & \frac{1}{\sigma}\left\{\nabla\cdot[(\mu+\rho\,\tilde{\nu})\nabla\,\tilde{\nu}] + \rho\,c_{b2}\,\nabla\,\tilde{\nu}\;\cdot\;\nabla\,\tilde{\nu}\right\} \\ & + c_{b\,1}(1-f_{t\,2})\rho\,\tilde{S}\,\tilde{\nu} \\ & - \left(c_{w\,1}\,f_w - \frac{c_{b\,1}}{\kappa^2}\,f_{t\,2}\right)\frac{1}{\rho}\left(\frac{\rho\,\tilde{\nu}}{D}\right)^2 \\ & + \rho\,f_{t\,1}\,\Delta\,U^2. \end{aligned} \tag{3.28}$$

The SA model is fundamentally an empirical approach, constructed with relations based on Galilean invariance, dimensional analysis and calibration using experimental and direct numerical simulation data (SPALART; ALLMARAS, 1992). The turbulent kinetic energy modelling is not covered for the classical SA approach. The effect of the previous limitation is compensated with the inclusion of a set of parameters experimentally calibrated. Each parameter of the extra equation is described below.

For the sake of compactness, some terms in this equation will be rewritten. Each source term can be described by its role in the behavior of the eddy viscosity and the turbulence phenomenon. Therefore

$$\mathcal{P} = c_{b\,1}(1 - f_{t\,2})\rho\,\tilde{S}\,\tilde{\nu}, \tag{3.29}$$

$$\mathcal{D} = \left(c_{w\,1}\,f_w - \frac{c_{b\,1}}{\kappa^2}\,f_{t\,2}\right)\frac{1}{\rho}\,\left(\frac{\rho\,\tilde{\nu}}{D}\right)^2, \tag{3.30}$$

$$\mathcal{T} = \rho\,f_{t\,1}\,\Delta\,U^2. \tag{3.31}$$

Substituting 3.29, 3.30 and 3.31 in 3.28

$$\begin{aligned}\frac{\partial}{\partial\,t}(\rho\,\tilde{\nu}) + \nabla\cdot(\rho\,\tilde{\nu}\mathbf{U}) = \\ \frac{1}{\sigma}\,\{\nabla\cdot[(\mu+\rho\,\tilde{\nu})\nabla\,\tilde{\nu}] + \rho\,c_{b2}\,\nabla\,\tilde{\nu}\,\cdot\,\nabla\,\tilde{\nu}\} \\ +\mathcal{P} - \mathcal{D} + \mathcal{T},\end{aligned} \tag{3.32}$$

where $\mathcal{P}$ is the production term, $\mathcal{D}$ is the destruction term and $\mathcal{T}$ is the trip term. The parameters $c_{w\,1}$, $c_{b\,1}$, $c_{b\,2}$ and κ and the functions $f_{t\,2}$, $f_{t\,2}$, f_w and $f_{\nu\,1}$ are calculated or set by means of the SA methodology correlations (ALLMARAS *et al.*, 2012b). D is a geometric measure and represents the distance to the nearest wall.

The term $\tilde{\mathcal{S}}$ seen in the equation 3.28 is the given by:

$$\tilde{S} = ||\Omega|| + \frac{\tilde{\nu}}{\kappa\,D^2}\,f_{\nu\,2}, \tag{3.33}$$

where Ω is the vorticity

$$\Omega = \nabla\,\times\,\mathbf{U}. \tag{3.34}$$

The SA model also employs some auxiliary functions:

$$f_w = g\left(\frac{1+{c_{w\,3}}^6}{g+{c_{w\,3}}^6}\right)^{\frac{1}{6}}, \tag{3.35}$$

$$g = r + c_{w\,2}\,(r^6 - r), \tag{3.36}$$

$$r = min\,\left(\frac{\tilde{\nu}}{\tilde{S}\,\kappa^2\,D^2}, 10\right), \tag{3.37}$$

$$f_{\nu\,1} = \frac{\chi^3}{\chi^3 + {c_{\nu\,1}}^3}, \tag{3.38}$$

$$f_{\nu\,2} = 1 - \frac{\chi}{1 + \chi\,c_{\nu\,1}}, \tag{3.39}$$

$$\chi = \frac{\tilde{\nu}\,\rho}{\mu}, \tag{3.40}$$

$$f_{t\,1} = c_{t\,1}\,g_t\,exp\left[-c_{t\,1}\,\frac{{\Omega_t}^2}{\Delta\,U^2}(D^2 + {g_t}^2\,{D_t}^2)\right], \tag{3.41}$$

$$g_t = min\left(0.1, \frac{\Delta\,U^2}{|\Omega_t|\,\Delta\,x_t}\right), \tag{3.42}$$

$$f_{t\,2} = c_{t\,3}\,exp\left(c_{t\,4}\,\chi^2\right). \tag{3.43}$$

where Ω_t regards to the vorticity of the nearest trip point and D_t the distance to this point. Besides the auxiliary functions, also there is a set of fixed parameters:

$$c_{b\,1} = 0.1355, c_{b\,2} = 0.622, \tag{3.44}$$

$$\sigma = \frac{2}{3}, \kappa = 0.41, \tag{3.45}$$

$$c_{w\,1} = \frac{c_{b\,1}}{\kappa^2} + \frac{1 + c_{b\,2}}{\sigma}, c_{w\,2} = 0.3\,c_{w\,3} = 2.0, \tag{3.46}$$

$$c_{v\,1} = 7.1, c_{t\,1} = 1.0, c_{t\,2} = 2.0, c_{t\,3} = 1.2, c_{t\,4} = 0.5. \tag{3.47}$$

The functions $f_{t\,2}$, $f_{t\,2}$, f_w, $f_{\nu\,1}$ and $f_{\nu\,2}$ have the role of adjusting the model during the transition to the turbulent regime.

The functions f_{t1} and f_{t2} can be set to zero as performed in (NGUYEN *et al.*, 2007) and (OLIVER, 2008), in considering the flow as fully turbulent. Thus, equations 3.29 and 3.30 can be rewritten as follows:

$$\mathcal{P} = c_{b1}\rho\,\tilde{S}\,\tilde{\nu}, \tag{3.48}$$

$$\mathcal{D} = c_{w1}\,f_w\frac{1}{\rho}\left(\frac{\rho\,\tilde{\nu}}{D}\right)^2, \tag{3.49}$$

The trip term vanishes and g_t is no longer necessary.

The PDE system formed by the RANS equations in addition to the eddy viscosity correlation is given by

$$\frac{\partial \mathbf{v}}{\partial\, t} + \nabla\cdot\boldsymbol{\mathcal{F}_c}(\mathbf{v}) = \nabla\cdot\boldsymbol{\mathcal{F}_v}(\mathbf{v},\nabla\mathbf{v}) + \nabla\cdot\boldsymbol{\mathcal{F}_t}(\mathbf{v},\nabla\mathbf{v}) + \boldsymbol{\mathcal{S}_t}(\mathbf{v},\nabla\mathbf{v}), \tag{3.50}$$

$$\mathbf{v} = \begin{bmatrix} \rho \\ \rho\mathbf{U} \\ \rho E \\ \rho\tilde{\nu} \end{bmatrix}, \tag{3.51}$$

$$\boldsymbol{\mathcal{F}_c} = \begin{bmatrix} \rho\,\mathbf{U} \\ \rho\,\mathbf{U}\,\mathbf{U^T} + p\,\mathbf{I} \\ (\rho E + p)\mathbf{U} \\ \rho\,\tilde{\nu}\,\mathbf{U} \end{bmatrix}, \tag{3.52}$$

$$\boldsymbol{\mathcal{F}_v} = \begin{bmatrix} \mathbf{0}^T \\ \boldsymbol{\tau}^T \\ \mathbf{U}^T\boldsymbol{\tau} + \mathbf{q} \\ 0 \end{bmatrix}, \tag{3.53}$$

$$\boldsymbol{\mathcal{F}_t} = \begin{bmatrix} \mathbf{0} \\ \left(\frac{\mu_e}{\mu}\right)\boldsymbol{\tau} \\ \mathbf{U}^T\left(\frac{\mu_e}{\mu}\right)\boldsymbol{\tau} - k_e\,\nabla U \\ \frac{1}{\sigma}(\mu + \rho\,\tilde{\nu})\nabla\,\tilde{\nu} \end{bmatrix}, \tag{3.54}$$

$$\boldsymbol{\mathcal{S}_t} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\rho\,c_{b2}}{\sigma}\,\nabla\,\tilde{\nu}\,\cdot\,\nabla\,\tilde{\nu} + \mathcal{P} - \mathcal{D} \end{bmatrix}. \tag{3.55}$$

3.5 Dimensionless RANS Equations

The different orders of magnitude of the RANS field variables can difficult the attainment of a proper convergence. In order to avoid such issue, we introduce the dimensionless form of the RANS equations, which is, a rescaling process that searches to approximate the orders of magnitude. In order to accomplish it, let us to consider the method for rescaling the independent and primitive variables such as

$$\begin{aligned} \bar{t} = \frac{U_{ref}\, t}{L_{ref}},\ \bar{x} = \frac{x}{L_{ref}},\ \bar{y} = \frac{y}{L_{ref}},\ \overline{D} = \frac{D}{L_{ref}},\ \bar{\mu} = \frac{\bar{\mu}}{\mu_{ref}}, \\ \bar{\rho} = \frac{\rho}{\rho_{ref}}\ \bar{u} = \frac{u}{U_{ref}},\ \bar{v} = \frac{v}{U_{ref}},\ \bar{\tilde{\nu}} = \frac{\tilde{\nu}}{\tilde{\nu}_{ref}}\,, \bar{p} = \frac{p}{\rho_{ref}{U_{ref}}^2}, \end{aligned} \tag{3.56}$$

and the definition of the Reynolds number in

$$Re = \frac{\rho_{ref} U_{ref} L_{ref}}{\mu_{ref}}, \tag{3.57}$$

which are the basis for rewriting the RANS PDE system.

The reference values (with subscripts *ref*) are chosen accordingly the specific problem. Making use of the assumptions 3.56 and 3.57 we can rewrite each term of the RANS equations, as

$$\bar{\mathbf{v}} = \begin{bmatrix} \bar{\rho} \\ \bar{\rho}\overline{\mathbf{U}} \\ \bar{\rho}\,\overline{E} \\ \bar{\rho}\bar{\tilde{\nu}} \end{bmatrix}, \tag{3.58}$$

$$\boldsymbol{\mathcal{F}_c} = \begin{bmatrix} \bar{\rho}\,\overline{\mathbf{U}} \\ \bar{\rho}\,\overline{\mathbf{U}}\,\overline{\mathbf{U}}^{\mathbf{T}} + \bar{p}\,\mathbf{I} \\ (\bar{\rho}\overline{E} + \bar{p})\overline{\mathbf{U}} \\ \bar{\rho}\,\bar{\tilde{\nu}}\,\overline{\mathbf{U}} \end{bmatrix}, \tag{3.59}$$

$$\boldsymbol{\mathcal{F}_v} = \frac{1}{Re} \begin{bmatrix} \mathbf{0}^T \\ \overline{\boldsymbol{\tau}}^T \\ \overline{\mathbf{U}}^T\overline{\boldsymbol{\tau}} + \overline{\mathbf{q}} \\ 0 \end{bmatrix}, \tag{3.60}$$

$$\boldsymbol{\mathcal{F}_t} = \frac{1}{Re} \begin{bmatrix} \mathbf{0} \\ \left(\frac{\mu_e}{\mu}\right)\overline{\boldsymbol{\tau}}^T \\ \mathbf{U}^T\left(\frac{\mu_e}{\mu}\right)\boldsymbol{\tau} - k_e\,\nabla\overline{U} \\ \frac{1}{\sigma}(\overline{\mu} + \overline{\rho}\,\tilde{\nu})\nabla\,\tilde{\nu}, \end{bmatrix} \tag{3.61}$$

$$\boldsymbol{\mathcal{S}_t} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\overline{\rho}\, c_{b2}}{\sigma} \nabla \overline{\tilde{\nu}} \cdot \nabla \overline{\tilde{\nu}} + \overline{\mathcal{P}} - \overline{\mathcal{D}} \end{bmatrix}. \tag{3.62}$$

The production and destruction terms of the eddy viscosity PDE are impacted by the rescaling, in the following way (LANDMANN, 2008).

$$\overline{\mathcal{P}} = c_{b\,1} \overline{\rho}\, \overline{\tilde{S}}\, \overline{\tilde{\nu}}, \tag{3.63}$$

$$\overline{\mathcal{D}} = \frac{1}{Re}\, c_{w\,1}\, f_w \frac{1}{\overline{\rho}} \left(\frac{\overline{\rho}\, \overline{\tilde{\nu}}}{\overline{D}} \right)^2. \tag{3.64}$$

The dimensionless versions of the modified vorticity and the adjusting parameter r are given, respectively, by (LANDMANN, 2008).

$$\overline{\tilde{S}} = ||\overline{\Omega}|| + \frac{1}{Re} \frac{\overline{\tilde{\nu}}}{\kappa\, \overline{D}^2}\, f_{\nu\,2}, \tag{3.65}$$

$$\overline{r} = min \left(\frac{\overline{\tilde{\nu}}}{Re \overline{\tilde{S}}\, \kappa^2\, \overline{D}^2}, 10 \right). \tag{3.66}$$

For the sake of simplicity, we will omit the superscripts when writing the variables and consider them in their dimensionless form.

3.6 The general structure of the RANS equations

The structure of the general Reynolds-Averaged Navier-Stokes equations are given by:

$$\overbrace{\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\mathbf{v})}^{\text{I}} = \overbrace{\nabla \cdot \boldsymbol{\mathcal{F}_v}(\mathbf{v}, \nabla \mathbf{v})}^{\text{II}} + \overbrace{\nabla \cdot \boldsymbol{\mathcal{F}_t}(\mathbf{v}, \nabla \mathbf{v}) + \boldsymbol{\mathcal{S}_t}(\mathbf{v}, \nabla \mathbf{v})}^{\text{III}} \tag{3.67}$$

I - Euler terms.

II - Laminar Navier-Stokes term.

III - Turbulence terms.

4 NUMERICAL ANALYSIS

4.1 An Overview

In the classical Finite Element Method (FEM) the original boundary value problem is converted to a integral formulation, termed weak form, in which the problem variables are approximated by expansions over a function space, usually polynomial. In addition, the definition domain of the boundary value problem is discretized onto subdomains and the integral formulation is imposed in each of them individually. In principle, there is no limitation regarding the rank of the basis function space, and the FEM is commonly referred as a high-order method. In order to ensure the consistency of the solution, the FEM enforces the continuity of the BVP solution among elements directly equaling the solution in the neighbors elemental boundaries. This technique, though seemingly simple, presents difficulties from the computational standpoint, mainly concerning to the computational parallelism. In addition, the continuity enforcement implies limitations for the physical modelling, given its natural pitfall in dealing with discontinuities, such as the shock phenomenon.

By contrast, the Finite Volume Method (FVM) is typically a low order approach, in which zero order interpolations are performed in each computational cell and the solution is communicated each time step by using the numerical fluxes, a set of operations executed in the element boundaries in order to ensure the consistency of the domain discretization. Some variants of the classical FVM enable higher order approximation error, such as the ENO and WENO approaches. However, these schemes are based on reconstruction techniques rather than elementwise interpolation. The naturally non-coupled structure of the classical FVM is interesting from the parallelism viewpoint and flexible enough for capturing the continuous and the discontinuous profiles on the elemental interfaces.

The Discontinuous Galerkin Finite Elements Method (DG-FEM) is a FEM variation in which the elements are naturally uncoupled concerning to the solution approximation. The communication between two neighbors is performed by a numerical flux rather than the classical FEM direct continuity imposition, being able to detect both the discontinuous and the continuous solutions. In the numerical analysis scenario, DG-FEM arises as a hybrid approach, binding the high order interpolation and domain description typical of the classical FEM approaches, with the flexibility in performing elementwise operations and information interchanging of the FVM.

Meanwhile, the increasing use of the DG-FEM for studying a wide range of problems has revealed its limitations with respect to the compatibility between the high order and discontinuities, leading to the developing of schemes for controlling numerical

oscillations, that have acquired important results in the recent years. Despite of such troubles, the DG capability in describing the features of complex phenomena allied to its computational flexibility justify choosing it as numerical tool for this work.

4.2 Boundary Value Problem

Let $\mathbf{\Omega}$ be a continuous domain and $\partial\mathbf{\Omega}$ its closed boundary. Now, consider the boundary value problem defined over $\mathbf{\Omega}$

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{c}}(\mathbf{v}) + \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}(\mathbf{v}, \nabla\mathbf{v}) + \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}(\mathbf{v}, \nabla\mathbf{v}) + \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}(\mathbf{v}, \nabla\mathbf{v}), \tag{4.1}$$

$$\mathbf{v} = \mathbf{v}(\mathbf{x}, t),\ \mathbf{x} \in \mathbf{\Omega},\ t \in [0, \infty), \tag{4.2}$$

$$\mathbf{B}(\mathbf{v}) = \mathbf{g},\ \mathbf{x} \in \partial\mathbf{\Omega}, \tag{4.3}$$

where $\mathbf{B}$ represents a set of operations for imposing known conditions on the boundary. Note that the system of equations here considered is the RANS PDE system, presented in 3.6. We consider that it is sufficiently general for conveying the main characteristics of the DG discretization process.

In order to simplify the notation, let the right-hand side of 4.1 be represented as a residual operator such as following:

$$\frac{\partial \mathbf{v}}{\partial t} = \mathcal{R}(\mathbf{v}, \nabla\mathbf{v}). \tag{4.4}$$

4.3 Domain Discretization

Consider a consistent tessellation of $\mathbf{\Omega}$ into a set of N_e non-overlapping sub-domains. Thus, let $\mathcal{T}(\mathbf{\Omega})$ be a partition of the domain $\mathbf{\Omega}$, such as:

$$\mathcal{T} : \bigcup_{k=1}^{N_e} \mathbf{\Omega}^k = \mathbf{\Omega}, \quad \bigcap_{k=1}^{N_e} \mathbf{\Omega}^k = \emptyset, \tag{4.5}$$

$$\mathbf{\Omega}_h = \mathcal{T}(\mathbf{\Omega}). \tag{4.6}$$

The sub-domains $\mathbf{\Omega}^k$ are named elements. For practical purposes, we limit the possible geometry of the elements by considering only simple meshes, triangular or quadrilateral on $\mathbb{R}^2$. In this way, the boundary $\partial\mathbf{\Omega}^k$ of each element is composed by a determined number N_f of edges:

$$\partial\mathbf{\Omega}^k = \bigcup_{f=1}^{N_f^k} \partial\mathbf{\Omega}_f^k \tag{4.7}$$

Despite of referring to two or three-dimensional space, we usually refer to the boundaries components $\partial\mathbf{\Omega}_f^k$ as faces and the elemental domains $\mathbf{\Omega}^k$ as volumes.

Beyond the spatial discretization, let us consider the approximation solution as direct a sum of elementwise solutions (HESTHAVEN; WARBURTON, 2008, p. 36):

$$\mathbf{v}_h(\mathbf{\Omega}) = \bigoplus_{k=1}^{N_e} \mathbf{v}_h^k(\mathbf{\Omega}^k) \tag{4.8}$$

The solutions $\mathbf{v}_h^k$ can be continuous or discontinuous at the element interfaces according to the problem requirements.

4.4 The DG-FEM weak form

Consider the PDE system given in 4.4 for each element $\mathbf{\Omega}^k \subset \mathbf{\Omega}$ from the partition 4.5 and an approximated numerical solution $\mathbf{v}_h^k$ for the vector variable $\mathbf{v}$ inside the element. The approximation gives rise to an error term $\mathcal{E}$:

$$\frac{\partial \mathbf{v}_h^k}{\partial\, t} = \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) + \mathcal{E}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k), \tag{4.9}$$

$$\mathbf{v}_h^k = \mathbf{v}_h^k(\mathbf{x}, t),\ \mathbf{x} \in \mathbf{\Omega}^k \subset \mathbf{\Omega},\ t \in [0, \infty), \tag{4.10}$$

$$\mathbf{B}(\mathbf{v}_h^k) = \mathbf{g},\ \mathbf{x} \in \partial\mathbf{\Omega}^k \,|\, \partial\mathbf{\Omega}^k \subset \partial\mathbf{\Omega}. \tag{4.11}$$

The presence of residual is an unavoidable aspect of any numerical solution, however, making use of an suitable formulation it is possible to obtain an approximation conditioned to enable the minimal error among the possible solutions.

In order to construct such formulation, we need make use of some fundamental hypothesis. We take the assumption that the approximated solution $\mathbf{v}_h^k$ inside $\mathbf{\Omega}^k$ can be represented as an expansion over a function space Φ^k of dimension N_k

$$\mathbf{v}_h^k = \sum_{i=1}^{N} \widehat{\mathbf{v}}_i^k\, \phi_i^k \,|\, \widehat{\mathbf{v}}_i^k \in \mathbb{R}, i = 1, 2, 3, ..., N^k \tag{4.12}$$

$$\Phi : \phi^k \in \Phi^k, \phi^k = \phi^k(\mathbf{x}), \mathbf{x} \in \mathbf{\Omega}^k. \tag{4.13}$$

Notice that we assume the possibility of using expansion spaces with different dimensions for each element k, or different maximum ranks in case of polynomial expansions. As second assumption, let Φ^k be a real function space endowed with an inner product and a p-norm. The space of the real functions over a domain $\mathbf{\Omega}$ have its inner product defined as following:

$$\langle u,\, v\rangle_{\mathbf{\Omega}} = \int_{\mathbf{\Omega}} u\, v\, \mathrm{d}\mathbf{x} \,|\, (u, v) \in \Phi. \tag{4.14}$$

A p-norm for a real function defined over a continuous domain $\mathbf{\Omega}$ is defined as

$$||u||_p = \left(\int_{\mathbf{\Omega}} u^p\, \mathrm{d}\mathbf{x}\right)^{\frac{1}{p}} \in]0, \infty] \ |\, u \in \Phi, p \in \mathbb{N}. \tag{4.15}$$

The norm determines if a space is measurable and establishes a metric to indicate it. The set of functions which meet the condition 4.15 is represented as L^p. Considering the aforementioned inner product definition, we just need a second order normed space for ensuring the integrability of 4.14. Hence, $\Phi \subset L^2$.

By definition, the set L^2 is the set of all real function second order normed spaces, or second order Lebesgue measurable, over a continuous domain $\mathbf{\Omega}$, endowed with a inner product as stated in 4.14. The elements of Φ are usually referred as expansion basis, and we will name Φ basis space.

Let Ψ^k be a non-zero real function space defined over $\mathbf{\Omega}^k$ such as

$$\Psi^k : \psi^k \in \Psi^k, \psi^k = \psi(\mathbf{x}), \psi \neq 0, \mathbf{x} \in \mathbf{\Omega}^k \tag{4.16}$$

Considering $\Psi^k \subset L^2$, it has the same functional properties of Φ^k. Ψ^k will be here named trial space for the element $\mathbf{\Omega}^k$.

Now, we will rewrite the BVP expression by multiplying the PDE system by $\psi^k \in \Psi^k$ and integrating over the domain $\mathbf{\Omega}^k$

$$\int_{\mathbf{\Omega}^k} \psi^k \frac{\partial \mathbf{v}_h^k}{\partial t} \, \mathrm{d}\mathbf{x} = \int_{\mathbf{\Omega}^k} \psi^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x} + \int_{\mathbf{\Omega}^k} \psi^k \mathcal{E}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x} \tag{4.17}$$

This expression is denominated variational formulation (KARNIADAKIS; SHERWIN, 2004) of the BVP 4.1.

Through the orthogonality enforcement between the error and the trial space Ψ^k, the integral error term 4.17 will vanish. Therefore,

$$\int_{\mathbf{\Omega}^k} \psi^k \frac{\partial \mathbf{v}_h^k}{\partial t} \, \mathrm{d}\mathbf{x} = \int_{\mathbf{\Omega}^k} \psi^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x}. \tag{4.18}$$

In order to enable the orthogonality condition we must guarantee that the chosen trial and basis spaces have the same dimension $dim(\Phi^k) = dim(\Psi^k) = N^k$ when expanding the variables. Then

$$\int_{\mathbf{\Omega}^k} \psi_j^k \sum_{i=1}^{N_k} \frac{\partial \widehat{\mathbf{v}}_i^k}{\partial t} \, \phi_i^k \, \mathrm{d}\mathbf{x} = \int_{\mathbf{\Omega}^k} \psi_j^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x} \quad 0 \leq j < N^k, \tag{4.19}$$

and

$$\sum_{i=1}^{N_k} \int_{\mathbf{\Omega}^k} \frac{\partial \widehat{\mathbf{v}}_i^k}{\partial t} \psi_j^k \, \phi_i^k \, \mathrm{d}\mathbf{x} = \int_{\mathbf{\Omega}^k} \psi_j^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x} \quad 0 \leq j < N^k. \tag{4.20}$$

Equation 4.18 is named weak form of the variational formulation (KARNIADAKIS; SHERWIN, 2004). We will not make assumptions about $\mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k)$ at the moment because it concentrates all the non-linearity of the problem and requires a more detailed explanation, as it will be shown later. As we can see in 4.20, for ensuring the

integrability of each term in the weak form expression is necessary that the basis and trial spaces can be measured with a L^2 norm.

For properly constructing our FEM-based formulation, we need make choices for the trial (Ψ^k) and the basis (Φ^k) spaces. Here we adopt the Galerkin approach, which defines such spaces as equals. Therefore

$$\Psi^k = \Phi^k. \tag{4.21}$$

Replacing the trial space in accordance with the Galerkin approach, we have

$$\sum_{i=1}^{N_k} \int_{\mathbf{\Omega}^k} \frac{\partial \widehat{\mathbf{v}}_i^k}{\partial t} \phi_j^k \, \phi_i^k \, \mathrm{d}\mathbf{x} = \int_{\mathbf{\Omega}^k} \phi_j^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x} \quad 0 \leq j < N_k. \tag{4.22}$$

The left side can be rewritten in the form of a matrix operation. Let us define the elemental mass matrix $\boldsymbol{\mathcal{M}}^k$ of the Galerkin formulation as:

$$\boldsymbol{\mathcal{M}}^k : \boldsymbol{\mathcal{M}}_{ij}^k = \int_{\mathbf{\Omega}^k} \phi_j^k \, \phi_i^k \, \mathrm{d}\mathbf{x}, \quad 0 \leq i, j < N_k. \tag{4.23}$$

Notice that when Φ^k is an orthogonal basis, $\boldsymbol{\mathcal{M}}^k$ will be a diagonal matrix. The weak form can be written such as following:

$$\boldsymbol{\mathcal{M}}^k \frac{\partial \widehat{\mathbf{v}}^k}{\partial t} = \int_{\mathbf{\Omega}^k} \phi_j^k \mathcal{R}(\mathbf{v}_h^k, \nabla \mathbf{v}_h^k) \, \mathrm{d}\mathbf{x}, \quad 0 \leq j < N_k. \tag{4.24}$$

In which $\widehat{\mathbf{v}}^k$ represents the vector of expansion coefficients for the field variables v_h^k at each element $\mathbf{\Omega}^k$.

4.5 DG Weak Form of the original BVP

We obtain the DG scheme applied to the RANS PDE system by replacing the operators notation with the original terms seen in the BVP definition 4.1. The elemental weak form for the RANS equations is given by

$$\begin{aligned} \boldsymbol{\mathcal{M}}^k \frac{\partial \widehat{\mathbf{v}}^k}{\partial t} = &- \int_{\mathbf{\Omega}^k} {\phi_j}^k \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k \, \mathrm{d}\mathbf{x} + \int_{\mathbf{\Omega}^k} {\phi_j}^k \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k \, \mathrm{d}\mathbf{x} \\ &+ \int_{\mathbf{\Omega}^k} {\phi_j}^k \nabla \cdot \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k \, \mathrm{d}\mathbf{x} + \int_{\mathbf{\Omega}^k} {\phi_j}^k \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \, \mathrm{d}\mathbf{x}, \quad 0 \leq j < N_k. \end{aligned} \tag{4.25}$$

The divergent terms can be rewritten making use of the Product Rule and Divergence Theorem. Considering a closed domain $\mathcal{V}$ delimited by a boundary $\partial\mathcal{V}$, a vector field $\mathbf{F}$ and some function $\phi \in \Phi \subset L^2$, the integral

$$\int_{\mathcal{V}} \phi \nabla \cdot \mathbf{F} \mathrm{d}\mathbf{x} \tag{4.26}$$

Can be expanded using the Product Rule as

$$\int_{\mathcal{V}} \phi \nabla \cdot \mathbf{F} \mathrm{d}\mathbf{x} = \int_{\mathcal{V}} \nabla \cdot \phi \mathbf{F} \mathrm{d}\mathbf{x} - \int_{\mathcal{V}} \mathbf{F} \cdot \nabla \phi \mathrm{d}\mathbf{x}. \tag{4.27}$$

The first integral term of the right-hand side can be rewritten using the Divergence Theorem as

$$\int_{\mathcal{V}} \nabla \cdot \phi \mathbf{F} \mathrm{d}\mathbf{x} = \oint_{\partial \mathcal{V}} \mathbf{n} \cdot \phi \mathbf{F} \mathrm{d}\mathbf{x}, \tag{4.28}$$

Finally, we obtain the equivalent expression

$$\int_{\mathcal{V}} \phi \nabla \cdot \mathbf{F} \mathrm{d}\mathbf{x} = \oint_{\partial \mathcal{V}} \mathbf{n} \cdot \phi \mathbf{F} \mathrm{d}\mathbf{x} - \int_{\mathcal{V}} \mathbf{F} \nabla \phi \mathrm{d}\mathbf{x}. \tag{4.29}$$

Using the relation 4.29 we can rewrite the expression 4.25, giving rise to flux contour integral terms, as seen in

$$\begin{aligned} \boldsymbol{\mathcal{M}}^k \frac{\partial \widehat{\mathbf{v}}^k}{\partial t} = & \oint_{\partial \boldsymbol{\Omega}^k} \mathbf{n} \cdot {\phi_j}^k \boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k \mathrm{d}\mathbf{x} + \int_{\boldsymbol{\Omega}^k} \boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ & + \oint_{\partial \boldsymbol{\Omega}^k} \mathbf{n} \cdot {\phi_j}^k \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k \mathrm{d}\mathbf{x} - \int_{\boldsymbol{\Omega}^k} \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ & + \oint_{\partial \boldsymbol{\Omega}^k} \mathbf{n} \cdot {\phi_j}^k \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k \mathrm{d}\mathbf{x} - \int_{\boldsymbol{\Omega}^k} \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ & + \int_{\boldsymbol{\Omega}^k} {\phi_j}^k \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \, \mathrm{d}\mathbf{x} \quad 0 \leq j < N_k. \end{aligned} \tag{4.30}$$

Reorganizing the terms in 4.30 and condensing them according to similarity criteria, we obtain

$$\begin{aligned} \boldsymbol{\mathcal{M}}^k \frac{\partial \widehat{\mathbf{v}}^k}{\partial t} = & \int_{\boldsymbol{\Omega}^k} \left(\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k \right) \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ & + \oint_{\partial \boldsymbol{\Omega}^k} \mathbf{n} \cdot {\phi_j}^k \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k \right) \mathrm{d}\mathbf{x} \\ & + \int_{\boldsymbol{\Omega}^k} {\phi_j}^k \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \, \mathrm{d}\mathbf{x} \quad 0 \leq j < N_k \end{aligned} \tag{4.31}$$

4.6 Faces Communication

The elemental weak form 4.31 imposes no *a priori* restriction at the interfaces of an element with its vicinity. In the classical FEM methods the solution consistency over the computational domain is enforced by directly asserting the variables continuity element to element. In the case of DG-FEM formulations, an ambiguous definition of the solution is allowed at the elemental boundaries. The consistency is achieved by means of information interchanging between adjacent elements, which is accomplished via operations known as numerical fluxes, a scheme originated from the finite volumes method. The numerical flux defined over the face f of element k with regard to the neighbor face f' of element l is defined such as

$$\mathcal{F}_f^k = \mathcal{F}(\mathbf{v}(\partial \boldsymbol{\Omega}_f^k), \mathbf{v}(\partial \boldsymbol{\Omega}_{f'}^l)) \tag{4.32}$$

For the first-order terms and

$$\mathcal{F}_f^k = \mathcal{F}(\mathbf{v}(\partial\mathbf{\Omega}_f^k), \mathbf{v}(\partial\mathbf{\Omega}_{f'}^l), \nabla\mathbf{v}(\partial\mathbf{\Omega}_f^k), \nabla\mathbf{v}(\partial\mathbf{\Omega}_{f'}^l)) \tag{4.33}$$

For the second-order ones. Considering the reference over the face $\partial\mathbf{\Omega}_f^k$, and making use of the notation $-$ for the interior of the element k and $+$ for the exterior, we can write:

$$\mathcal{F}_f^k = \mathcal{F}(\mathbf{v}(\partial\mathbf{\Omega}_f^k)^+, \mathbf{v}(\partial\mathbf{\Omega}_f^l)^-). \tag{4.34}$$

For the first-order terms and

$$\mathcal{F}_f^k = \mathcal{F}(\mathbf{v}(\partial\mathbf{\Omega}_f^k)^+, \mathbf{v}(\partial\mathbf{\Omega}_f^l)^-, \nabla\mathbf{v}(\partial\mathbf{\Omega}_f^k)^+, \nabla\mathbf{v}(\partial\mathbf{\Omega}_f^l)^-). \tag{4.35}$$

As can be noticed, the numerical flux implicitly connect the elemental solutions $\mathbf{v}_h^k$ on both sides of the face. The numerical flux schemes seeks approximated solutions for the Riemann's problem (the discontinuity propagation problem) in order to deal with possible discontinuities at the element interfaces (HESTHAVEN; WARBURTON, 2008, p. 22). In case of continuous solutions, the numerical flux schemes are intended to progress for equaling the solutions at the elemental interfaces.

Considering the expression 4.31, we can introduce the numerical flux terms by directly replacing the surface integrals of the convective and diffusive physical fluxes with the numerical ones as can be seen below

$$\begin{aligned}\boldsymbol{\mathcal{M}}^k \frac{\partial\widehat{\mathbf{v}}^k}{\partial t} = &\int_{\mathbf{\Omega}^k} \left(\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k\right) \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ &+ \oint_{\partial\mathbf{\Omega}^k} \mathbf{n} \cdot {\phi_j}^k \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{k*} + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{k**} + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{k**}\right) \mathrm{d}\mathbf{x} \\ &+ \int_{\mathbf{\Omega}^k} {\phi_j}^k \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \,\mathrm{d}\mathbf{x}, \quad 0 \leq j < N_k,\end{aligned} \tag{4.36}$$

where the superscripts $*$ and $**$ refer to the convective and diffusive numerical fluxes respectively. In the Manticore implementation, they are available the classical convective numerical fluxes HLLC (TORO, 2008, p. 322) and Roe (TORO, 2008, p. 345), as well the diffusive numerical fluxes BR1 (BASSI *et al.*, 2004) and LDG (KIRBY; KARNIADAKIS, 2005).

As stated in 4.7, each element boundary $\partial\mathbf{\Omega}_k$ is composed by a finite number of faces, N_f^k. Therefore, we can split the closed contour integrals into a summation of integrals over the elemental faces, as

$$\oint_{\partial\mathbf{\Omega}^k} \mathbf{n}^k \cdot {\phi_j}^k \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{k*} + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{k**} + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{k**}\right) \mathrm{d}\mathbf{x} = \tag{4.37}$$

$$\sum_{f=1}^{N_f} \int_{\partial\mathbf{\Omega}_f^k} \mathbf{n}^{kf} \cdot {\phi_j}^k \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{kf*} + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{kf**} + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{kf**}\right) \mathrm{d}\mathbf{x} \tag{4.38}$$

In a general case, the faces domains $\partial\mathbf{\Omega}_k^f$ can be high-order curves, that way, we cannot simply pull out the face normal $\mathbf{n}^f$ from the face integrals, since this might depend on the space. After such modifications, the weak form can be written

$$\begin{aligned}\boldsymbol{\mathcal{M}}^k \frac{\partial \hat{\mathbf{v}}^k}{\partial t} = &\int_{\mathbf{\Omega}^k} \left(\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^k - \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^k\right) \cdot \nabla {\phi_j}^k \mathrm{d}\mathbf{x} \\ &+ \sum_{f=1}^{N_f^k} \int_{\partial\mathbf{\Omega}_f^k} \mathbf{n}^{kf} \cdot {\phi_j}^k \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{kf*} + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{kf**} + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{kf**}\right) \mathrm{d}\mathbf{x} \\ &+ \int_{\mathbf{\Omega}^k} {\phi_j}^k \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \,\mathrm{d}\mathbf{x} \quad 0 \leq j < N^k\end{aligned} \tag{4.39}$$

4.7 Local DG-FEM Weak Form

Although considering simple meshes, composed by a single type of element topology, the mesh generation process produces non-uniform cells, each of them with its own shape and size. Moreover, the elements can have different curvature orders, making more complex the evaluation of the operations given in 4.39. In order to avoid that, it is usual to map all the physical elements to an ideal straight sided reference element $\bar{\mathbf{\Omega}}$, also named standard element.

The square reference element is defined such as (KARNIADAKIS; SHERWIN, 2004, p. 93):

$$\bar{\mathbf{\Omega}} : \xi = (\xi_1, \xi_2) \in \bar{\mathbf{\Omega}} \,|\, -1 \leq \xi_1, \xi_2 \leq 1. \tag{4.40}$$

And the triangular reference element (KARNIADAKIS; SHERWIN, 2004, p. 93):

$$\bar{\mathbf{\Omega}} : \xi = (\xi_1, \xi_2) \in \bar{\mathbf{\Omega}} \,|\, -1 \leq \xi_1, \xi_2,\ \xi_1 + \xi_2 < 0. \tag{4.41}$$

For integrating over each elemental region $\mathbf{\Omega}^k$ we must to convert it to the reference region via some variables transformation, such as

$$\mathcal{T}^k(\mathbf{x}, \boldsymbol{\xi}) : \mathbf{x} \rightarrow \boldsymbol{\xi} \,|\, \mathbf{x} \in \mathbf{\Omega}^k, \boldsymbol{\xi} \in \bar{\mathbf{\Omega}}, \tag{4.42}$$

Likewise, let us consider the following mappings involved in the transformation of the general-shaped elemental faces to the reference ones:

$$\mathcal{T}^{kf}(\mathbf{x}, \boldsymbol{\xi}) : \mathbf{x} \rightarrow \boldsymbol{\xi} \,|\, \mathbf{x} \in \partial\mathbf{\Omega}_f^k, \boldsymbol{\xi} \in \partial\bar{\mathbf{\Omega}}. \tag{4.43}$$

In order to integrate within a general-shaped element $\mathbf{\Omega}^k$, which can be mapped to the reference element $\bar{\mathbf{\Omega}}$ via the transformation $\mathcal{T}^k$, we proceed by evaluating the integral in terms of the elemental coordinate system, $\boldsymbol{\xi}$ such as (KARNIADAKIS;

SHERWIN, 2004, p. 157)

$$\int_{\mathbf{\Omega}^k} f(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\bar{\mathbf{\Omega}}} f(\boldsymbol{\xi})|\boldsymbol{\mathcal{J}}^k|\,\mathrm{d}\boldsymbol{\xi}. \tag{4.44}$$

In which $|\boldsymbol{\mathcal{J}}^k|$ is the determinant of the Jacobian matrix $\boldsymbol{\mathcal{J}}^k$, due to the transformation $\mathcal{T}^k$ (KARNIADAKIS; SHERWIN, 2004, p. 158), defined as

$$\boldsymbol{\mathcal{J}}^k = \begin{bmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_1}{\partial \xi_2} \\ \frac{\partial x_2}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_2} \end{bmatrix} \tag{4.45}$$

Let us to consider some integral terms from the equation 4.39 and rewrite them in terms of the local coordinates system. The mass matrix can be rewritten as

$$\boldsymbol{\mathcal{M}}^k \,:\, \boldsymbol{\mathcal{M}}^k_{ij} = \int_{\mathbf{\Omega}^k} \phi^k_j(\mathbf{x})\,\phi^k_i(\mathbf{x})\,\mathrm{d}\mathbf{x}. = \int_{\bar{\mathbf{\Omega}}} \phi^k_j(\boldsymbol{\xi})\,\phi^k_i(\boldsymbol{\xi})\,|\boldsymbol{\mathcal{J}}^k|\mathrm{d}\boldsymbol{\xi}. \tag{4.46}$$

We should notice that even when the basis space Φ is orthogonal, the presence of the Jacobian correction factor modifies the features of $\boldsymbol{\mathcal{M}}^k$, making it a dense matrix. Orthogonality is preserved only when the element curvature id firs-order, corresponding to an affine mapping. The flux terms can be modified likewise

$$\begin{aligned} &\int_{\mathbf{\Omega}^k} \left(\boldsymbol{\mathcal{F}}^k_{\boldsymbol{c}}(\mathbf{x}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{v}}(\mathbf{x}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{t}}(\mathbf{x})\right) \cdot \nabla {\phi_j}^k(\mathbf{x})\mathrm{d}\mathbf{x} = \\ &\int_{\bar{\mathbf{\Omega}}} \left(\boldsymbol{\mathcal{F}}^k_{\boldsymbol{c}}(\boldsymbol{\xi}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{v}}(\boldsymbol{\xi}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{t}}(\boldsymbol{\xi})\right) \cdot \nabla {\phi_j}^k(\boldsymbol{\xi})\,|\boldsymbol{\mathcal{J}}^k|\mathrm{d}\boldsymbol{\xi}. \end{aligned} \tag{4.47}$$

For the source terms, it is possible to easily rewrite the integral expression as a local form:

$$\int_{\mathbf{\Omega}^k} {\phi_j}^k(\mathbf{x})\boldsymbol{\mathcal{S}}^k_{\boldsymbol{t}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_{\bar{\mathbf{\Omega}}} {\phi_j}^k(\boldsymbol{\xi})\boldsymbol{\mathcal{S}}^k_{\boldsymbol{t}}(\boldsymbol{\xi})\,|\boldsymbol{\mathcal{J}}^k|\mathrm{d}\boldsymbol{\xi}. \tag{4.48}$$

Making use of the face mappings defined in 4.43 we can evaluate the contour integrals on the reference element faces:

$$\int_{\partial \mathbf{\Omega}^k_f} \mathbf{n}^{kf} \cdot {\phi_j}^k(\mathbf{x}) \left(-\boldsymbol{\mathcal{F}}^{kf*}_{\boldsymbol{c}}(\mathbf{x}) + \boldsymbol{\mathcal{F}}^{kf**}_{\boldsymbol{v}}(\mathbf{x}) + \boldsymbol{\mathcal{F}}^{kf**}_{\boldsymbol{t}}(\mathbf{x})\right)\mathrm{d}\mathbf{x} = \tag{4.49}$$

$$\int_{\partial \bar{\mathbf{\Omega}}^f} \mathbf{n}^{kf} \cdot {\phi_j}^k(\boldsymbol{\xi}) \left(-\boldsymbol{\mathcal{F}}^{f*}_{\boldsymbol{c}}(\boldsymbol{\xi}) + \boldsymbol{\mathcal{F}}^{f**}_{\boldsymbol{v}}(\boldsymbol{\xi}) + \boldsymbol{\mathcal{F}}^{f**}_{\boldsymbol{t}}(\boldsymbol{\xi})\right)\ |\boldsymbol{\mathcal{J}}^{kf}|\mathrm{d}\boldsymbol{\xi}, \tag{4.50}$$

where $\bar{\mathbf{n}}^f$ represents the normal of face f the reference element. As the reference element is a straight sided polygon, the normal is a vector with constant direction. After these considerations the form

$$\begin{aligned} \boldsymbol{\mathcal{M}}^k \frac{\partial \hat{\mathbf{v}}^k}{\partial\, t} = &\int_{\bar{\mathbf{\Omega}}} \left(\boldsymbol{\mathcal{F}}^k_{\boldsymbol{c}}(\boldsymbol{\xi}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{v}}(\boldsymbol{\xi}) - \boldsymbol{\mathcal{F}}^k_{\boldsymbol{t}}(\boldsymbol{\xi})\right) \cdot \nabla {\phi_j}^k(\boldsymbol{\xi})|\boldsymbol{\mathcal{J}}^k|\mathrm{d}\boldsymbol{\xi} \\ &+ \sum_{f=1}^{N_f} \int_{\partial \bar{\mathbf{\Omega}}^f} \mathbf{n}^{kf} \cdot {\phi_j}^k(\boldsymbol{\xi}) \left(-\boldsymbol{\mathcal{F}}^{kf*}_{\boldsymbol{c}}(\boldsymbol{\xi}) + \boldsymbol{\mathcal{F}}^{kf**}_{\boldsymbol{v}}(\boldsymbol{\xi}) + \boldsymbol{\mathcal{F}}^{kf**}_{\boldsymbol{t}}(\boldsymbol{\xi})\right)\ |\boldsymbol{\mathcal{J}}^{kf}|\mathrm{d}\boldsymbol{\xi} \\ &+ \int_{\bar{\mathbf{\Omega}}} {\phi_j}^k(\boldsymbol{\xi})\boldsymbol{\mathcal{S}}^k_{\boldsymbol{t}}(\boldsymbol{\xi})\,|\boldsymbol{\mathcal{J}}^k|\mathrm{d}\boldsymbol{\xi} \quad 0 \leq j \leq N_k \end{aligned} \tag{4.51}$$

is obtained. For a sake of simplicity, the notation indicating the dependency to the coordinate system is omitted in the further sections.

In order to obtain the approximations for the time-derivatives $\frac{\partial \widehat{\mathbf{v}}_h}{\partial t}$, of the field variables we should invert the mass matrix $\boldsymbol{\mathcal{M}}^k$ for each element. In case of large meshes, the storage of element information can represent a significant cost, given that the matrices need be precomputed and stored in their decomposed form (CHAN *et al.*, 2016).

4.8 Polynomial Expansion

In 4.12 we assumed that the field variables $\mathbf{v}_h^k$ inside each element $\boldsymbol{\Omega}^k$ can be approximated by an expansion over a function space. Thereafter, we constructed an integral form based on this assumption in order to allow . However, no Considering the reference element, each approximated field u_h from $\mathbf{v}_h$ can be expanded basically in two ways:

- The modal approach

$$\mathbf{u}^k(\boldsymbol{\xi}, t) = \sum_{i=1}^{N_k} \widehat{\mathbf{u}}_i^k(t)\phi_i(\boldsymbol{\xi}), \quad \phi_i \in \Phi, \Phi \subset L^2; \tag{4.52}$$

- And the nodal approach:

$$\mathbf{u}^k(\boldsymbol{\xi}, t) = \sum_{i=1}^{N_k} \widehat{\mathbf{u}}_i^k(\boldsymbol{\xi}, t)l_i, \quad l_i \in \mathcal{N}, \mathcal{N} \subset L^2. \tag{4.53}$$

In the modal approach, Φ is a hierarchical polynomial space, with all the spatial information of the expansion term, the coefficients u_i are purely time-dependent. A usual polynomial basis employed on modal expansions is described in terms of Jacobi polynomials. In the case of the nodal expansion, all the polynomials l_i have the same rank and are defined based on points of the domain, denominated nodes. Usual choices in nodal applications are the Lagrange and the Chebyshev polynomials.

In this work it is employed the modal basis which uses the Jacobi polynomials. This choice lies in some convenient properties of this approach, mainly concerning the numerical oscillations controlling, once the hierarchical functions spaces allow more easily to limit the high order modes contribution and avoid unexpected behaviours.

On a two-dimensional expansion, the polynomial space is generated by applying tensor product between two one-dimensional hierarchical polynomial spaces, defined over each reference coordinate axis (ξ_1 and ξ_2, defined in 4.40 and 4.41). Let us consider the most general case, where the one-dimensional expansion spaces admit different maximum polynomial orders for each coordinate axis of the reference element. Hence, we can define the approximation space such as described below.

For the squared regions, we use the full expansion (KARNIADAKIS; SHERWIN, 2004, p. 89):

$$\begin{aligned} &\Phi : \phi_{pq} = \xi_1^p \xi_2^q, (p,q) \in \mathcal{Y}, \\ &\mathcal{Y} : (p,q) \,|\, 0 \leq p \leq N_1, 0 \leq q \leq N_2. \end{aligned} \tag{4.54}$$

For the triangular regions, the Serendipity expansion (KARNIADAKIS; SHERWIN, 2004, p. 89):

$$\begin{aligned} &\Phi : \phi_{pq} = \xi_1^p \xi_2^q, (p,q) \in \mathcal{Y}, \\ &\mathcal{Y} : (p,q) \,|\, 0 \leq p, q \leq N, p+q \leq N \\ &N = max(N_1, N_2) \end{aligned} \tag{4.55}$$

The Serendipity expansion seeks to remove the higher-order modes by limiting the rank of the polynomials Φ_{pq}. The tensor products can be visualized by using the Pascal triangle (KARNIADAKIS; SHERWIN, 2004, p. 89-90).

4.9 Numerical Integration

The terms enclosed in the volumetric integrals of the weak form have been handled in the continuous space, but it is just a theoretical treatment. In fact, for computational purposes, all integral terms need to be discretized and evaluated by means of a quadrature rule, more specifically a Gaussian quadrature. In a general way, considering a function $f = f(\xi)$, the Gauss' quadrature is given by (KARNIADAKIS; SHERWIN, 2004, p. 141):

$$\int_{\bar{\Omega}} f(\xi) \, \mathrm{d}\, \xi = \sum_{q=1}^{N_Q} w_q \, f(\xi_q) + \mathcal{R}(f), \tag{4.56}$$

where $\{w_q\}$ is a set of quadrature coefficients defined for weighting the values of the integrand f evaluated on a set of points $\{\xi_q\}$ of $\bar{\mathbf{\Omega}}$. $\mathcal{R}$ is the integral residue of the approximation. The integral 4.56 will be exactly evaluated (considering the machine precision) for a polynomial integrand if the number of quadrature points is properly chosen (KARNIADAKIS; SHERWIN, 2004, p. 141).

Considering a one-dimensional element (basically a line segment), the set $\{\xi_q\}$ is usually chosen to be the roots of the Legendre polynomial of order N_Q. That special set is referred as Gauss-Legendre (GL) distribution and is represented here as $\{\xi_{\mathcal{L}}\}$. The Gauss-Legendre distribution is chosen because it offers the suitable numerical features for avoiding interpolation error growth (HESTHAVEN; WARBURTON, 2008, p. 49). Another usual choice is the Gauss-Legendre-Lobatto (GLL) distribution, which includes the two extreme coordinates -1 and 1 of the one-dimensional reference element. A GL

quadrature of order N_Q can exactly integrate polynomials $f \in \mathcal{P}_{2N_Q-1}$ and a GLL quadrature of order N_Q polynomials $f \in \mathcal{P}_{2N_Q-3}$ (KARNIADAKIS; SHERWIN, 2004, p. 60). The non-linear terms from the right-hand side of the weak form 4.39 are composed by products and sums between polynomial expansions, so, we are able to determine the correct number of quadrature terms N_Q for accurately evaluating each of them in both the quadrature rules.

Taking the assumption $f \in \mathcal{P}_{2N_Q-1}$ and considering the use of the correct number of expansion terms N_Q for the integral, zeroing the integral residual $\mathcal{R}$, it is possible to rewrite the integral 4.56 as the matrix product:

$$\int_{\bar{\Omega}} f(\xi)\, \mathrm{d}\,\xi = \mathbf{w}^T\mathbf{F}, \tag{4.57}$$

where $\mathbf{F}$ is a column matrix containing the values of f evaluated on $\{\xi_q\}$.

For higher-dimensional geometry, the coordinates $\{\xi_q\}$ are obtained by means of the tensor product between the Legendre distributions along the reference coordinates axis.

For square regions, the tensor product is equivalent to Cartesian product:

$$\begin{aligned} &\{\xi_q\} : (\xi_{\mathcal{L}p}, \xi_{\mathcal{L}q}) \,|\, (p,q) \in \mathcal{X}, \\ &\mathcal{X} : (p,q) \,|\, 0 \leq p \leq N_{Q1},\ 0 \leq q \leq N_{Q2}. \end{aligned} \tag{4.58}$$

In case of triangular elements, we consider convenient to employ the collapsed reference system (KARNIADAKIS; SHERWIN, 2004, p. 93) in order to use the quadrature points described in 4.58.

In both the cases, the total number of quadrature points N_Q is given by:

$$N_Q = N_{Q1}N_{Q2}. \tag{4.59}$$

There is no a priori reason for choosing different numbers of quadrature orders for each reference axis, so, for the sake of simplicity, we will consider the same order for both of them $N_{Q1} = N_{Q2}$.

4.10 Discrete Local Weak Form

Now we will proceed with the construction of the computational structure used in the Manticore implementation. The basic step consists in completely discretizating the integral terms by using the quadrature rule and convert it into matrix form. For this purpose, we need to define some auxiliary operators. Let us define the Vandermonde matrix $\boldsymbol{\mathcal{V}}^k$ for the element $\boldsymbol{\Omega}_k$ as follows (KARNIADAKIS; SHERWIN, 2004, p. 125):

$$\boldsymbol{\mathcal{V}}^k : \mathcal{V}^k_{ij} = \phi^k_i(\xi_j),\ 0 \leq i < N_k, 0 \leq j < N_Q. \tag{4.60}$$

In matrix notation,

$$\mathbf{v}_h^k = \boldsymbol{\mathcal{V}}^k \widehat{\mathbf{v}}_h^k \tag{4.61}$$

The derivative matrix for each element $\boldsymbol{\Omega}_k$ is defined as

$$\mathcal{D}^{\mathbf{k}\xi_\mathbf{l}} : \left(\mathcal{D}^{k\xi_l}\right)_{ij} = \left.\frac{\partial\, \phi_i}{\partial\, \xi_l}\right|_{\xi_j}, \; 0 \leq i < N_k, 0 \leq j < N_Q, \; l \in \{1, 2\}. \tag{4.62}$$

It should be noticed that $\mathcal{D}$ is defined for each reference direction (ξ_1 and ξ_2). We also define the auxiliary Jacobian matrix as:

$$\mathfrak{J}^{\mathbf{k}} : \mathcal{J}_{ij}^k = |\boldsymbol{\mathcal{J}}^k|, \; 0 \leq i < N_Q, \; j \in \{1, 2\}. \tag{4.63}$$

In principle, we could consider that each integral term in 4.39 needs a specific number of quadrature terms in order to be properly evaluated and has its own weights array $\mathbf{w}$ and integration points array. However, that can be unnecessary since we can choose a fixed number of integration points in order to properly integrate every non-linear term. Based on this assumption, we can rewrite the local weak form in matrix notation as

$$\begin{aligned}
\boldsymbol{\mathcal{M}}^k \frac{\partial \widehat{\mathbf{v}}_h}{\partial\, t} =& \mathbf{w}_k^T (\boldsymbol{\mathcal{D}}^{k\xi_1} \circ (\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{k\xi_1} - \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{k\xi_1} - \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{k\xi_1}) \circ \boldsymbol{\mathfrak{J}}^{\boldsymbol{k}}) \\
&+ \mathbf{w}_k^T (\boldsymbol{\mathcal{D}}^{k\xi_2} \circ (\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{k\xi_2} - \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{k\xi_2} - \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{k\xi_2}) \circ \boldsymbol{\mathfrak{J}}^{\boldsymbol{k}}) \\
&+ \sum_{f=1}^{N_f^k} \mathbf{w}_{kf}^T (\boldsymbol{\mathcal{V}}^{\boldsymbol{k}} \circ \left(-\boldsymbol{\mathcal{F}}_{\boldsymbol{c}}^{kf*} + \boldsymbol{\mathcal{F}}_{\boldsymbol{v}}^{kf**} + \boldsymbol{\mathcal{F}}_{\boldsymbol{t}}^{kf**}\right) \mathbf{n}^{kf} \circ \boldsymbol{\mathfrak{J}}^{\boldsymbol{k}}) \\
&+ \mathbf{w}_k^T (\boldsymbol{\mathcal{V}}^k \circ \boldsymbol{\mathcal{S}}_{\boldsymbol{t}}^k \circ \boldsymbol{\mathfrak{J}}^{\boldsymbol{kf}}),
\end{aligned} \tag{4.64}$$

where the operation $\circ$ represents the Hadamard product:

$$\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \circ \boldsymbol{\mathcal{B}} = \mathcal{A}_{ij} \mathcal{B}_{ij}. \tag{4.65}$$

As mentioned in 4.7, in case of curvilinear elements there is a significant computational cost for inverting $\boldsymbol{\mathcal{M}}^k$ and store its decomposed form for large tessellations. In order to circumvent that issue, we can employ the approach proposed by Chan et al. (CHAN *et al.*, 2016), (Weight-Adjusted Discontinuous Galerkin, WADG) by approximating the inverse mass matrix as follows:

$$\left(\boldsymbol{\mathcal{M}}^k\right)^{-1} \approx \widehat{\boldsymbol{\mathcal{M}}}^{-1} \boldsymbol{\mathcal{M}}^{\frac{1}{\mathcal{J}^k}} \widehat{\boldsymbol{\mathcal{M}}}^{-1}, \tag{4.66}$$

where $\widehat{\boldsymbol{\mathcal{M}}}$ is the mass matrix for the reference element and $\boldsymbol{\mathcal{M}}^{\frac{1}{\mathcal{J}^k}}$ is defined as:

$$\boldsymbol{\mathcal{M}}^{\frac{1}{\mathcal{J}^k}} : \boldsymbol{\mathcal{M}}_{ij}^{\frac{1}{\mathcal{J}^k}} = \int_{\bar{\boldsymbol{\Omega}}} \phi_j^k \, \phi_i^k \, \frac{1}{|\boldsymbol{\mathcal{J}}^k|} \mathrm{d}\boldsymbol{\xi}. \tag{4.67}$$

Based on 4.66 and 4.67, we can evaluate and store $\left(\boldsymbol{\mathcal{M}}^k\right)^{-1}$ during the program initialization and access it when necessary, reducing the processing and storage costs.

Expression 4.64 is evaluated for each element k in order to obtain the time-derivatives of field variables for being used in the time-integration process. The state of the neighbor elements and their communication faces are stored for each time-step and accessed when the numerical fluxes from the right-hand side of 4.64 are estimated. Up to now, we have ensured a considerable generality level when considering the possibility of using basis spaces Φ^k specifically chosen for each elemental domain $\boldsymbol{\Omega}^k$.

Although such feature might be important in some specific applications, the validation tests performed in a further chapter of this work (6) uses a same basis space for all the elements of the domain tessellations.

4.11 Time-Integration

There are fundamentally two ways for performing time-integration, the explicit and the implicit techniques. The most conventional way is the explicit approach, where the field variable vector is evaluated for the next instant just depending of the present state. Therefore,

$$\mathbf{v}_h^{j+1} = \mathcal{I}\left(\mathbf{v}_h^j, \frac{\partial \widehat{\mathbf{v}}_h}{\partial\, t}, \mathrm{d}t\right), \tag{4.68}$$

where $\mathcal{I}$ represents a generic explicit time integrator. In certain cases, such as well-behaved laminar flows over low order geometric curvatures, it has been empirically verified that the explicit approaches are more convenient to the problem time-evolution. However, the Navier-Stokes and the RANS equations present severe stiffness (BIRKEN, 2012) (CONTENT *et al.*, 2013) as the number of DOFs is increased. This partly arises from the multi-scale characteristic of the turbulence problem, given that for respecting the time-integration stability region of the problem, the spatial and time discretization should be refined in order to contain the fastest eddy scales, which can imply in a major computational barrier. Furthermore, when solving steady-state problems we are interested in skipping intermediary states and as rapidly as possible to achieve the stationary solution. In order to do that, we need to be able to use larger time-steps without undermining the field variables accuracy. An approach developed to match this requirements is the fully implicit time integration, which uses present and predicted information, which enables unconditional stability with respect to the time step.

The implicit integration has the following general structure

$$\mathbf{v}_h^{j+1} = \mathcal{I}\left(\mathbf{v}_h^j, \mathbf{v}_h^{j+1}, \frac{\partial \widehat{\mathbf{v}}_h}{\partial\, t}, \mathrm{d}t\right). \tag{4.69}$$

It is worth noting that each integration step results in a nonlinear system. In order to solve this, it is necessary to employ iterative algorithms, as the Newton-Raphson's varieties, as a GMRES (BIRKEN, 2012) and JFNK (Jacobian-Free Newton-Krylov) (CONTENT *et al.*, 2013) schemes.

The fully implicit approach has been used to solve steady-state problems, (OLIVER, 2008) and (LANDMANN, 2008). Nevertheless, (BASSI *et al.*, 2004) and (NGUYEN *et al.*, 2007) applied this strategy both for steady and unsteady problems.

At this point the question is mostly computational. Despite these, (BIGARELLA, 2007) accomplished all the integrations using an artificial viscosity aided explicit addressing, but restricted to the low order domain.

The time-integration of the field and turbulence variables are performed in a coupled way, that is, the approximated field vector $\mathbf{v}_h$ also includes the turbulence variables and all the residual evaluations are simultaneoulsy performed. Although the literature mentions schemes to solve the uncoupled RANS equations, evaluating the field and the turbulence variables separately, these methods ensure converged results just for conditions which are close to the equilibrium, showing difficulties in dealing with unsteady flows (CONTENT *et al.*, 2013).

Manticore offers pre-implemented options of time-integrators, some of them, used on this work, are briefly described in the following. For the explicit integration, The strong-stability Runge-Kutta 54 (SSRKP54) (HESTHAVEN; WARBURTON, 2008, p. 157), with five steps and 4th-order error. For the implicit case, a standard Newton-GMRES backward-Euler.

4.12 Positivity Limiting

Even though the imposition of conservation laws throughout of the RANS solving process, the numerical approximation is not completely safe of arising non-physical values for the turbulent variables during the time-integration. The eddy viscosity can became negative in some regions, such as zero-valued regions and regions where the source terms are very intensive, given that the production and destruction terms can achieve high values for small wall distances. During the simulation start-up, the occurrence of negativity in the turbulent variables becomes more frequent due to the great unbalance produced by the bad fields initialization (such as homogeneous initialization).

Considering the devastating impact of non-physical values propagation all over the simulation domain, it is mandatory the positivity enforcement via some auxiliary scheme. The approach adopted in this work is a reconstruction-type technique denominated hard limiting (LANDMANN, 2008) for the stepwise and the linear cases.

The stepwise case corresponds to the interpolation order $p = 0$, either for the quadrilateral and the triangular meshes, in which there is just one degree of freedom $\tilde{\nu}_0$ by field variable in each element, equivalent to the mean value over it. If $\tilde{\nu}_0 \leq 0$, hard limiting consists simply in limiting it by imposing some restriction $\tilde{\nu}_0 = \epsilon$, where ϵ is a small value chosen to avoid the negativity without adversely interfering in the vicinity elements evaluation. The reference (LANDMANN, 2008) uses $\epsilon = 0$, but in this work the value of ϵ set up in the tests is 10^{-15}.

It is necessary to notice that as the stepwise limiting interferes in the mean value of the element, it affects the elementwise conservation law. However, it is expected that this interference be proportional to ϵ when the expected $\tilde{\nu}$ is near to zero, or, otherwise, it is assumed that the limiting scheme was able of conducing the solving process to the proper values and is no longer active if the numerical implementation was properly done.

The linear limitation is applied to expansions with $p \leq 1$ and consists of adjusting the coefficients of the linear modes in order to avoid the negativity inside the element, it corresponds to avoid the solution becomes lower than zero in the element corners (LANDMANN, 2008). For the triangular elements, the positivity algorithm is (LANDMANN, 2008, pp. 44)

$$\begin{aligned} &\textbf{if}: \tilde{\nu}_0 < 0, \textbf{set}: \tilde{\nu}_0 = \tilde{\nu}_1 = \tilde{\nu}_2 = \epsilon \\ &\textbf{if}: \tilde{\nu}_0 \geq 0, \textbf{set}: \tilde{\nu}_0 \leq \tilde{\nu}_0 \textbf{ and } \tilde{\nu}_2 \geq -\frac{\tilde{\nu}_0}{2} \\ &\qquad\qquad \tilde{\nu}_1 \leq \tilde{\nu}_0 - \tilde{\nu}_2 \textbf{ and } \tilde{\nu}_1 \geq \tilde{\nu}_2 - \tilde{\nu}_0 \end{aligned} \tag{4.70}$$

and for the square regions (LANDMANN, 2008, pp. 44)

$$\begin{aligned} &\textbf{if}: \tilde{\nu}_0 < 0, \textbf{set}: \tilde{\nu}_0 = \tilde{\nu}_1 = \tilde{\nu}_2 = \epsilon \\ &\textbf{if}: \tilde{\nu}_0 \geq 0, \textbf{set}: \tilde{\nu}_1 = s\tilde{\nu}_1 \textbf{ and } \tilde{\nu}_2 = s\tilde{\nu}_2 \\ &\qquad\qquad s = \frac{-\tilde{\nu}_0}{min(-\tilde{\nu}_1 + \tilde{\nu}_2, -\tilde{\nu}_1 - \tilde{\nu}_2, \tilde{\nu}_1 - \tilde{\nu}_2, \tilde{\nu}_1 + \tilde{\nu}_2)}, \ 0 \leq s \leq 1. \end{aligned} \tag{4.71}$$

In which the $\tilde{\nu}_0$, $\tilde{nu}_1$ and $\tilde{\nu}_2$ represents the coefficients of the modal expansion, respectively the constant and the linear modes. For elements with modes ranks higher than 1, the problem of correctly reconstructing the interior of the element avoiding negativity becomes more cumbersome and requires more computationally expensive approaches (LANDMANN, 2008), given that, the addressing chosen in this work is to truncate the higher order modes and impose the hard limiting over the lower ones. Once again, it is necessary to stress that the hard limiting does not substantially affect the conservation laws in the linear case if the mean value be proportional to the tolerance ϵ or becomes completely inactive in any time, as assumed beforehand.

5 NUMERICAL SOFTWARE IMPLEMENTATION

5.1 Overview

The software developed in this work was implemented by using the framework Manticore, a free and open-source numerical engine fully-developed in Python 3 and devised to operate in FEM-based applications. Besides the DG-FEM approach for RANS problems, it was necessary to implement specialized pre and post-processing operations as well as auxiliary stabilization schemes in order to enable running the numerical tests. This work have extended Manticore by a development branch specialized in CFD, but the Manticore's scope is a general purpose framework. This chapter intends to summarize the stages of such implementations by exposing the modules organization whilst introduces the Manticore's architecture. It is important to notice that all the references to directories paths are based in the main directory manticore, created in the local host when the software is downloaded from the remote repository.

5.2 Manticore's General Architecture

Manticore is object-oriented and its core is hierarchically organized in order to maintain encapsulated the increasing complexity of the implementation stages. In other words, we can handle any high-level operation by a set of lower level pre-constructed objects. For instance, we can handle a numerical flux boundary integral without directly dealing with polynomial expansions or quadrature rules, but just invoking the suitable classes.

Such complexity level hierarchy allows to create new operators or terms in an equation in a simple way: constructing dedicated classes for the specific operations and instantiating them in the proper places. Thus, implementing a problem in a framework such as Manticore is most of time, comparable to assemble prefabricated parts according to some established logic. Naturally, the software encapsulation can eventually hide the fundamental operations at a level in which the code loses legibility and becomes difficult to track origin and content of some objects. However, such matter can be bypassed by properly employing some basic software engineering techniques.

5.3 Input Reading

Each problem simulated in Manticore at the current development state needs three input files in order to be executed, namely geometrical mesh file, initialization file and set up file. The mesh file is originally written in ASCII format using a mesh generator

program, after that is converted to a binary HDF5-based format before being read by the simulation software. This work employed the software Gmsh as mesh generator and implemented the scripts to convert the Gmsh's text format to the default MDF (Manticore data format), a binary format built from HDF5 and already specified when this work was started.

The MDF format seeks to enforce generality by considering a wide scope of possibilities. The input reading resources available in Manticore was constructed to cover cases in which different regions of the problem domain could be modelled by specific systems of equations, for instance, multiphase flow combining solid, liquid and gaseous fluids, each of them with a specific nature and hence a given mathematical model, that way, the mesh file is subdivided in groups related to each modelling regions, which store geometrical and physical information, at the set up time. All the tests performed in this work are single phase, so, there is just one group in the mesh files. The groups of the mesh files also can be split into partitions in order to be used in parallel executions, yet that is not the case of this work, since all tests are serially performed. In other words, there is just one partition per group whose elements are assessed at a time. The serial execution was used since the MPI parallelism was not complete at the time of this work.

The initialization file is a MDF file which stores the initial state of the field variable for each element and group in the problem domain. The set up file is a Python 3 script which controls simulation workflow, determining the mesh and initialization files, the name of the domain and boundary groups, the modelling system of equations, the choice of the turbulence models, numerical fluxes and time-integrators. An example of set up file is available at manticore/models/turbulence/tests/manufactured_solution_setup.py, where there are multiple test cases in terms of Python functions.

5.4 The Manticore's Numerical Engine

After reading the input information the solver starts the construction of the computational mesh, in which each geometrical element is enriched with an interpolation ansatz and vicinity information. The Manticore numerical engine has been constructed and tested following the philosophy of ensuring the locality at most when evaluating the numerical operators, based on that, the fundamental entity, used as basis for constructing the entire formulation, is the elementwise expansion and able of accessing the most part of the information necessary during the solving process. The class ExpansionEntity defined in the module manticore/lops/entity.py is the implementation of an elementwise expansion and its communication vicinity, taking the role of the element entity itself. The block of code in the Listing 5.1 shows an example of element class entity, in which the information of some physical fields is reconstructed and stored in temporary arrays.

Listing 5.1 – A short block of code for showing the usage of the element entity class.

```
        rho = e.state(FieldType.ph, FieldVariable.RHO)
        dnudx = e.state(FieldType.ph, FieldVariable.DNUIDX)
        dnudy = e.state(FieldType.ph, FieldVariable.DNUIDY)

        np.power(dnudx, 2, out=TurbModelWsp_v.flux_aux1)
        np.power(dnudy, 2, out=TurbModelWsp_v.flux_aux2)
```

A more detailed view of the element entity class and its methods can be seen in the Section A.1 of the Appendix A. From the physical modelling standpoint, the core entities are the class Equation and its extensions, defined in manticore/models/compressible/equations/equation.py. The classes based on Equation basically defines the primitive and conservative variables to be modelled and the characteristic right-hand side residue used to describe the problem. The residue is defined in a specific class and introduced as an attribute of the equation class during the set up.

The equation terms operate over element entities sets in order to evaluate the numerical residue of each element individually, since all the necessary information, including the face neighbourhood state, can be obtained from it. The characteristic equation expressions, such as fluxes and forcing terms, are defined as dedicated Python classes, in which the __call__ method (a method invoked when the class is executed as a function) receives an element entity, evaluates the mathematical expressions for the integration points and returns the values necessary to perform the time-integrations. A view of the general structure for constructing the convective residue class can be seen in Listing 5.2.

Listing 5.2 – General structure of the factory for constructing the convective residue

```
def factory_weak_dg_convres(conv_flux_t, turb_model_t):
    """Factory for the residual of the weak DG form of the Euler equation.
    Args:
        conv_flux_t: A template parameter on the kind of convective
            flux we are using (Lax-Friedrich, Roe, HLLC, etc).
    """
    Workspace_f = conv_flux_t.Workspace_f
    InverseMassOperator = [class_wadg_inverse_mas_op,class_rwadg_inverse_mas_op]

    TurbModel_v = turb_model_t.TurbModel_v
    TurbModel_f = turb_model_t.TurbModel_f

    class weak_dg_convective_residual(
            EntityOperator,
            model_description_mixin,
            equation_domain_signature,
            expansion_numbers):

        def __init__(self, model_desc):
            model_description_mixin.__init__(self, model_desc)
            equation_domain_signature.__init__(self)
            expansion_numbers.__init__(self)
            EntityOperator.__init__(self)

            # Initializing the class global variables
```

```
26
27          def setup(self, eqname, domain):
28
29              equation_domain_signature.setup(self, eqname, domain)
30              matname = self.desc.get_equation(eqname).mat
31              self.fluid = self.desc.get_material(matname)
32
33              Specific test case statements
34
35          def container_operator(self, group, key, container):
36
37              # Recovering information from the expansion
38              entities.
39
40          def __call__(self, e):
41
42              # Evaluates the convective residual of a single element.
43              # Invoke the auxiliary methods and manages the entire process
44
45          def eval(self, e):
46
47              # Evaluates the residual without the inverse mass matrix.
48              """
49              Note:
50
51                  * This residual evaluation 'kernel' is useful for
52                      combining with more complex residuals (e.g. artificial
53                      dissipation, viscous flow, etc.)
54              """
55              # Filling the primitive variables workspace
56              Primitive_v = PrimitiveWsp_v.get_instance(self.nQ)
57              Primitive_v.fill(e, self.fluid)
58
59              # Zeroing residuals
60              Residual_p = ResidualWsp_p.get_instance(self.nS)
61              Residual_p.zero()
62
63              # Volumetric operations
64              self.volumeContribution(e)
65
66              # Facial operations
67              self.faceContribution(e)
68
69          def volumeContribution(self, e):
70
71              # Evaluation of values related to the volumetric flux and source integrals
72
73          def faceContribution(self, e):
74
75              # It access information about the choice of numerical fluxes
76              # in order to evaluate them.
77
78              for ff in range(num_faces):
79
80                  # Evaluates values of the boundary integrals for each
81                          elemental face
82
83      # end of factory_weak_dg_convres.weak_dg_convective_residual
84
```

```
85 |     return weak_dg_convective_residual
```

In the previous code, lines are suppressed in some parts and replaced by short explanations in order to compress the original long file and present just the main code structure. The values of the flux terms are used to evaluate the domain and boundary integrals using Gaussian quadrature, as seen in the code Listing 5.3, in which the method volumeContribution of the Listing 5.2 is detailed.

Listing 5.3 – Volumetric contribution of the convective terms

```
1  def volumeContribution(self, e):
2      Primitive_v = PrimitiveWsp_v.get_instance(self.nQ)
3      Residual_p  = ResidualWsp_p.get_instance(self.nS)
4      #Workspace for turbulence facial auxiliar variables
5      TurbModelWsp_v = TurbModel_v.get_instance(self.nS, self.nQ)
6
7      J  = e.get_jacobian()
8      IM = e.get_inv_jacobian_matrix()
9
10     #
11     # FIRST EQUATION: rho * [u, v]
12     #
13     rho = e.state(FieldType.ph, FieldVariable.RHO)
14     ConvectiveFlux_1_X(rho, Primitive_v.u, self.Fx )
15     ConvectiveFlux_1_Y(rho, Primitive_v.v, self.Fy )
16     # S operator, physical -> modal
17     self.innerpg[e.type](J, IM, self.Fx, self.Fy, Residual_p.rho )
18
19     #
20     # SECOND EQUATION: [rhou*u+p, rhou*v]
21     #
22     rhou = e.state(FieldType.ph, FieldVariable.RHOU)
23     ConvectiveFlux_2_X(rhou, Primitive_v.u, Primitive_v.p, self.Fx )
24     ConvectiveFlux_2_Y(rhou, Primitive_v.v,                self.Fy )
25     # S operator, physical -> modal
26     self.innerpg[e.type](J, IM, self.Fx, self.Fy, Residual_p.rhou )
27
28     #
29     # THIRD EQUATION: [rhov*u, rhov*v+p]
30     #
31     rhov = e.state(FieldType.ph, FieldVariable.RHOV)
32     ConvectiveFlux_3_X(rhov, Primitive_v.u,                self.Fx )
33     ConvectiveFlux_3_Y(rhov, Primitive_v.v, Primitive_v.p, self.Fy )
34     # S operator, physical -> modal
35     self.innerpg[e.type](J, IM, self.Fx, self.Fy, Residual_p.rhov )
36     #
37     # FOURTH EQUATION: (e+p)*[u, v]
38     #
39
40     aux = e.state(FieldType.ph, FieldVariable.RHOE) + Primitive_v.p
41     ConvectiveFlux_4_X( aux, Primitive_v.u, self.Fx )
42     ConvectiveFlux_4_Y( aux, Primitive_v.v, self.Fy )
43     # S operator, physical -> modal
44     self.innerpg[e.type](J, IM, self.Fx, self.Fy, Residual_p.rhoe )
```

self.innerpg is a list of wrapper methods for the classes defined in manticore.lops.modal_dg.phyops2d

which performs inner product for Gaussian quadrature. The output of the self.innerpg methods fills the arrays of Residual_p, a Manticore workspace, basically a class containing sets of arrays used to store temporary values, for the right-hand side of each equation in the modal coefficients space.

On the top of this scheme, there is a time-integrator class, which evaluates the residue terms at each iteration and updates the values of the field variables. The explicit time-integrators are defined in the module manticore/models/compressible/timeint/integrator.py and the implicit time-integrators in manticore/models/compressible/timeint/implicit.py. After updating the field variables state, the values of that are stored in attributes arrays inside the elemental entity.

The choice for operating locally and distributing the processing is not mandatory. The numerical engine is flexible enough to cover cases in which all the element domains are grouped into a global system and the complete set of degrees of freedom solved at a time. The choice between distributed memory and shared memory ends up to the classical trade-off between memory and CPU, being both of them valid according to the goals of the implementation and viable from the Manticore's numerical engine viewpoint.

5.5 The RANS Structure and Add-ons

The Manticore's branch for Navier-Stokes simulations covers the evaluation of convective ($\boldsymbol{\mathcal{F}}_c$) and viscous ($\boldsymbol{\mathcal{F}}_v$) flux terms. In order to construct a structure for simulating RANS-turbulent problems it was necessary to implement modules for assessing the turbulent flux ($\boldsymbol{\mathcal{F}}_t$) and the turbulent source ($\boldsymbol{\mathcal{S}}_t$) terms.

The turbulent residue (the turbulent contribution in the form of flux and source terms) is defined in manticore/models/turbulence/equations/turbres.py and works as a main module for organising the evaluation of the turbulent residue terms by means of the class weak_dg_turbulent_res.py. The turbulent flux term is evaluated by using the same structure of the viscous ones applied to the considered characteristic equations. The turbulent residue class is shown in the Section A.3 of the Appendix A.

The source terms are defined in the module manticore/models/turbulence/equations/turbsource.py by the source_terms_res class. The problem definition classes could be modified according to specific conditions chosen at the simulation set up. A basic polymorphism should be ensured and it is done through factory functions. In this case, if-else conditional statements are used in order to properly return the problem classes with the necessary features required for the problem conditions. An example of factory function is shown in the code Listing 5.4.

Listing 5.4 – General structure of a factory function used to construct the source terms class.

```
def factory_turbulent_source(feature):

    if  feature=='tripped_evolution':

        #For cases considering the transitional state

        class source_terms_res:

            def __init__(self, model_desc, fluid):
                self.model_desc = model_desc
                self.fluid = fluid
                # others initialization commands

            def _others_methods(self, *args, **kwargs):
                # commands

   elif feature=='fully_turbulent':

        #For cases without considering the transitional state

        class source_terms_res:

            def __init__(self, model_desc, fluid):
                self.model_desc = model_desc
                self.fluid = fluid
                # others initialization commands

            def _others_methods(self, *args, **kwargs):
                # commands
   else:
            raise AssertionError("Case not covered!")
```

The content of the flux and source terms classes is basically a transcription of the expressions seen in the Chapter 3 to Python code using the Manticore structures, as exemplified in the code Listing 5.5, in which the first source term (here named dissipation term) of equation 3.28 is implemented.

Listing 5.5 – Source dissipation implementation

```
def __call__(self, e, TurbModelWsp_v,  TurbPrimWsp_v, PrimitiveWsp_v=None):

    D = self.D.get(e.ID)
    TurbPrimWsp_v.fill(e, self.fluid)

    ### Source dissipation

    nu = TurbPrimWsp_v.nu

    #(cb2/sigma)*{rho*[grad(nu).grad(nu)]}
    rho = e.state(FieldType.ph, FieldVariable.RHO)
    dnudx = e.state(FieldType.ph, FieldVariable.DNUIDX)
    dnudy = e.state(FieldType.ph, FieldVariable.DNUIDY)

    np.power(dnudx, 2, out=TurbModelWsp_v.flux_aux1)
    np.power(dnudy, 2, out=TurbModelWsp_v.flux_aux2)

    np.add(TurbModelWsp_v.flux_aux1, TurbModelWsp_v.flux_aux2, \
    out=TurbModelWsp_v.flux_aux3)
```

```
20 |     np.multiply(rho, TurbModelWsp_v.flux_aux3, out=TurbModelWsp_v.nu_dissip)
21 |     TurbModelWsp_v.nu_dissip *= (self.cb2/(self.Re*self.sigma))
```

The values of the flux and source terms are evaluated for each integration point at the element entity e, after that the integral terms of the weak formulation are calculated via Gaussian quadrature. As mentioned in Section 4.12, the solving process can violate the positivity of the turbulent variables when seeking the steady-state solution and in order to control this issue it was purposed the use of positivity limiting. The limiting algorithm is implemented in manticore/models/turbulence/equations/limiters.py and applied as a post-processing operation over the computational mesh at each iteration of the time-integration process. See the Section A.4 of the Appendix A to an in-depth view of the limiter implementation.

5.6 Post-processing the Output

After the convergence has been achieved, the coefficient fields stored in each element are recovered and used to reconstruct the physical fields in order to be employed for visualization and error assessment purposes. Both the error evaluation and visualization modules are implemented as complementary tools of this work and bequeathed to the Manticore project. The error metric class, as well as other post-processing operations, is located in the module manticore/models/turbulence/manager/post_process.py. The pointwise error in each element is evaluated for all the integration points and subsequently integrated using the Gaussian quadrature rule, after that the values of all the elementwise errors are summed. The visualization tool works in two ways, plotting the elementwise fields one at a time, thus allowing the highlighting of discontinuities between elements, and the global plot, where the field is smoothed by the inter-elemental interpolation. This second option in convenient when the computational mesh resolution is still too coarse to ensure a good visualization. As all the examples tests in the validation section of this work are smooth and regular, the choice between the two visualization approaches is not crucial. All the visualization engine implementations are in manticore/visualization/PlotUtils.py.

6 TESTS AND VALIDATION

6.1 Overview

In order to verify the correctness of the implementation accomplished in this work, we should to incrementally validate the different mathematical components of the RANS system, starting with the inviscid Euler equations and subsequently adding the viscous and turbulent terms, thus checking the overall behavior. That way, the validation of the framework here implemented is performed in three stages:

- Inviscid terms (Euler) validation.
- Viscous terms (Laminar Navier-Stokes) validation.
- Turbulent terms (Spalart-Allmaras RANS) validation.

The Inviscid validation is performed using two usual test cases: the isentropic vortex and the smooth bump. Both the cases, despite being idealizations, are physically grounded problems. The Viscous and turbulent validation are performed using manufactured solutions (ROY *et al.*, 2007), since the complexity of the realistic problems on this matter demands hardware and software infrastructure still not available for this work.

6.2 Validation of the Convective Terms

6.2.1 Overview

The first step for testing the numeric implementation is to validate the evaluation of the inviscid convective term ($\boldsymbol{\mathcal{F}_c}$). In order to accomplish that, were employed the Euler equations for modelling two reference problems, the isentropic vortex (HESTHAVEN; WARBURTON, 2008, 209) and the inviscid Gaussian bump (GALBRAITH, 2011). The main purpose was to verify the numerical approximation accuracy, robustness in dealing with mesh curvature and adverse boundary conditions.

6.2.2 The Isentropic Vortex Test Case

The first validation test case is the time-dependent isentropic vortex problem, which has the exact solution given by (HESTHAVEN; WARBURTON, 2008, p. 209).

$$u = 1 - \beta\, e^{(1-r^2)} \left(\frac{y - y_0}{2\pi}\right), \tag{6.1}$$

$$v = \beta\, e^{(1-r^2)} \left(\frac{x - x_0}{2\pi}\right), \tag{6.2}$$

$$\rho = \left[1 - \left(\frac{\gamma - 1}{16\gamma\pi^2}\right) \beta^2 e^{2(1-r^2)}\right]^{\left(\frac{1}{\gamma-1}\right)}, \tag{6.3}$$

$$p = \rho^{\gamma}, \tag{6.4}$$

where $r = \sqrt{(x - t - x_0)^2 + (y - y_0)^2}$, $x_0 = 5$, $y_0 = 0$, $\beta = 5$, $\gamma = 1.4$. This test case already was implemented in the Manticore framework and is presented here just for validation purposes.

The problem domain is a square of side $L = 10$, whose boundaries are under Dirichlet conditions based on the known solution. The time-integration is performed by a Strong-Stability Runge-Kutta of five steps and 4th-order accuracy](HESTHAVEN; WARBURTON, 2008, p. 157). A sequence of tests was performed in order to evaluate the robustness of inviscid flow solver. The computational grids are composed of $N \times N$ uniform elements with length of side $h = L/N$, where N ranges from 5 to 20.

Tables 6.1 and 6.2 present the continuous L^2-normed error for each combination (h, p) (minimum element size and polynomial interpolation order) for the variables ρ and ρu.

Table 6.1 – L^2-error for the variable ρ

		p				
		1	2	3	4	5
h	2	4.00×10^{-1}	1.48×10^{-1}	1.10×10^{-1}	3.99×10^{-2}	1.69×10^{-2}
	1	1.49×10^{-1}	5.65×10^{-2}	7.80×10^{-3}	2.72×10^{-3}	7.74×10^{-4}
	0.5	8.22×10^{-2}	4.91×10^{-3}	6.15×10^{-4}	1.02×10^{-4}	1.24×10^{-5}

Table 6.2 – L^2-error for the variable $\rho\, u$

		p				
		1	2	3	4	5
h	2	6.36×10^{-1}	2.81×10^{-1}	1.51×10^{-1}	6.19×10^{-2}	2.80×10^{-2}
	1	2.72×10^{-1}	8.79×10^{-2}	1.44×10^{-2}	5.71×10^{-3}	9.56×10^{-4}
	0.5	4.75×10^{-2}	9.15×10^{-3}	1.50×10^{-3}	1.32×10^{-4}	2.76×10^{-5}

It is possible to see that the error is reduced as the number of elements and the order are increased for both the variables, validating the capability of the inviscid

solver in dealing with the transient problem. The error regarding the variables ρv and ρE are not reduced by the *hp*-refinement. The reason for that cannot be inferred given that the literature reference (HESTHAVEN; WARBURTON, 2008, p. 210) does not present results for such variables.

6.2.3 The inviscid smooth bump problem

The second test case consists of a steady-state compressible inviscid flow over a smooth bump inside a channel. Air moves into the channel at a Mach number $M_\infty = 0.5$ coming from a non-disturbed freestream flow, where the pressure and temperature conditions are stationary (see Figure 6.1). As there is no friction dissipation or heat transfer in the system, the flow is ideally isentropic. The conditions at the inlet are summarized in the Table 6.3.

Table 6.3 – Inlet flow conditions.

Inlet Pressure (p_∞)	$1,01kPa$
Inlet Temperature (T_∞)	$303K$
Inlet Mach Number (M_∞)	0.5

A slip wall boundary condition is imposed on the superior channel and the bump surfaces. The stagnation pressure is fixed in the inlet based on the freestream flow, and the conditions in the outlet are modelled based on a backward state criterion. The density and momentum are equalled to that of the internal neighbour element face and the energy are limited to be evaluated at a fixed static pressure in case of locally supersonic flow.

In order to starting the solution procedure an uniform field initial condition is set up and the implicit time integrator scheme is left to run until the achievement of the convergence criterion.

The test has been performed employing the convective numerical flux of Roe. An implicit Standard Newton-GMRES integrator is used for conducing the scheme to the convergence, each time iteration is solved by using a matrix-free GMRES method preconditioned by a diagonal block LU approach. The implicit time-integration is crucial on such kind of problem, due the intrinsic numerical stiffness of that in accommodating the steady-state solution to the curved slip boundary, mainly in the start from a non-physical condition. Explicit approaches would require very limited CFL choices in order to enforce stability.

6.2.4 Testing workflow

An interpolation order (p) refinement sequence of tests was performed using a 48 quadrilateral elements mesh with 4th-order curvature. The test array was initiated

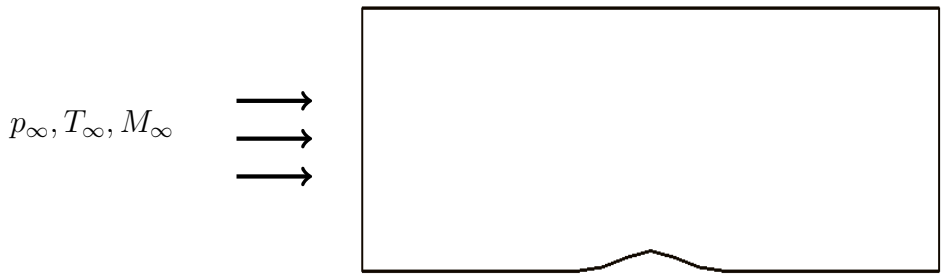

Figure 6.1 – The inviscid flow testing domain.

using $p = 0$, similar by to a finite volume implementation, and continued until p = 6. The result of each p was used for initializing the next refinement, so, the stiffness observed on the higher orders was reduced by the enhancement of the initial condition.

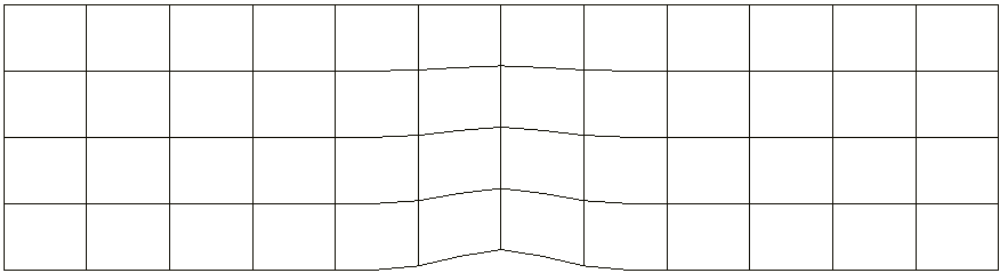

Figure 6.2 – Quadrilateral mesh used in the bump test case.

It is important to stress that using a quadrilateral mesh, the maximum rank of the expansion modes is equal to $2p$, therefore, in case of using $p = 6$, we have monomials with rank at most 12 for the interpolation of the field variables. The CFL number was ramping up as the numerical residue decreased, ranging from 50 to 150 (values empirically determined). Results for $p = 5$ can be seen in Figure 6.3.

All the tests were performed using a workstation computer equipped with an octa-core Intel i7 processor. Such feature enabled the native multithreading parallelism of the Python library Numpy, extensively used by Manticore. Multithreading is specially effective when running the higher interpolation and curvature orders, given that is set up for processing computationally expensive matrix operations.

6.2.5 Validation

Considering that the bump test case does not have an exact solution, the accuracy of the results were evaluated by using the dimensionless entropy error L^2-norm (YANO; DARMOFAL, 2018), defined as:

$$E_{L^2} = \sqrt{\frac{1}{\mathcal{V}} \int_{\mathbf{\Omega}} \left[\frac{p}{p_\infty} \left(\frac{\rho_\infty}{\rho} \right)^\gamma - 1 \right]^2 \mathrm{d}\,\mathcal{V}}, \tag{6.5}$$

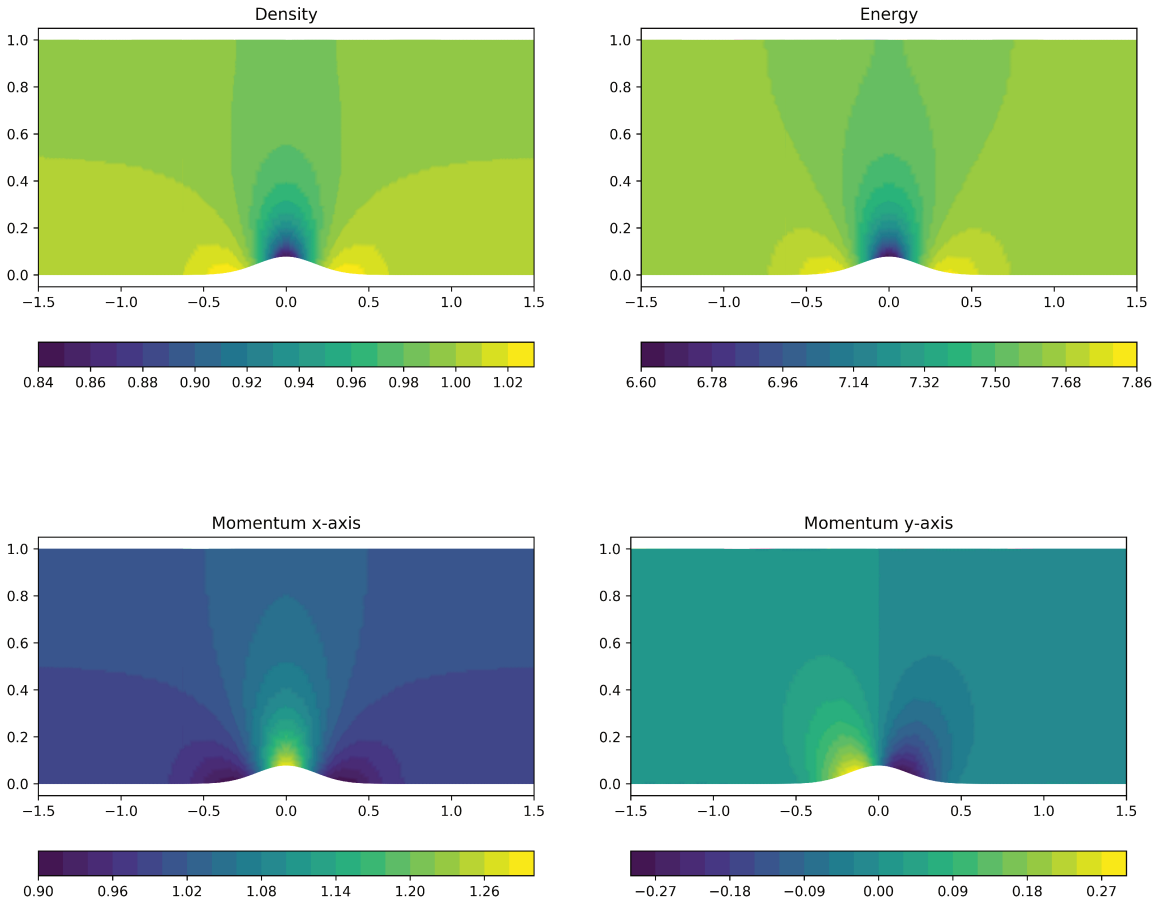

Figure 6.3 – Solution for 48 elements and $p = 6$.

where p and ρ are, respectively, the pressure and density fields inside the flow domain, γ the fluid compressibility constant and $\mathcal{V}$ the domain volume. This norm measures the total entropy generation produced by the numerical approximation when compared to the inlet condition given that it implicitly verifies the preserving of isentropic flow relations. The entropy error curve is plot in the Figure 6.4, in terms of the number of degree of freedom ($NDOF$), that corresponds to the total number of coefficients used in the elemental expansions.

Figure 6.4 shows the decreasing of the entropy error in terms of the number of DOFs, from a coarse approximation with p = 0 to an already significantly low numerical interference in p = 6. The error orders correspond to that observed in the benchmark literature for the same range of the NDOF parameter, as can be seen on Wang et al. (Wang *et al.*, 2014) and Kast (KAST, 2013). The decreasing curve profile validates the capacity of the numerical implementation in solving a complex inviscid compressible flow problem using p-refinement.

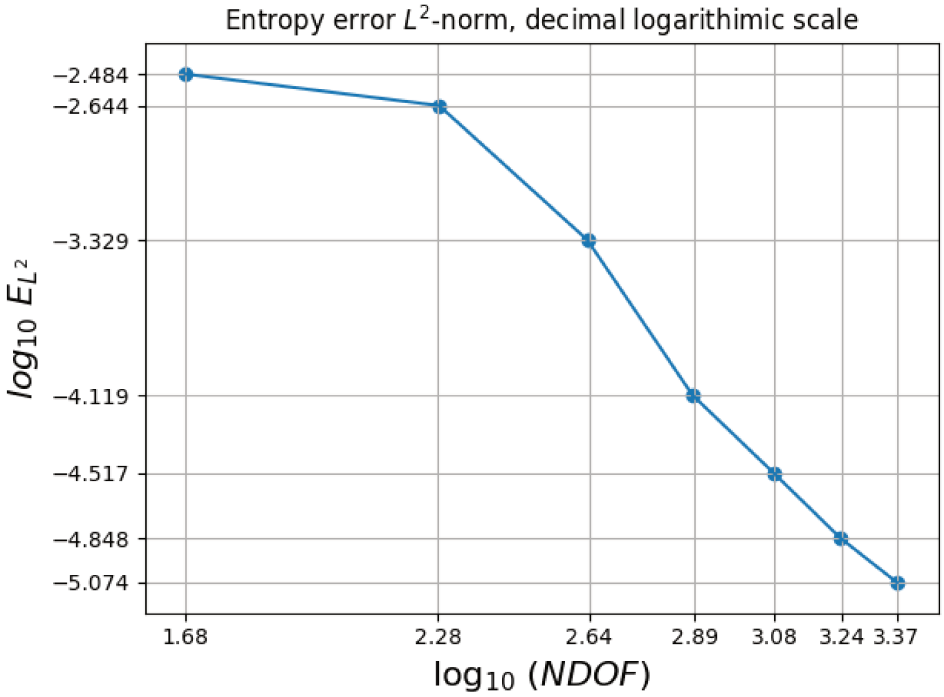

Figure 6.4 – Comparison of the entropy error norm to the number of degrees of freedom.

6.3 Validation of the Viscous Terms

6.3.1 Overview

For the validation of the viscus term ($\boldsymbol{\mathcal{F}}_v$) was used a manufactured problem (ROY *et al.*, 2007). There is main reason for choice of the manufactured expressions, the limited capability of the Manticore engine to deal with both large meshes and high orders for realistic problems, which could preclude the p-refinement analysis. In this way, the manufactured solutions arise as a simple technique for evaluating the correctness of the implementation without excessive problem-biased issues.

6.3.2 Sinusoidal Manufactured Solution for the Navier-Stokes Equations

The following manufacture solution is considered:

$$\mathbf{u}_{ms} = \mathbf{u}_0 + \mathbf{u}_x\, sin\left(a_{\mathbf{u}_x}\pi\frac{x}{L}\right) + \mathbf{u}_y\, sin\left(a_{\mathbf{u}_y}\pi\frac{y}{L}\right) + \mathbf{u}_{xy}\, sin\left(a_{\mathbf{u}_{xy}}\pi\frac{xy}{L^2}\right), \tag{6.6}$$

where $\mathbf{u}_{ms}$ represents the primitive field variables array $\mathbf{u} = [\rho, u, v, p]^T$. The parameters $\mathbf{u}_0$, $\mathbf{u}_x$, $\mathbf{u}_y$, $\mathbf{u}_{xy}$, $a_{\mathbf{u}_x}$, $a_{\mathbf{u}_y}$ and $a_{\mathbf{u}_{xy}}$ are constants specifically chosen for each primitive variable in $\mathbf{u}$. L is the domain characteristic length.

The manufactured expression 6.6 obviously does not satisfy the Navier-Stokes equations. When the field variables are replaced by their corresponding expressions an

extra source term $\boldsymbol{\mathcal{S}}_{ms}$ is generated,

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}}_c(\mathbf{v}) = \nabla \cdot \boldsymbol{\mathcal{F}}_v(\mathbf{v}, \nabla \mathbf{v}) + \boldsymbol{\mathcal{S}}_{ms}. \tag{6.7}$$

The numerical method is forced to search a solution $\mathbf{v}_h$ in order to reduce the right-hand side residue perturbed by the imposed solutions. In some sense, the manufactured expression plays a role very similar to an exact solution. Although the manufactured expressions have no physical consistency, their use can be valuable for validating the capability of the software implementation in dealing with the inconsistencies and to attain the problem convergence, since that expressions exercise all the equation terms (ROY *et al.*, 2007).

6.3.3 Testing Workflow

The problem domain is a straight rectangle, over which a p-refinement array of tests was performed using a 64 uniform quadrilateral and a 128 uniform triangular meshes as can be seen in the Figure 6.5. The analytical expressions of the manufactured solution are used to impose boundary conditions and the maximum values of them used as reference values for the non-dimensionalization of the PDE system (see 3.5 for further explanation).

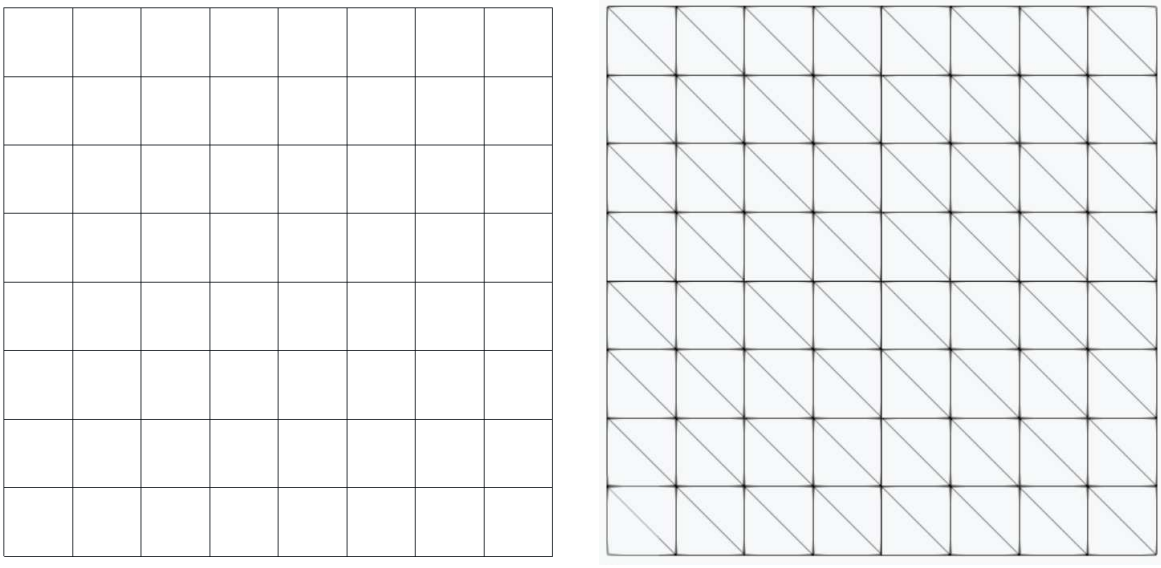

Figure 6.5 – Meshes employed in the diffusive validation tests.

The flux terms on the faces are approximated using the Roe numerical flux for the convective terms and the BR1 for the diffusive ones. The dynamical viscosity μ is constant on the simulation domain and over time. The polynomial order p ranges from 0 to 4. The parameters of 6.6 used in the experiment are listed in Table 6.4 and were based on the reference literature (ROY *et al.*, 2007).

Table 6.4 – Parameters of the manufactured expressions used in the test.

	$\mathbf{u}_0$	$\mathbf{u}_x$	$\mathbf{u}_y$	$\mathbf{u}_{xy}$	$a_{\mathbf{u}_x}$	$a_{\mathbf{u}_y}$	$a_{\mathbf{u}_{xy}}$
$\rho\ (kg/m^3)$	1	0.15	-0.1	0.08	0.75	1	1.25
$u\ (m/s)$	70	7	-8	5.5	1.5	0.5	0.6
$v\ (m/s)$	90	-5	10	-11	1.5	1	0.9
$p\ (kPa)$	1.0	0.2	0.175	-0.25	1	1.25	0.75

Table 6.5 lists the physical constants used in the experiment, even though the manufactured expression has no physical meaning, the choices for the physical parameters are based on the air properties at the normal atmospheric conditions. The solution using the quadrilaterals elements mesh with $p = 4$ is seen in the Figure 6.6.

Table 6.5 – Physical constants used in the manufactured test.

$\mu\ (\mu\, Pa.s)$	17.90
Pr	0.7
γ	1.4

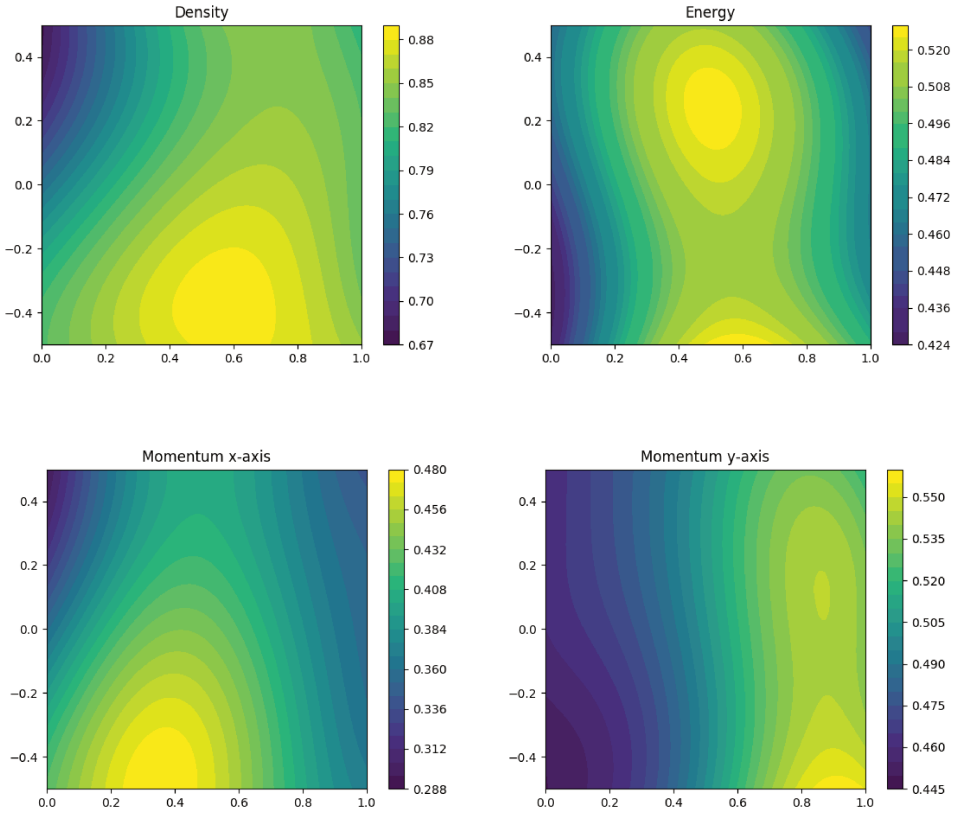

Figure 6.6 – Solution for 64 quadrilateral elements and $p = 4$.

In order to facilitate the attainment of the numerical convergence for the higher orders, an enhancement of the initial condition was performed using the solution of the

immediate lower order for each case starting in $p = 1$,. The order $p = 0$ was initialized using a homogeneous field and the problem of solving the higher orders from scratch was circumvented. The CFL number ranges from 10 to 500. Large values are possible given the regularity of the mesh used in the test and the good behaviour of the manufactured solutions here used. The tests was performed in a machine equipped with a 40-core processor Xeon-E5. The high interpolation order enabled the multithreading parallelism available in Manticore.

6.3.4 Validation

For evaluating the solver accuracy, the L^2-norm

$$E_{L^2} = \sqrt{\int_{\mathbf{\Omega}} (\mathbf{v} - \mathbf{v}_{ms})^2 \, \mathrm{d}\,\mathcal{V}}, \tag{6.8}$$

was used and $\mathbf{v}$ represents each field variable. The way to the error norm should decrease is related to conditions of the problem tested and the discretization type (SZABO; BABUSKA, 2011, pp. 195). The manufactured solution 6.6 is continuous and analytic on the domain simulation $\mathbf{\Omega}$ and its boundaries, which implies that is a Category A problem of the Szabó and Babuška's classification (SZABO; BABUSKA, 2011, pp. 170). For problems in this category it is expected exponential convergence when p-refinement is performed (SZABO; BABUSKA, 2011, pp. 195).

Figures 6.7 and 6.8 show the L^2 error graphics and the respective angular coefficients for each variable are given in Tables 6.6 and 6.7. L^2 error is reduced for all the variables, and the rate of convergence is grow, as expected in the literature.

Table 6.6 – Angular coefficients of the L^2 error curves shown in Figure 6.7 for each p interval.

		ρ	ρu	ρv	ρE
p interval	$0 - 1$	-2.561	-2.223	-2.176	-2.410
	$1 - 2$	-4.215	-3.790	-3.773	-4.157
	$2 - 3$	-5.804	-5.740	-5.435	-5.767
	$3 - 4$	-8.314	-7.671	-7.751	-7.610

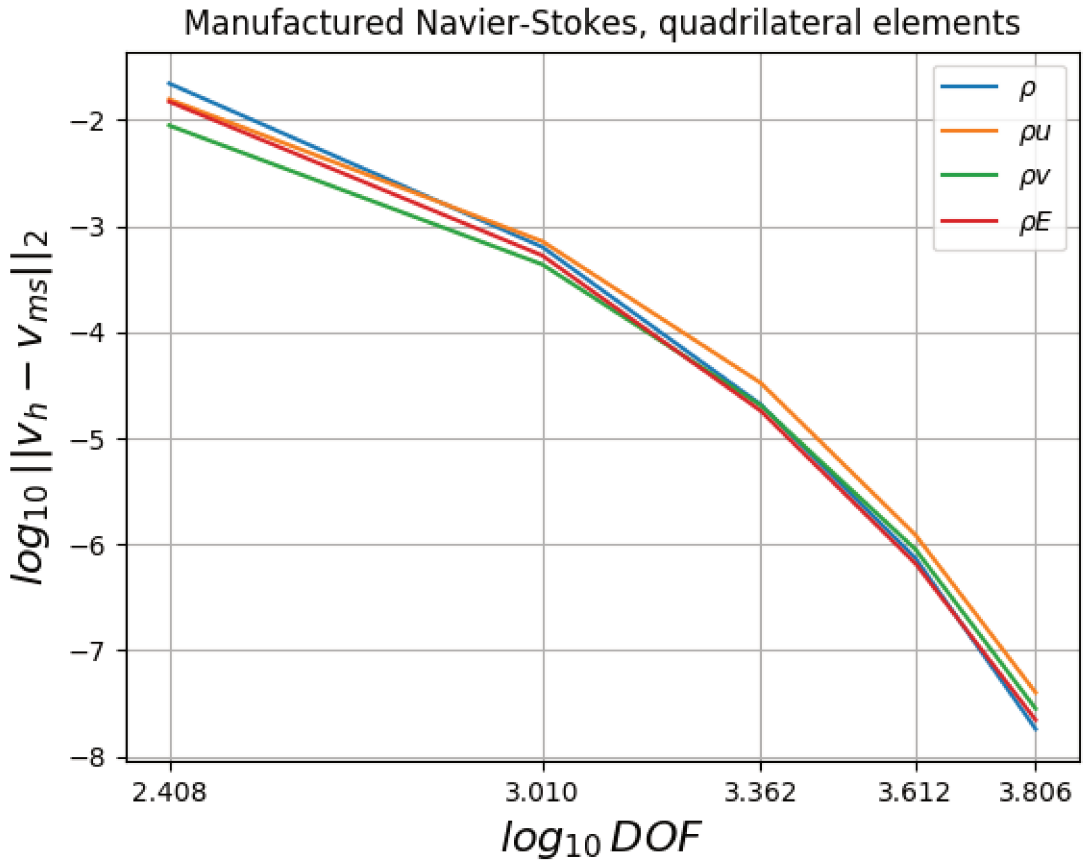

Figure 6.7 – L^2 error curve for the quadrilateral mesh

Table 6.7 – Angular coefficients of the L^2 error curves shown in Figure 6.8 for each p interval.

		ρ	ρu	ρv	ρE
p interval	$0 - 1$	-2.834	-2.595	-2.462	-2.918
	$1 - 2$	-4.255	-4.457	-4.126	-4.376
	$2 - 3$	-5.925	6.243	-6.146	-5.895
	$3 - 4$	-7.455	-7.765	-7.878	-8.020

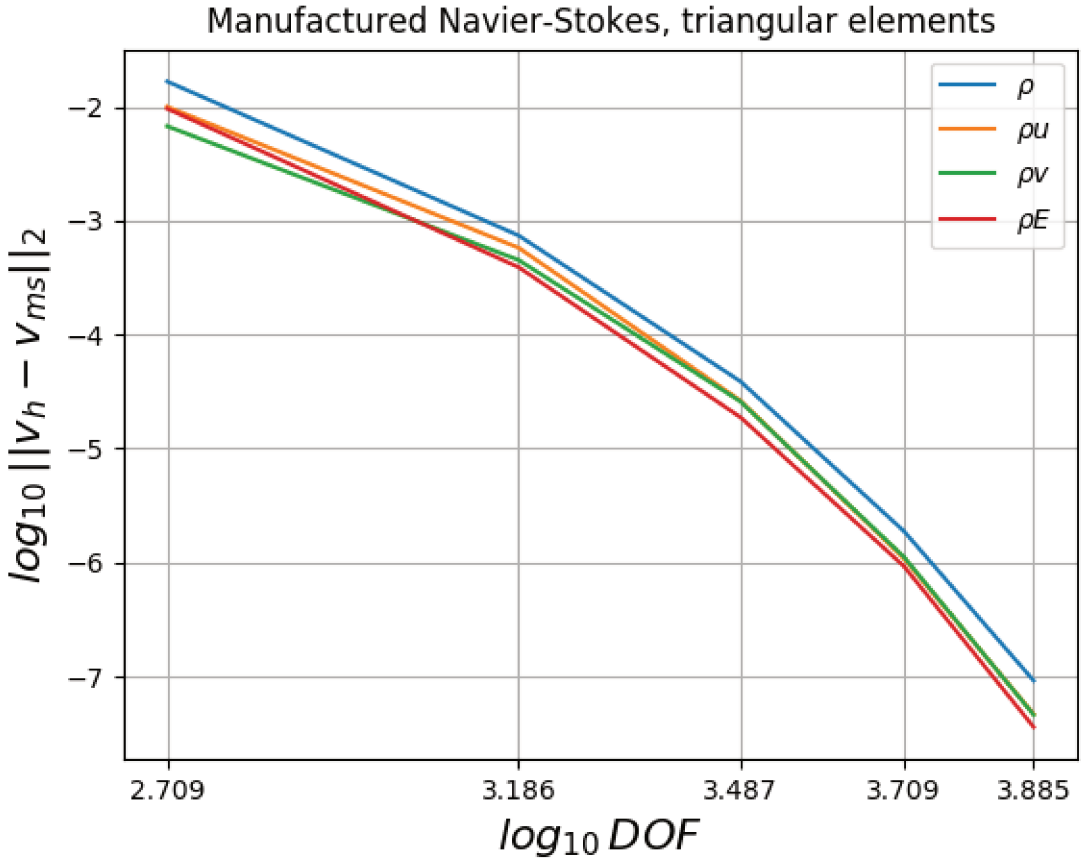

Figure 6.8 – L^2 error curve for the triangular mesh.

6.4 Turbulent Terms Validation

6.4.1 Overview

For the validation of the turbulent terms ($\boldsymbol{\mathcal{F}_t}$ and $\boldsymbol{\mathcal{S}_t}$), a similar scheme to that used for the laminar terms is employed, the use of manufactured expressions. The smooth and well-behaved manufactured expressions presented in 6.6 are here used given that the main purpose of this work is to implement RANS schemes in a high order DG framework for validating them and not to solve realistic turbulent cases. As previously stated, realistic cases in turbulence demand mesh refinement and software infrastructure (such as MPI parallelism, multi-grid, among others) currently not available in the Manticore numerical engine.

6.4.2 Sinusoidal manufactured solution in a regular grid

The manufactured expressions have the form seen in 6.6 and are used as exact solutions for the primitive variables of the array $[\rho\ u\ v\ p\ \tilde{\nu}]^T$. As previously stated, the manufactured expressions 6.6 are not exact solutions for the RANS PDE system. When the field variables are substituted by their corresponding expressions, an extra source term $\boldsymbol{\mathcal{S}}_{ms}$ will be generated

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot \boldsymbol{\mathcal{F}_c}(\mathbf{v}) = \nabla \cdot \boldsymbol{\mathcal{F}_v}(\mathbf{v}, \nabla\mathbf{v}) + \nabla \cdot \boldsymbol{\mathcal{F}_t}(\mathbf{v}, \nabla\mathbf{v}) + \boldsymbol{\mathcal{S}_t}(\mathbf{v}, \nabla\mathbf{v}) + \boldsymbol{\mathcal{S}}_{ms}. \tag{6.9}$$

6.4.3 Testing Workflow

The test performed was to use the same manufactured expressions 6.6 (with an extra equation for the eddy viscosity) and meshes 6.5 employed in the laminar tests to observe the performance of the RANS implementation in conducing the problem to convergence. The analytical expressions were used to impose boundary conditions. The reference values for the non-dimensionalization corresponds to the maximum of the manufactured expressions.

The numerical fluxes used are the HLLC for the convective and the BR1 for the diffusive and turbulent ones. The polynomial order p ranges from 0 to 3, and the CFL number from 0.5 to 30 for all the tests. The dynamical viscosity μ is constant on the simulation domain and over time. The parameters of 6.6 used in the experiment are listed in Table 6.8.

The parameters values used for the four Navier-Stokes variables are the same employed in the laminar tests and are based on the reference literature (ROY *et al.*, 2007). The values for the turbulent variable was chosen to have at most the same magnitude order of the dynamic viscosity, seen in Table 6.5. The results for the triangular mesh with $p = 3$ is seen in Figure 6.9.

Table 6.8 – Parameters of the manufactured expressions used in the test.

	ϕ_0	ϕ_x	ϕ_y	ϕ_{xy}	a_{ϕ_x}	a_{ϕ_y}	$a_{\phi_{xy}}$
$\rho\ (kg/m^3)$	1	0.15	-0.1	0.08	0.75	1	1.25
$u\ (m/s)$	70	7	-8	5.5	1.5	0.5	0.6
$v\ (m/s)$	90	-5	10	-11	1.5	1	0.9
$p\ (kPa)$	1.0	0.2	0.175	-0.25	1	1.25	0.75
$\tilde{\nu}\ (\mu\, Pa.s)$	10.00	2.00	1.75	−2.50	1	1.25	0.75

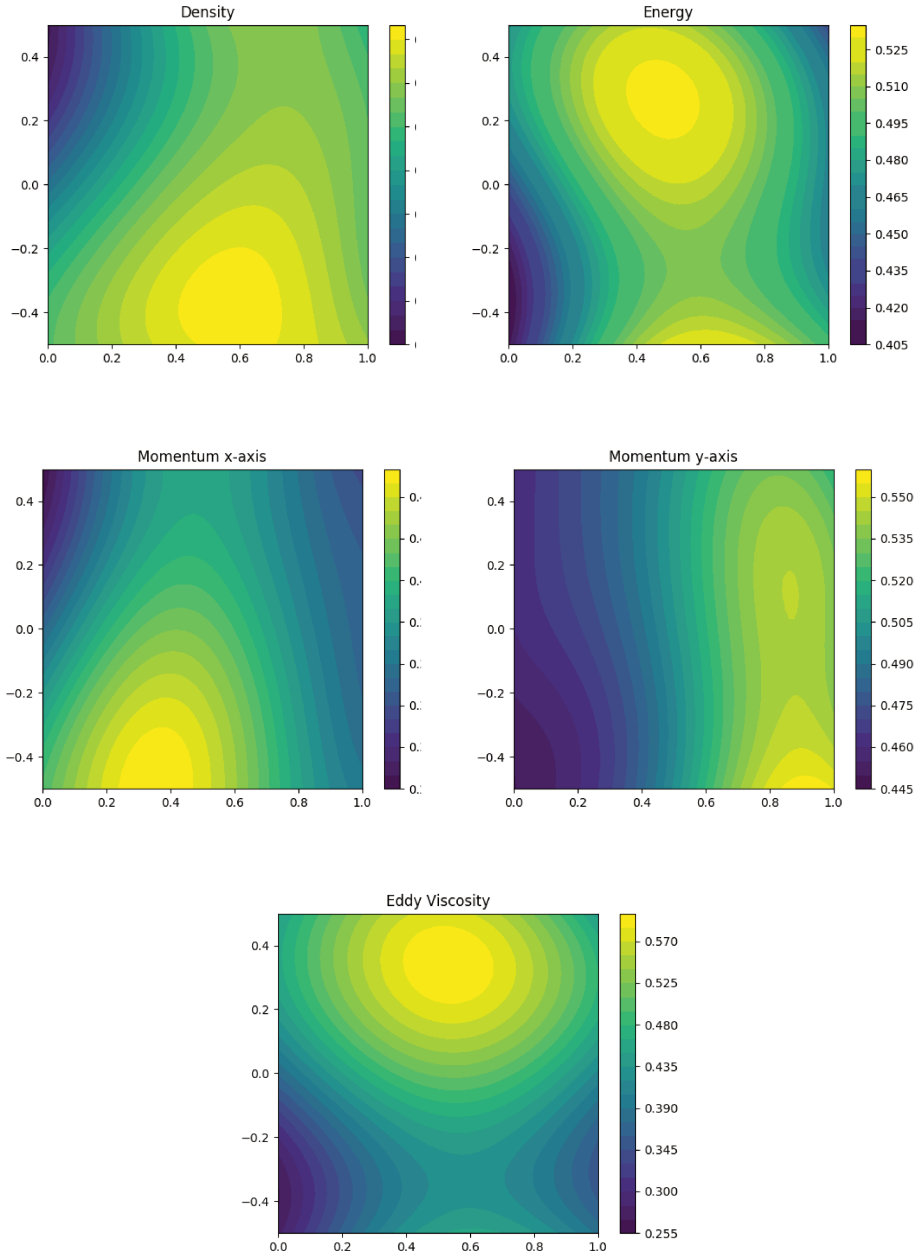

Figure 6.9 – Results for 128 triangular elements and p=3

It was observed that the positivity limiter is highly active during the start-up of the case $p = 0$, which is initialized with constant fields. It is no longer necessary for

the higher orders, probably due to a combination of the manufactured solution regularity and the initialization enhancement.

6.4.4 Validation

Using the L^2-norm given in equation 6.8 the error evaluation was performed and summarized in Figures 6.10 and 6.11. The conditions of the turbulent terms validation problem are similar to that seen in the diffusive validation. Therefore, it is expected that the error norm decreases accordingly to the exponential convergence. Tables 6.9 and 6.10 shows the angular coefficients for each p order interval in the error plots.

Table 6.9 – Angular coefficients of the L^2 error curves shown in Figure 6.10 for each p interval.

		ρ	ρu	ρv	ρE	$\rho\tilde{\nu}$
p interval	$0-1$	-2.578	-2.228	-2.213	-2.314	-1.996
	$1-2$	-4.236	-3.792	-3.785	-4.164	-4.163
	$2-3$	-5.776	-5.738	-5.428	-5.737	-5.706

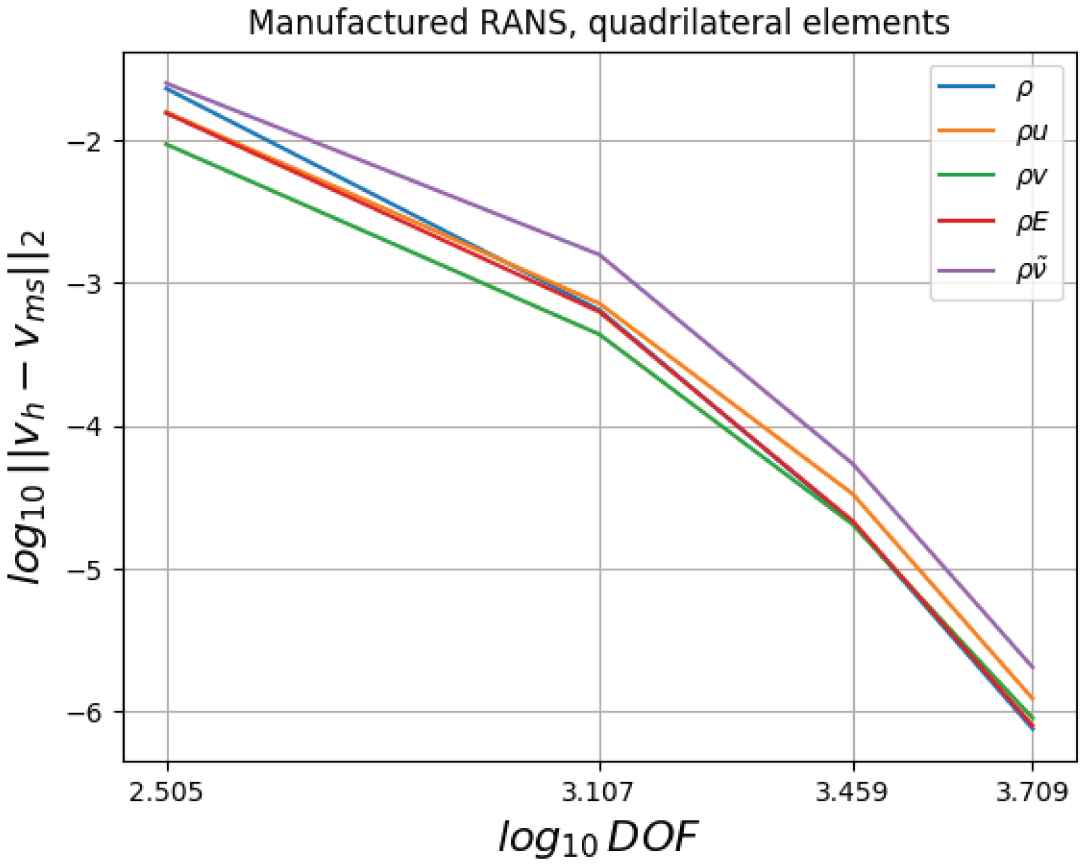

Figure 6.10 – L^2 error curve for the quadrilateral mesh.

The error of the numerical approximation is reduced as the number of degree of freedom is increased and the rates of the error decreases is continuously growing up such

Table 6.10 – Angular coefficients of the L^2 error curves shown in Figure 6.11 for each p interval.

		ρ	ρu	ρv	ρE	$\rho\tilde{\nu}$
p interval	$0 - 1$	-2.864	-2.614	-2.512	-2.807	-2.286
	$1 - 2$	-4.277	-4.464	-4.138	-4.456	-4.802
	$2 - 3$	5.848	-6.206	-6.117	-5.928	-6.101

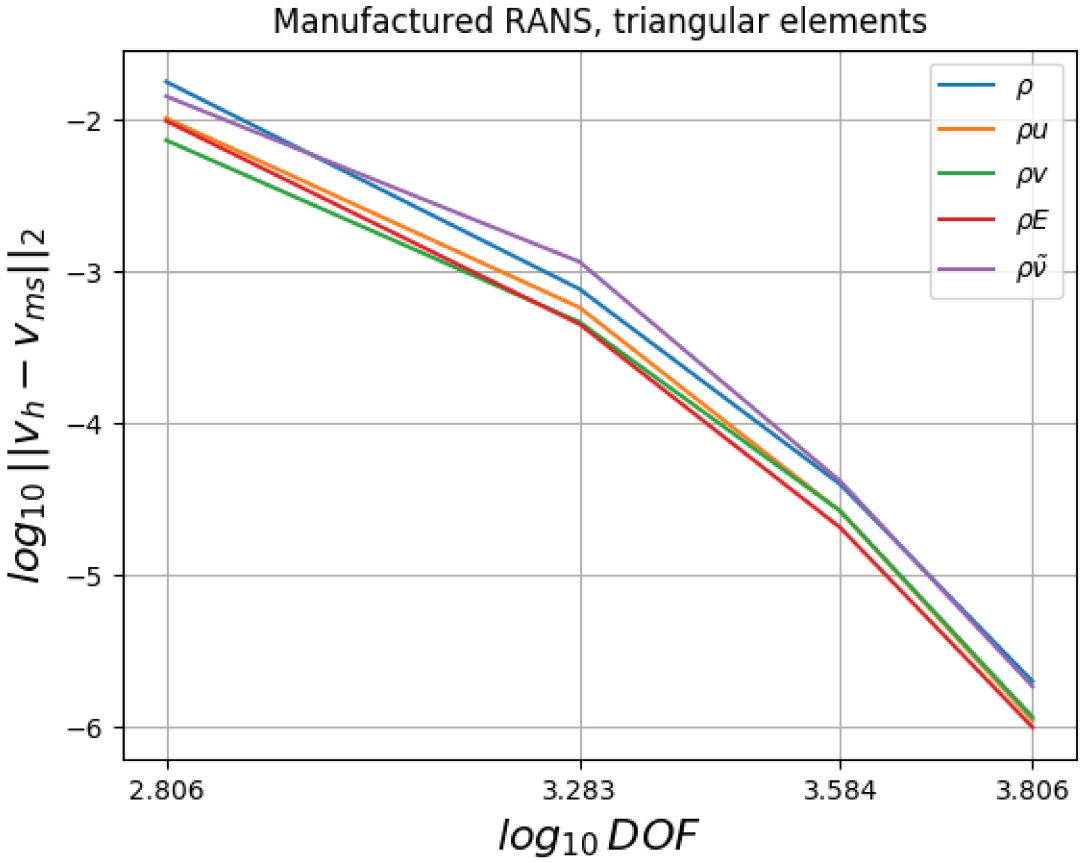

Figure 6.11 – L^2 error curve for the triangular mesh.

as expected. However, given that the size of tests array is relatively small (just four values of p), it is not possible to state if the quality of the convergence is so good as observed in the literature for the higher orders. But it is possible to verify that the implementation is able of solving the problem and ensuring the exponential convergence for relatively high interpolation orders.

7 CONCLUSION

This work was focused on implementing the RANS approach Spalart-Allmaras by extending the compressible flow engine implemented in the framework Manticore, a numerical engine for DG-FEM applications, after which a sequence of validation tests was performed. The tests validated each term of the Navier-Stokes and the RANS PDE systems.

The convective validation using the simplified problem of inviscid vortex demonstrated that the DG numerical approximation implemented in Manticore was able to solve the Euler's equations and obtaining a consistent error reduction. The additional and more challenging problem of the smooth bump confirmed the previous ascertainments when verifying that the inviscid solver achieves the expected solution with a controlled entropy creation, reduced as the refinement (in this case a pure interpolation order refinement) is increased. In addition, the ability of the numerical engine in dealing with high order mesh curvatures also was confirmed.

The diffusive and turbulent terms validation employed smooth and regular manufactured problems for circumventing the computational requirements of the realistic cases. Due to the software infrastructure limitations still presented in the Manticore's engine, the accomplishment of the tests set was extremely difficult, thereby inducing the seek for a less demanding way of demonstrating the convergence of the numerical methods implemented. The error curves for both diffusive and turbulent cases showed that the L^2 error decreases with the interpolation order refinement following an exponential convergence. Such sequence of tests demonstrated that the implemented solvers are able to achieve convergence for certain classes of problems derived from the Navier-Stokes equations, confirming the correctness of the software infrastructure.

We intend to continue with the development based on the Manticore's infrastructure in order to extend the set of test cases supported by the numerical engine. Nonetheless, it is expected that some basic improvements will be necessary before we are able to do it. First of all, the MPI paralellism must be concluded in order to simulate cases with large meshes, since the multithreading paralellism native of the Python libraries is not sufficient for dealing with the considered cases. We also have realized that the implicit time-integration is taking a long time to evaluate the residue derivatives, due to the technique currently implemented evaluates the residue for each field variable serially and calculates the derivatives via finite differences. It can be enhanced by employing multiprocess evaluation or automatic differentiation (which could be executed in GPU). Also it is considered the construction of a full-implicit time-integration scheme, unlike the

approach followed up to the moment, in order to construct a global system that could be more easily solved with a multithreaded CPU or even in a GPU machine.

REFERENCES

ALLMARAS, R. S.; FORRESTER, T. J.; SPALART, P. R. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. **Seventh International Conference on Computational Fluid Dynamics (ICCFD7)**, p. 1–11, 01 2012.

ALLMARAS, S.; JOHNSON, F.; SPALART, P. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. p. 1–11, 01 2012.

BASSI, F. *et al.* Discontinuous galerkin solution of the reynolds-averaged navier–stokes and k-omega turbulence model equations. **Computers & Fluids**, n. 34, p. 507–540, 5 2004.

BASSI, F.; REBAY, S. A high-order accurate discontinuous finite element method for the numerical solution of the compressible navier-stokes equations. **J. Comput. Phys.**, Academic Press Professional, Inc., San Diego, CA, USA, v. 131, n. 2, p. 267–279, mar. 1997. ISSN 0021-9991. Disponível em: <http://dx.doi.org/10.1006/jcph.1996.5572>.

BASSI, F.; REBAY, S. Gmres discontinuous galerkin solution of the compressible navier-stokes equations. **Discontinuous Galerkin Methods: Theory, Computation and Applications**, v. 11, 01 2000.

BIGARELLA, E. D. V. **Advanced Turbulence Modelling for Complex Aerospace Applications**. Tese (Doutorado) — Intituto Tecnológico de Aeronáutica, 2007.

BIRKEN, P. **Numerical Methods for the Unsteady Compressible Navier-Stokes Equations**. Tese (Doutorado) — University of Kassel, 2012. Habilitation thesis.

BIRKEN, P. *et al.* Efficient time-integration for discontinuous galerkin methods for the unsteady 3d navier-stokes equations. **European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS)**, september 2012.

CHAN, J.; RUSSEL, J. H.; WARBURTON, T. Weight-adjusted discontinuous galerkin methods: Curvilinear meshes. 2016. Disponível em: <https://arxiv.org/abs/1608.03836>.

COCKBURN, B.; LIN, S.-Y.; SHU, C.-W. Tvb runge-kutta local projection discontinuous galerkin finite element method for conservation laws iii: One-dimensional systems. **J. Comput. Phys.**, Academic Press Professional, Inc., San Diego, CA, USA, v. 84, n. 1, p. 90–113, set. 1989. ISSN 0021-9991. Disponível em: <http://dx.doi.org/10.1016/0021-9991(89)90183-6>.

COCKBURN, B.; SHU, C.-W. The runge-kutta local projection p^1-discontinuous-galerkin finite element method for scalar conservation laws. **ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique**, Dunod, v. 25, n. 3, p. 337–361, 1991. Disponível em: <http://www.numdam.org/item/M2AN_1991__25_3_337_0>.

COCKBURN, B.; SHU, C.-W. The local discontinuous galerkin method for time-dependent convection-diffusion systems. **SIAM J. Numer. Anal.**, Society for Industrial

and Applied Mathematics, Philadelphia, PA, USA, v. 35, n. 6, p. 2440–2463, out. 1998. ISSN 0036-1429. Disponível em: <https://doi.org/10.1137/S0036142997316712>.

CONTENT, C.; OUTTIER, P.-Y.; CINNELLA, P. Coupled/uncoupled solutions of rans equations using a jacobian-free newton-krylov method. In: **21st Computational Fluid Dynamics Conference, Volume: AIAA-2013-2423**. [S.l.: s.n.], 2013.

GALBRAITH, M. **VI2 - Inviscid Flow through a Channel with a Smooth Gaussian Bump**. 2011. Disponível em: <https://how5.cenaero.be/content/vi2-smooth-gaussian-bump>.

HESTHAVEN, J. S.; WARBURTON, T. **Nodal Discontinuous Galerkin Methods: Algorithms, Analysis and Applications**. [S.l.]: Springer Ed., 2008.

JONES, W. P.; LAUNDER, B. The prediction of laminarization with a two-equation model of turbulence. **International Journal of Heat and Mass Transfer**, v. 15, p. 301–314, 02 1972. Disponível em: <https://www.researchgate.net/publication/223669472_The_prediction_of_laminarization_with_a_two-equation_model_of_turbulence>.

JOSSERAND, C. *et al.* Turbulence: does energy cascade exist? **Journal of Statistics Physics**, p. 596–625, 5 2017.

KÁRMÁN, T. von. Progress in the statistical theory of turbulence. **Proceedings of the National Academy of Sciences of the United States of America**, v. 11, n. 34, p. 530–539, 1938. ISSN 0027-8424.

KARNIADAKIS, E.; SHERWIN, S. **Spectral/hp Element Methods for CFD**. [S.l.]: Oxford University Press, 2004.

KAST, S. C1.1 internal inviscid flow over a smooth bump. 2013.

KIRBY, R. M.; KARNIADAKIS, G. E. Selecting the numerical flux in discontinuous galerkin methods for diffusion problems. 2005.

LANDMANN, B. **A parallel Discontinuous Galerkin Code for the Navier-Stokes and Reynolds-Averaged Navier Stokes Equations**. Tese (Doutorado) — University of Stuttgart, 2008.

LEVEQUE, R. J. **Finite Volume Methods for Hyperbolic Problems**. [S.l.]: Cambridge University Press, 2004.

MCDONOUGH J., M. **Introduction Lectures on Turbulence: Physics, Mathematics and Modeling**. UKnowledge, 2007. Disponível em: <https://uknowledge.uky.edu/me_textbooks/2>.

MENTER, F. R. Two-equation eddy-viscosity turbulence models for engineering applications. **AIAA Journal**, v. 32, p. 1598–1605, august 1994. Disponível em: <http://adsabs.harvard.edu/abs/1994AIAAJ..32.1598M>.

NGUYEN, N. C.; PERSSON, P. O.; PERAIRE, J. Rans solutions using high order discontinuous galerkin methods. In: **45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, FOR DISCONTINUOUS GALERKIN 25**. [S.l.: s.n.], 2007.

OLIVER, T. A. **A High-Order, Adaptive, Discontinuous Galerkin Finite Element Method for the Reynolds-Averaged Navier-Stokes Equations**. Tese (Doutorado) — Massachusetts Institute of Technology, 2008.

PERAIRE, J.; PERSSON, P.-O. The compact discontinuous galerkin (cdg) method for elliptic problems. **SIAM Journal on Scientific Computing**, v. 30, p. 1806–1824, 02 2007.

REED, W.; HILL, T. Triangular mesh methods for the neutron transport equation. 10 1973.

ROY, C. J. *et al.* Verification of rans turbulence models in loci-chem using the method of manufactured solutions. 2007. Disponível em: <http://www.dept.aoe.vt.edu/~cjroy/Conference-Papers/AIAA-2007-4203.pdf>.

SCHMITT F, G. About boussinesq's turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity. **Comptes Rendus Mécanique**, Elsevier, v. 335, p. 617–627, 2007. Disponível em: <https://hal.archives-ouvertes.fr/hal-00264386>.

SPALART, P.; ALLMARAS, S. A one-equation turbulence model for aerodynamic flows. v. 439, 01 1992.

SZABO, B.; BABUSKA, I. **Introduction to Finite Elements Analysis, Formulation, Verification and Validation**. [S.l.]: Wiley, 2011.

TAYLOR, G. I. Statistical theory of turbulence. **Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 151, n. 873, p. 421–444, 1935. ISSN 0080-4630. Disponível em: <http://rspa.royalsocietypublishing.org/content/151/873/421>.

TORO, E. **Riemann Solvers and Numerical Methods for Fluid Dynamics**. [S.l.]: Springer Press, 2008.

Wang, L.; Anderson, W. K.; Erwin, J. T. Inviscid channel flow over a smooth bump via discontinuous galerkin and stabilized petrov-galerkin methods. 2014.

WILCOX, D. C. Reassessment of the scale-determining equation for advanced turbulence models. **AIAA Journal**, v. 26, p. 1299–1310, november 1988. Disponível em: <http://adsabs.harvard.edu/abs/1988AIAAJ..26.1299W>.

YANO, M.; DARMOFAL, D. Case c1.1: Inviscid flow through a channel with a gaussian bump. p. 1–6, 09 2018.

ÇENGEL, Y. A.; CIMBALA, J. **Fluid Mechanics: Fundamentals and Applications**. [S.l.]: McGraw-Hill, 2010.

Appendices

Appendix A – SOFTWARE EXCERPTS

A.1 Elemental Entity Class

```
class ExpansionEntity(CoGeometry):
    """ This class defines a single elemental expansion (a.k.a. finite element).

    """

    __slots__ = [
        '_geotype', '_type', '_subtype', '_ID', '_seqID', '_role', '_key',
        '_fc_n1d', '_vol', 'var', 'cte', 'field', 'cte_field', 'vertex_var',
        'vertex_field', 'vertex_map', 'op', '_gID', 'DR', '_dof', '_eqID'
    ]

    def __init__(self, geotype, volmap, facemap):
        """ Constructor of ExpansionEntity objects.

        Args:
            geotype (geom.StandardGeometry): type of geometric interpolation.

            volmap (CollapsedToGlobalMaps): standard region to global mapping.

            facemap (CollapsedToGlobalFaceMaps): standard region to
                global mapping restricted to faces.

        Attributes:
            _geotype (geom.StandardGeometry): type of geometric interpolation.

            _type (dgtypes.StdRegionType): modal DG standard region

            _subtype (dgtypes.RegionAdapter):

            _ID (np.uint64): Entity id, from mesh generator

            _seqID (np.uint64): Sequential ID, for book keeping

            _role (dgtypes.EntityRole): entity's role (physical,
                ghost, comm_ghost)

            _key (ExpansionKey): expansion key associated to this element.

            _fc_n1d (array): number of integration points required for face
                communication between two elements with different expansions
                max(_key.n1d, ng.n1d) at each face.

            _dof (array): global id's of the degrees of freedom of this element.
                DOF's are sequential at each element: it is only necessary to
                store the first and last+1.

            var (list(FieldVarList)): model/problem names of fields
                interpolated by the DG polynomial expansion. These fields can
                be: state variables, residual variables, auxiliary variables.

            cte (FieldVarList): model/problem names of fields that are constant
                by element.
```

```
53
54              field (list(LocalField)): model/problem values of fields
55                  interpolated by the DG polynomial expansion. These
56                  fields can be: state variables, residual variables,
57                  auxiliary variables.
58
59              cte_field (np.array): model/problem values of fields that are
60                  constant by element.
61
62              vertex_var (FieldVarList): model/problem names of fields that are
63                  interpoled by vertex nodal shape functions.
64
65              vertex_field (LocalField): model/problem values of
66                  fields that are interpoled by vertex nodal shape
67                  functions.
68
69              vertex_map (dict): map from global vertices IDs to local indices
70                  (V->local vertex index).
71
72              DR (np.array): residual derivative.
73
74          """
75
76          assert geotype in std_geometry_as_list()
77
78          self._geotype = geotype
79          self._type = RegionInfo.mesh_to_dg(geom(geotype))
80
81          self._subtype = RegionAdapter.WADG
82          self._ID = np.uint64(0)
83          self._seqID = np.uint64(0)
84          self._role = EntityRole.GHOST
85          self._gID = 0
86          self._key = None
87          self._fc_n1d = array('I', [])
88          self._vol = 0.
89          self._dof = array('I', [0, 0])
90          self._eqID = np.uint64(0)
91
92          self.var = [None for i in range(FieldRole.size())]
93          self.cte = None
94          self.field = [None for i in range(FieldRole.size())]
95          self.cte_field = None
96          self.vertex_var = None
97          self.vertex_field = None
98          self.vertex_map = None
99          self.DR = np.zeros(0)
100
101         #
102         # The StdBackward operator is identical to the PhysBackward
103         # one. No need for an extra function calling overhead.
104         #
105         # self.op[0] = factory_stdbackward1d(DG_Quadrangle)
106         # self.op[1] = factory_stdbackward1d(DG_Triangle)
107         self.op = [None, None]
108
109         #
110         # Parents initialization
111         #
```

```
112         ExpansionNeighbourhood.__init__(self, self._type)
113         CoGeometry.__init__(self, volmap, facemap)
114 
115     def init_variables(self, stv, rsv=[], axv=[]):
116         """ Initialize the fields interpolated by the DG expansion
117         within the element.
118 
119         Args:
120             stv (list(FieldVariable)): State variables
121 
122             rsv (list(FieldVariable)): Residual variables
123 
124             axv (list(FieldVariable)): Auxiliar variables
125 
126         """
127 
128         role = self._role
129         key = self._key
130 
131         if (role == EntityRole.PHYSICAL):
132 
133             S = ExpansionSize.get(self._type, key.order)
134             N = key.n2d
135 
136             assert S > 0
137             assert N > 0
138 
139             stv_size = len(stv)
140             self.var[FieldRole.State] = FieldVarList.get_instance(*stv)
141             self.field[FieldRole.State] = LocalField(stv_size, N, S)
142 
143             rsv_size = len(rsv)
144             if len(rsv) > 0:
145                 self.var[FieldRole.Residual] = FieldVarList.get_instance(*rsv)
146                 self.field[FieldRole.Residual] = LocalField(rsv_size, 0, S)
147 
148             axv_size = len(axv)
149             if len(axv) > 0:
150                 self.var[FieldRole.Auxiliar1] = FieldVarList.get_instance(*axv)
151                 self.field[FieldRole.Auxiliar1] = LocalField(axv_size, 0, S)
152                 self.var[FieldRole.Auxiliar2] = FieldVarList.get_instance(*axv)
153                 self.field[FieldRole.Auxiliar2] = LocalField(axv_size, 0, S)
154 
155             sz = rsv_size * S
156             self.DR.resize( (sz, sz), refcheck=False )
157 
158         elif (role == EntityRole.COMM_GHOST):
159             S = ExpansionSize.get(self._type, key.order)
160             N = key.n2d
161 
162             assert S > 0
163             assert N > 0
164 
165             stv_size = len(stv)
166             self.var[FieldRole.State] = FieldVarList.get_instance(*stv)
167             self.field[FieldRole.State] = LocalField(stv_size, N, S)
168 
169         elif (role == EntityRole.GHOST):
170             # We need the expansion from the neighbour (physical elment)
```

```
171
172             kn = self.get_neighbour(0).key
173             N = kn.n1d + 1  # <-- Note: the correct size is n1d+1! K&S, p. 558
174
175             stv_size = len(stv)
176             self.var[FieldRole.State] = FieldVarList.get_instance(*stv)
177             self.field[FieldRole.State] = LocalField(stv_size, N)
178
179         else:
180             raise AssertionError("Malformed Expansion entity!")
181
182     def init_ctes(self, ctv):
183         """Initialize the constant fields within the element.
184
185         Args:
186
187             ctv (list(FieldVariable)): constant fields.
188
189         """
190         if len(ctv) > 0:
191             self.cte = FieldVarList.get_instance(*ctv)
192             self.cte_field = np.zeros(len(ctv))
193
194     def init_vertex_variables(self, vtv):
195         """ Initialize fields that are interpoled by vertex nodal
196         shape functions.
197
198         Args:
199
200             vtv (list(FieldVariable)): vertex fields.
201
202         """
203         if len(vtv) > 0:
204             self.vertex_var = FieldVarList.get_instance(*vtv)
205             self.vertex_field = LocalField(
206                 len(vtv), self._key.n2d, number_of_edges(self._geotype))
207
208     def init_vertex_map(self, vertices_list):
209         """ Initialize the map from global vertices IDs to local
210         indices.
211
212         Args:
213             vertices_list (iterable(np.uint64))
214
215         """
216
217         assert len(vertices_list) == number_of_edges(self._geotype)
218         local_idx = 0
219         for vertex in vertices_list:
220             self.vertex_map[vertex] = local_idx
221             local_idx += 1
222
223     @property
224     def type(self):
225         return self._type
226
227     @property
228     def subtype(self):
229         return self._subtype
```

```
230
231     @subtype.setter
232     def subtype(self, value):
233         assert value in RegionAdapter
234         self._subtype = value
235
236     @property
237     def geom_type(self):
238         return self._geotype
239
240     @geom_type.setter
241     def geom_type(self, value):
242         assert value in std_geometry_as_list()
243         self._geotype = value
244         self._type = base_to_std_region[geom(value)]
245
246     @property
247     def ID(self):
248         return self._ID
249
250     @ID.setter
251     def ID(self, value):
252         self._ID = np.uint64(value)
253
254     @property
255     def seqID(self):
256         return self._seqID
257
258     @seqID.setter
259     def seqID(self, value):
260         self._seqID = np.uint64(value)
261
262     @property
263     def gID(self):
264         return self._gID
265
266     @gID.setter
267     def gID(self, value):
268         self._gID = int(value)
269
270     @property
271     def role(self):
272         return self._role
273
274     @role.setter
275     def role(self, value):
276         assert value in EntityRole
277         self._role = value
278
279     @property
280     def key(self):
281         return self._key
282
283     @key.setter
284     def key(self, k):
285         assert type(k) is ExpansionKey
286         self._key = k
287         self.op[0] = class_quad_stdbackward1d(k)
288         self.op[1] = class_tria_stdbackward1d(k)
```

```
289
290     @property
291     def n1d(self):
292         return self._key.n1d
293
294     @property
295     def volume(self):
296         return self._vol
297
298     @volume.setter
299     def volume(self, value):
300         assert value > 0.
301         self._vol = value
302
303     @property
304     def dof(self):
305         return self._dof
306
307     @property
308     def eqID(self):
309         return self._eqID
310
311     def init_face_comm_sizes(self):
312         """Number of integration points on each face.
313
314         Notes:
315
316         * According to K&S p.558, the number of integration points
317         on each face must be n1d+1 for obtaining the correct mesh
318         convergence.
319         """
320         n = self.n1d
321
322         if ((self.role == EntityRole.PHYSICAL)
323                 or (self.role == EntityRole.COMM_GHOST)):
324             for ng in self.neigh:
325                 self._fc_n1d.append(max(n + 1, ng.n1d + 1))
326
327     def face_comm_size(self, ff):
328         return self._fc_n1d[ff]
329
330     @property
331     def face_comm_sizes(self):
332         return self._fc_n1d
333
334     def set_vertices(self, V):
335         CoGeometry.set_vertices(self, V)
336
337         nfaces = number_of_edges(self._geotype)
338
339         if nfaces == 3:
340             self.set_face_vertices(0, V[0], V[1])
341             self.set_face_vertices(2, V[1], V[2])
342             self.set_face_vertices(1, V[2], V[0])
343         elif nfaces == 4:
344             self.set_face_vertices(0, V[0], V[1])
345             self.set_face_vertices(3, V[1], V[2])
346             self.set_face_vertices(1, V[2], V[3])
347             self.set_face_vertices(2, V[3], V[0])
```

```
348             else:
349                 raise AssertionError("Incorrect number of faces!")
350 
351         def eval_jacobian(self):
352             CoGeometry.eval_jacobian(self, self._geotype, self._key, self._fc_n1d)
353 
354         def eval_mass(self):
355             CoGeometry.eval_mass(self, self._type, self._key)
356 
357         def eval_face_char_length(self):
358             pass
359 
360         def normals_tangents(self, face_id, n, t):
361             # The correct number of integration points on faces is n1d+1,
362             # see K&S p.558.
363             k = InterpolationKey.get_instance(self._geotype, self._key.n1d + 1,
364                                               face_id)
365             self._fmap.normals_tangents(k, self.C, n, t)
366 
367         def get_field(self, role, field_type, v):
368             """ Get field. """
369             return self.field[role].get(field_type, self.var[role](v))
370 
371         def state(self, field_type, v):
372             role = FieldRole.State
373             return self.field[role].get(field_type, self.var[role](v))
374 
375         def residual(self, field_type, v):
376             role = FieldRole.Residual
377             return self.field[role].get(field_type, self.var[role](v))
378 
379         def auxiliar1(self, field_type, v):
380             role = FieldRole.Auxiliar1
381             return self.field[role].get(field_type, self.var[role](v))
382 
383         def auxiliar2(self, field_type, v):
384             role = FieldRole.Auxiliar2
385             return self.field[role].get(field_type, self.var[role](v))
386 
387         def copy_field(self, e):
388 
389             if ((self._role == EntityRole.PHYSICAL)
390                     or (self._role == EntityRole.COMM_GHOST)):
391 
392                 for role in FieldRole:
393                     self.field[role].copy(FieldType.ph, e.field[role])
394                     self.field[role].copy(FieldType.tr, e.field[role])
395 
396         def get_face_field(self, field_role, v, face_id, face_values):
397 
398             if ((self._role == EntityRole.PHYSICAL)
399                     or (self._role == EntityRole.COMM_GHOST)):
400                 # Returns values at i.p.s on the face:
401                 self.op[self._type].at_ips(face_id,
402                                            self.get_field(field_role, FieldType.tr,
403                                                           v), face_values)
404 
405             elif (self._role == EntityRole.GHOST):
406                 # Values are already stored at i.p.s on the face but, in
```

```
407             # this case, it only makes sense field_role==FieldRole.State
408             # and field_type==FieldType.ph).
409
410             assert field_role == FieldRole.State
411
412             np.copyto(face_values, self.get_field(field_role, FieldType.ph, v))
413
414         else:
415             raise AssertionError("Malformed Expansion entity!")
416
417     def get_face_field_as_passive(self, field_role, v, actV, actF, actNIP,
418                                   pasF, face_values):
419
420         assert ((self._role == EntityRole.PHYSICAL)
421                 or (self._role == EntityRole.COMM_GHOST))
422
423         face_values = self.op[self._type].at_ips_as_passive(
424             actV, actF, actNIP, pasF,
425             self.get_field(field_role, FieldType.tr, v), face_values)
426
427     def get_cte(self, v):
428         return self.cte_field[self.cte(v)]
429
430     def set_cte(self, v, value):
431         self.cte_field[self.cte(v)] = value
432
433     def get_vertex_field(self, field_type, v):
434         return self.vertex_field.get(field_type, self.vertex_var(v))
435
436     def get_vertex_value(self, v, node_global_id):
437         assert node_global_id in self.vertex_map
438         return self.vertex_field.get(
439             FieldType.tr, self.vertex_var(v))[self.vertex_map[node_global_id]]
440
441     def set_vertex_value(self, v, node_global_id, value):
442         assert node_global_id in self.vertex_map
443         self.vertex_field.get(
444             FieldType.tr,
445             self.vertex_var(v))[self.vertex_map[node_global_id]] = value
446
447     def __repr__(self):
448
449         msg = '<{} (ID: {}, seqID: {}, geo: {}, subtype: {}, role: {})'.format(
450             self.__class__, self.ID, self.seqID,
451             geometry_name(self.geom_type), self.subtype, self.role)
452         msg += '\n{}'.format(self.key.key)
453         msg += '\nVertices: {}'.format(self.get_vertices())
454         msg += '\nNodal coordinates:\n{}'.format(self.get_coeff())
455         msg += '\nNeighborhood([ '
456
457         neigh = self.get_neighbourhood()
458         for ng in neigh:
459             if ng is not None:
460                 msg += '{} '.format(ng.ID)
461
462         msg += '])'
463
464         msg += '\n>'
465
```

```
466         return msg
```

A.2 Weak form of the convective residual

```
1  class weak_dg_convective_residual(
2              EntityOperator,
3              model_description_mixin,
4              equation_domain_signature,
5              expansion_numbers):
6
7          def __init__(self, model_desc):
8              model_description_mixin.__init__(self, model_desc)
9              equation_domain_signature.__init__(self)
10             expansion_numbers.__init__(self)
11             EntityOperator.__init__(self)
12
13             self.innerpg  = [None, None] # InnerProductGradient[quad, tria]
14             self.innerp_t = [None, None] # InnerProduct1d[quad, tria]
15             self.Fx       = np.zeros(0)
16             self.Fy       = np.zeros(0)
17             self.inner_temp = [np.zeros(0), np.zeros(0)]
18
19             self.temp = np.zeros(0)
20             self.aux  = np.zeros(0)
21
22             if exec_dir==ExecDirective.NS_VALIDATION:
23                 self.innerpval = [None, None] # InnerProduct2d[quad, tria]
24                 self.evalSrc   = [None, None]
25
26             self.eq_dir = model_desc.get_equations()[0].data.name
27
28             #RANS turbulence
29             #Instantion of the turbulence model
30             self.turb_model = turb_model_t(model_desc)
31
32             if self.eq_dir=='turbulent':
33                 self.innerpval = [None, None] # InnerProduct2d[quad, tria]
34                 self.evalSrc   = [None, None]
35
36         def setup(self, eqname, domain):
37             equation_domain_signature.setup(self, eqname, domain)
38             matname = self.desc.get_equation(eqname).mat
39             self.fluid = self.desc.get_material(matname)
40
41             if exec_dir==ExecDirective.NS_VALIDATION:
42                 g       = self.fluid.thermo.gamma
43                 cp      = self.fluid.thermo.Cp
44                 cv      = self.fluid.thermo.Cv
45                 mu      = self.fluid.transport.mu
46                 Pr      = self.fluid.transport.Pr
47                 k       = mu*cp/Pr
48
49                 # Get analytical expression
50                 self.fval = ns_validation(g, mu, k, cv)
51
52         def container_operator(self, group, key, container):
53             k = ExpansionKeyFactory.make(key.order, key.nip)
54
55             regions =  [StdRegionType.DG_Quadrangle, StdRegionType.DG_Triangle]
```

```
56              for r in regions:
57                  expansion_numbers.eval(self, r, k) # set nS and nQ=k.n2d
58                  self.inner_temp[r].resize( (self.nS,), refcheck=False)
59
60              self.innerpg[regions[0]] = class_quad_physinnerproductgrad2d(k)
61              self.innerpg[regions[1]] = class_tria_physinnerproductgrad2d(k)
62
63              self.innerp_t[regions[0]] = class_quad_physinnerproduct1d
64              self.innerp_t[regions[1]] = class_tria_physinnerproduct1d
65
66              self.Fx.resize( (self.nQ,), refcheck=False)
67              self.Fy.resize( (self.nQ,), refcheck=False)
68
69              self.temp.resize( (key.nip+1,), refcheck=False)
70              self.aux.resize( (key.nip+1,), refcheck=False)
71
72              if exec_dir==ExecDirective.NS_VALIDATION or self.eq_dir=='turbulent':
73
74                  self.innerpval[regions[0]] = class_quad_physinnerproduct2d(k)
75                  self.innerpval[regions[1]] = class_tria_physinnerproduct2d(k)
76
77                  self.evalSrc[regions[0]] = class_quad_physevaluator2d(k)
78                  self.evalSrc[regions[1]] = class_tria_physevaluator2d(k)
79
80
81          def __call__(self, e):
82              """Evaluates the convective residual of a single element.
83
84              The usual __call__() is provided to be used as a standalone
85              residual evaluator (it will multiply the residuals by the
86              inverse of the mass matrix). If you need to use it as a part of
87              a bigger residual evaluation (Navier-Stokes, for instance), you
88              should resort to eval().
89              """
90
91              # Check setup was done
92              assert self.domain
93
94              # Set nS and nQ for this element
95              expansion_numbers.eval(self, e.type, e.key)
96
97              self.eval(e)
98
99              Residual_p = ResidualWsp_p.get_instance(self.nS)
100
101             # M^{-1} product
102             IM = e.get_inv_mass()
103             tr = FieldType.tr
104
105             InverseMassOperator[e.subtype].solve(
106                 IM, Residual_p.rho,e.residual(tr,FieldVariable.RHO) )
107             InverseMassOperator[e.subtype].solve(
108                 IM, Residual_p.rhou, e.residual(tr,FieldVariable.RHOU) )
109             InverseMassOperator[e.subtype].solve(
110                 IM, Residual_p.rhov, e.residual(tr,FieldVariable.RHOV) )
111             InverseMassOperator[e.subtype].solve(
112                 IM, Residual_p.rhoe, e.residual(tr,FieldVariable.RHOE) )
113
114             for name, var in self.turb_model.fieldvariables:
```

```
115                 var_res = getattr(Residual_p,name)
116                 InverseMassOperator[e.subtype].solve(
117                     IM, var_res, e.residual(tr,var) )
118
119         def eval(self, e):
120             """Evaluates the residual without the inverse mass matrix.
121
122             Note:
123
124                 * This residual evaluation 'kernel' is useful for
125                     combining with more complex residuals (e.g. artificial
126                     dissipation, viscous flow, etc.)
127             """
128             # Filling the primitive variables workspace
129             Primitive_v = PrimitiveWsp_v.get_instance(self.nQ)
130             Primitive_v.fill(e, self.fluid)
131
132             # Zeroing residuals
133             Residual_p = ResidualWsp_p.get_instance(self.nS)
134             Residual_p.zero()
135
136             # Volumetric operations
137             self.volumeContribution(e)
138
139             # Facial operations
140             self.faceContribution(e)
```

A.3 Turbulent Residual

```
1  def factory_weak_dg_turbulent_res(conv_flux_t, visc_flux_t, turb_model_t):
2
3      weak_dg_viscous_res = factory_weak_dg_viscous_res(conv_flux_t,
4      visc_flux_t, turb_model_t)
5
6      if turb_model_t == SpallardAlmaras:
7
8          TurbModel_v = RANS_SA_Wsp_v
9          TurbModel_f = RANS_SA_Wsp_f
10         weak_dg_viscous_res = factory_weak_dg_viscous_res(conv_flux_t,
11         visc_flux_t, turb_model_t)
12
13         class weak_dg_turbulent_res(weak_dg_viscous_res):
14
15             def __init__(self, model_desc):
16
17                 weak_dg_viscous_res.__init__(self, model_desc)
18                 self.Re = model_desc.get_reference().Re # Reynolds
19                 self.rans_settings = model_desc.get_rans_settings()
20                 self.feature = self.rans_settings.feature
21                 self.cv1 = self.rans_settings.cv1
22                 self.Pr_e = self.rans_settings.Pr_e
23                 self.sigma =  self.rans_settings.sigma
24                 self.D = self.rans_settings.D
25
26             def setup(self, eqname, domain):
27
28                 weak_dg_viscous_res.setup(self, eqname, domain)
29                 matname = self.desc.get_equation(eqname).mat
30                 self.fluid = self.desc.get_material(matname)
```

```
31                  self.cp = self.fluid.thermo.Cp
32                  self.gamma = self.fluid.thermo.gamma
33
34                  #It constructs the source terms evaluation for SA model
35                  according to the
36                  #consideration of transition state or not
37                  turb_source =  factory_turbulent_source(self.feature)
38                  self.turb_sourceContribution = turb_source(self.desc, self.fluid)
39                  self.turb_model.cp = self.cp
40
41              def eval(self, e):
42                  """Evaluates the residual without the inverse mass matrix.
43
44                  """
45                  super().eval(e)
46
47                  # Volumetric terms operations
48                  self.turb_volumeContribution(e)
49
50                  # Facial terms operations has been evaluated togheter with the
51                  # laminar viscous flux because the regarded methods are defined
52                  # on TurbulenceModels.py
53
54              #Volumetric operations for the turbulence models
55              def turb_volumeContribution(self, e):
56
57                  Tau_v       = TauWsp_v.get_instance(self.nQ)
58                  p_wsp       = PrimitiveWsp_v.get_instance(self.nQ)
59                  Residual_p = ResidualWsp_p.get_instance(self.nS)
60                  TurbModelWsp_v = TurbModel_v.get_instance(self.nS, self.nQ)
61                  RANS_SAPrimWsp_v = RANS_SAPrimitiveWsp_v.get_instance(self.nQ)
62
63                  mu   = e.state(FieldType.ph, FieldVariable.MU)
64                  dudx = e.state(FieldType.ph, FieldVariable.DUDX)
65                  dudy = e.state(FieldType.ph, FieldVariable.DUDY)
66                  dvdx = e.state(FieldType.ph, FieldVariable.DVDX)
67                  dvdy = e.state(FieldType.ph, FieldVariable.DVDY)
68
69                  #Viscous Newtonian tensor evalaution
70                  Tau11( mu, dudx, dvdy, Tau_v.t11 )
71                  Tau12( mu, dudy, dvdx, Tau_v.t12 )
72                  Tau22( mu, dudx, dvdy, Tau_v.t22 )
73
74                  J  = e.get_jacobian()
75                  IM = e.get_inv_jacobian_matrix()
76                  inner_temp = self.inner_temp[e.type]
77                  innerpg    = self.innerpg[e.type]
78                  innerpval  = self.innerpval[e.type]
79                  Reynolds   = self.Re
80
81                  #
82                  # FIRST EQUATION: Zero.
83                  #
84
85                  ### Viscosity ratio: mu_e/mu
86                  mu   = e.state(FieldType.ph, FieldVariable.MU)
87                  rhonu = e.state(FieldType.ph, FieldVariable.RHONU)
88
89                  #Cho = rhonu/mu
```

```
90                  np.divide(rhonu, mu, out=TurbModelWsp_v.Cho)
91                  #Cho^3
92                  np.power(TurbModelWsp_v.Cho, 3, out=TurbModelWsp_v.flux_aux1)
93                  #Cho^3+cv1^3
94                  np.add(TurbModelWsp_v.flux_aux1,self.cv1**3, out=TurbModelWsp_v.
                        flux_aux2)
95                  #fv1 = Cho^3/(Cho^3+cv1^3)
96                  np.divide(TurbModelWsp_v.flux_aux1, TurbModelWsp_v.flux_aux2,
97                  out=TurbModelWsp_v.fv1)
98
99                  #mu_e = rho*nu*fv1
100                 np.multiply(rhonu, TurbModelWsp_v.fv1, out=TurbModelWsp_v.mu_e)
101
102                 #
103                 # SECOND EQUATION:
104                 #
105                 np.multiply(TurbModelWsp_v.mu_e, Tau_v.t11, out=TurbModelWsp_v.
                        flux_aux1)
106                 np.multiply(TurbModelWsp_v.mu_e, Tau_v.t12, out=TurbModelWsp_v.
                        flux_aux2)
107
108                 innerpg(J, IM, TurbModelWsp_v.flux_aux1, TurbModelWsp_v.flux_aux2,
                        inner_temp)
109                 inner_temp /= Reynolds
110                 Residual_p.rhou += inner_temp
111
112                 #
113                 # THIRD EQUATION:
114                 #
115                 np.multiply(TurbModelWsp_v.mu_e, Tau_v.t22, out=TurbModelWsp_v.
                        flux_aux3)
116
117                 innerpg(J, IM, TurbModelWsp_v.flux_aux2, TurbModelWsp_v.flux_aux3,
                        inner_temp)
118                 inner_temp /= Reynolds
119                 Residual_p.rhov += inner_temp
120
121                 ###Turbulent thermal condutivity, k_e
122                 np.multiply((self.gamma/self.Pr_e), TurbModelWsp_v.mu_e, out=
                        TurbModelWsp_v.k_e)
123
124                 #
125                 # FOURTH EQUATION:
126                 #
127                 np.multiply(p_wsp.u, TurbModelWsp_v.flux_aux1, out=TurbModelWsp_v.
                        flux_tmp1)
128                 np.multiply(p_wsp.v, TurbModelWsp_v.flux_aux2, out=TurbModelWsp_v.
                        flux_tmp2)
129                 np.multiply(p_wsp.u, TurbModelWsp_v.flux_aux2, out=TurbModelWsp_v.
                        flux_tmp3)
130                 np.multiply(p_wsp.v, TurbModelWsp_v.flux_aux3, out=TurbModelWsp_v.
                        flux_tmp4)
131
132                 np.add(TurbModelWsp_v.flux_tmp1, TurbModelWsp_v.flux_tmp2, out=self.
                        Fx)
133                 np.add(TurbModelWsp_v.flux_tmp2, TurbModelWsp_v.flux_tmp3, out=self.
                        Fy)
134
135                 dedx = e.state(FieldType.ph, FieldVariable.DEIDX)
```

```
136                     dedy = e.state(FieldType.ph, FieldVariable.DEIDY)
137
138                     self.Fx += np.multiply(TurbModelWsp_v.k_e, dedx)
139                     self.Fy += np.multiply(TurbModelWsp_v.k_e, dedy)
140
141                     innerpg(J, IM, self.Fx, self.Fy, inner_temp)
142                     inner_temp *= -1./Reynolds
143                     Residual_p.rhoe += inner_temp
144
145                     #
146                     # FIFTH EQUATION:
147                     #
148                     mu = e.state(FieldType.ph, FieldVariable.MU)
149                     dnudx = e.state(FieldType.ph, FieldVariable.DNUIDX)
150                     dnudy = e.state(FieldType.ph, FieldVariable.DNUIDY)
151
152                     TurbulentFlux_5_X(mu, rhonu, (1./self.sigma)*dnudx, TurbModelWsp_v.
                            flux_aux1, self.Fx )
153
154                     TurbulentFlux_5_Y(mu, rhonu, (1./self.sigma)*dnudy, TurbModelWsp_v.
                            flux_aux1, self.Fy )
155                     innerpg(J, IM, self.Fx, self.Fy, inner_temp)
156
157                     inner_temp /= Reynolds
158                     Residual_p.rhonu += inner_temp
159
160                     #
161                     #SOURCE TERMS FOR THE TURBULENCE VARIABLES
162                     #
163                     self.turb_sourceContribution(e, TurbModelWsp_v,  RANS_SAPrimWsp_v,
                            PrimitiveWsp_v=p_wsp)
164                     innerpval(TurbModelWsp_v.rhonu_res, J, inner_temp)
165
166                     Residual_p.rhonu += inner_temp
167
168                     D = self.D.get(e.ID)
169
170                     #In case of the RANS validation
171                     if exec_dir==ExecDirective.RANS_VALIDATION:
172
173                         C    = e.get_coeff()
174                         geo  = e.geom_type
175                         vmap = e.vmap
176                         f    = self.rans_settings.f
177                         f.D  = D
178
179                         inner_temp = self.inner_temp[e.type]
180                         evalSrc    = self.evalSrc[e.type]
181                         innerpval  = self.innerpval[e.type]
182
183                         src = evalSrc(f.src_rho, C, geo, vmap)
184                         innerpval(src, J, inner_temp)
185                         Residual_p.rho += inner_temp
186
187                         src = evalSrc(f.src_rhou, C, geo, vmap)
188                         innerpval(src, J, inner_temp)
189                         Residual_p.rhou += inner_temp
190
191                         src = evalSrc(f.src_rhov, C, geo, vmap)
```

```
192                     innerpval(src, J, inner_temp)
193                     Residual_p.rhov += inner_temp
194
195                     src = evalSrc(f.src_rhoe, C, geo, vmap)
196                     innerpval(src, J, inner_temp)
197                     Residual_p.rhoe += inner_temp
198
199                     src = evalSrc(f.src_rhonu, C, geo, vmap)
200                     innerpval(src, J, inner_temp)
201                     Residual_p.rhonu += inner_temp
202
203
204     else:
205         raise RuntimeError("Model not implemented!")
206
207     return  weak_dg_turbulent_res
```

A.4 Positivity Limiter

```
1  #Hard limiting based on Landmann (2008).
2  #This approach is specific for the Spalart-Allmaras turbulence modelling.
3
4  #Positivity limiting for quadrilaterals
5  class hard_limit_quad:
6
7      def __init__(self):
8          pass
9
10     def __call__(self, rhonu_coeff):
11
12         #Truncate the high order modes
13         #Apply the linearization
14         # s = -u0/min(-u1+u2, -u1-u2, u1-u2, u1+u2)
15         u0 = rhonu_coeff[0]
16         u1 = rhonu_coeff[1]
17         u2 = rhonu_coeff[2]
18
19         rhonu_coeff[1:].fill(0)
20
21         aux1 = np.array([-u1+u2, -u1-u2, u1-u2, u1+u2])
22         aux2 = aux1.min()
23
24         s = np.minimum(1,-u0/aux2)
25
26         rhonu_coeff[1] = s*u1
27         rhonu_coeff[2] = s*u2
28
29 #Positivity limiting for triangles
30 class hard_limit_triang:
31
32     def __init__(self):
33         pass
34
35     def __call__(self, rhonu_ceoeff):
36
37         u0 = rhonu_coeff[0]
38         u1 = rhonu_coeff[1]
39         u2 = rhonu_coeff[2]
40
```

```
41          #u2>=u0 && u2<=-u0/2
42          u2 = np.minimum(u0, u2)
43          u2 = np.maximum(-u0/2, u2)
44
45          #u1<=u2-u0 && u1>=u0-u2
46          u1 = np.maximum(u0-u2)
47          u1 = np.minimum(u2-u0)
48
49
50  class hard_limiting:
51
52      def __init__(self, model_desc, turb_model_t):
53
54          self.desc = model_desc
55          self.fluid = None
56          self.integ = [None, None]
57          self.bw    = [None, None]
58          self.op    = [[None, None],[None, None]]
59          #Minimum acceptable value for the eddy viscosity from the SA model
60          self.eps = 1e-14
61          self.turb_model_t = turb_model_t
62          #Choose the limiter according to the element type
63          self.limiters_switcher = {0:hard_limit_quad(), 1:hard_limit_triang()}
64
65      def container_operator(self, group, key, container):
66
67          k  = ExpansionKeyFactory.make(key.order, key.nip)
68
69          self.integ[0] = class_quad_physintegrator2d(k)
70          self.integ[1] = class_tria_physintegrator2d(k)
71
72          self.bw[0] = class_quad_physbackward2d(k)
73          self.bw[1] = class_tria_physbackward2d(k)
74
75          ctx = EntityOperator.contexts
76
77          self.op[ctx[0][0]][ctx[0][1]] = class_quad_wadg_physforward(k)
78          self.op[ctx[1][0]][ctx[1][1]] = class_quad_rwadg_physforward(k)
79          self.op[ctx[2][0]][ctx[2][1]] = class_tria_wadg_physforward(k)
80          self.op[ctx[3][0]][ctx[3][1]] = class_tria_rwadg_physforward(k)
81
82
83      def setup(self, eqname, domain):
84
85          equation_domain_signature.setup(self, eqname, domain)
86
87          matname = self.desc.get_equation(eqname).mat
88
89          self.fluid = self.desc.get_material(matname)
90
91      def __call__(self, e):
92
93          ###Volumetric values limiting
94          #Recovering the physical values
95          rhonu = e.state(FieldType.ph, FieldVariable.RHONU)
96          rho = e.state(FieldType.ph, FieldVariable.RHO)
97
98          rhonu_min = np.amin(rhonu)
99
```

```
100         #Getting the correct limitercase according to the element type
101         hard_limit = self.limiters_switcher.get(e.type)
102
103         if rhonu_min < self.eps:
104
105             #Reconstructing the inner expansion
106             rho = e.state(FieldType.ph, FieldVariable.RHO)
107             rhonu = e.state(FieldType.ph, FieldVariable.RHONU)
108             rhonu_coeff = e.state(FieldType.tr, FieldVariable.RHONU)
109             logger.debug('RHONU:{}'.format(rhonu))
110             logger.debug('Limiting the turbulence variables on the element {}'.
                    format(e.ID))
111
112             #Limiting when the element mean value is lower than the accepted minimum
113             if rhonu_coeff[0] < self.eps:
114
115                 #When the mean value on the element is negative, all the
                        coefficients should be
116                 #nulled.
117                 rhonu_coeff.fill(self.eps)
118
119                 #Updating the current state
120                 np.copyto(e.state(FieldType.tr, FieldVariable.RHONU), rhonu_coeff)
121
122                 #The new values are converted to the physical space
123                 self.bw[e.type](rhonu_coeff,e.state(FieldType.ph, FieldVariable.
                        RHONU))
124
125                 logger.debug('RHONU:{}'.format(e.state(FieldType.ph, FieldVariable.
                        RHONU)))
126
127             if rhonu_coeff[0] > self.eps and rhonu_coeff.shape[0] > 1:
128
129                 logger.debug('Truncating the higher order modes on the element {}'.
                        format(e.ID))
130                 hard_limit(rhonu_coeff)
131
132                 #Updating the current state
133                 np.copyto(e.state(FieldType.tr, FieldVariable.RHONU), rhonu_coeff)
134                 #The new values are converted to the physical space
135                 self.bw[e.type](rhonu_coeff,e.state(FieldType.ph, FieldVariable.
                        RHONU))
136
137                 logger.debug('RHONU:{}'.format(e.state(FieldType.ph, FieldVariable.
                        RHONU)))
138
139             rhonu = e.state(FieldType.ph, FieldVariable.RHONU)
140             nu = e.state(FieldType.ph, FieldVariable.NU)
141             np.divide(rhonu, rho, out=nu)
142             np.copyto(e.state(FieldType.ph, FieldVariable.NU), nu)
143
144             self.op[e.type][e.subtype](
145                     e.get_jacobian(), e.get_inv_mass(), rhonu,
146                     e.state(FieldType.tr, FieldVariable.RHONU))
147
148             ###Face values limiting
149             # Access workspace storage for face reconstruction
150             FaceReconstruct_f = FaceReconstructWsp_f.get_instance(
151                 *(e.face_comm_sizes), turb_model=self.turb_model_t)
```

```
152
153             num_faces  = len(e.neigh)
154
155             #It searches the indices of the points in which the positivity is violed
                    .
156             for ff in range(num_faces):
157
158                 FaceReconstruct_f.reconstruct_internal(ff, e, self.fluid)
159                 cases_facial_rhonu = np.where(FaceReconstruct_f.rhonuI[ff] < self.
                        eps)
160
161                 if cases_facial_rhonu:
162
163                     logger.debug('Limiting the turbulence variables on the element
                            {}, face {}'.format(e.ID, ff))
164                     logger.debug('RHONU: {}'.format(cases_facial_rhonu))
165
166
167 class LimiterInvoker:
168
169     def __init__(self, model_desc, turb_model_t, limiter_type):
170
171         self.limiter = (globals()[limiter_type])(model_desc, turb_model_t)
172         self.covered_cases = [EqType.RANS_SA, EqType.RANS_SST]
173         self._desc = model_desc
174
175     def apply(self, cm):
176
177         # Access list of PHYSICAL subdomains
178         subds = SubDomainRoleIterator(cm).find(EntityRole.PHYSICAL)
179
180         for s in subds:
181             eqs = self._desc.get_data(s.name).equations
182
183             for eq in eqs:
184
185                 eqtype = self._desc.get_data(eq).type
186
187                 if eqtype in self.covered_cases:
188                     self.limiter.setup(eq, s.name)
189                     foreach_entity_in_subdomain(s, self.limiter)
```