Space adaptive methods with error control based on adaptive multiresolution for the simulation of low-Mach reactive flows

Marc-Arthur N'Guessan

université
PARIS-SACLAY

Space adaptive methods with error control based on adaptive multiresolution for the simulation of low-Mach reactive flows

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°574, mathématiques Hadamard (EDMH)
Spécialité de doctorat: Mathématiques appliquées
Unité de recherche : École Polytechnique, Centre de Mathématiques Appliquées, 91128, Palaiseau, France
Référent : CentraleSupélec

Thèse présentée et soutenue à Palaiseau, le 11-03-2020, par

Marc-Arthur N'GUESSAN

Composition du Jury

Jean-Luc GUERMOND Professeur, Texas A&M University	Président & Rapporteur & Examinateur
Gilles VILMART Maître d'enseignement et de recherche, Université de Genève	Rapporteur & Examinateur
Ludovic GOUDENÈGE Chargé de recherche CNRS, CentraleSupélec	Examinateur
Clinton GROTH Professeur, University of Toronto	Examinateur
Raphaèle HERBIN Professeure, Université d'Aix-Marseille	Examinatrice
Marie POSTEL Maître de Conférences, Université Pierre et Marie Curie	Examinatrice
Marc MASSOT Professeur, École Polytechnique	Directeur de thèse
Christian TENAUD Directeur de recherche CNRS, Université d'Orsay	Co-Directeur de thèse
Vincent GIOVANGIGLI Directeur de recherche CNRS, École Polytechnique	Invité
Guilhem LACAZE Principal numerical physicist, SpaceX	Invité

Thèse de doctorat

NNT : 2020UPASC017

université PARIS-SACLAY

ECOLE DOCTORALE N°574
Mathématiques Hadamard (EDMH)

Titre : Méthodes adaptatives en espace avec contrôle de précision basées sur la multirésolution adaptative pour la simulation d'écoulements réactifs à faible nombre de Mach

Mots clés : Problèmes multi-échelles ; Équations de Navier-Stokes incompressibles ; Multirésolution adaptative ; Contrôle d'erreur ; Schémas de volumes finis collocalisés; Équations Différentielles Algébriques ; Méthodes de Runge-Kutta additives d'ordre élevé ; transport de scalaire passif

Résumé : Ce travail vise au développement de nouvelles méthodes numériques adaptatives pour la simulation numérique de phénomènes physiques multi-échelles en temps et en espace. Nous nous concentrons sur les écoulements réactifs à faible nombre de Mach, caractéristiques d'un grand nombre de configurations industrielles telles que la convection naturelle, la dynamique de fronts de flamme ou encore les décharges plasmas. La raideur associée à ce type de problèmes, que ce soit via le terme source chimique qui présente un large spectre d'échelles de temps caractéristiques ou encore via la présence de forts gradients très localisés associés aux fronts de réaction, génère des difficultés numériques considérables. Il est donc nécessaire de concevoir des méthodes sur mesure pour traiter la raideur de telles applications, afin d'obtenir des résultats d'une grande précision avec un coût calcul raisonnable.
Dans ce cadre général, nous introduisons de nouvelles méthodes numériques pour la résolution des équations de Navier-Stokes incompressibles, une étape importante dans la réalisation d'un solveur hydrodynamique pour les écoulements à faible nombre de Mach. Nous construisons un solveur volumes finis avec adaptation de maillage par l'analyse de multirésolution, qui permet un contrôle a priori des erreurs générées par l'adaptation de maillage. Pour ce faire, nous développons un nouveau schéma de volumes finis collocalisé, avec un traitement original des modes de pression et de vitesse parasites qui n'affecte pas la précision de la discrétisation spatiale. Cette dernière est couplée à un nouveau schéma de Runge-Kutta additif d'ordre 3 pour les écoulements incompressibles, qui présente des propriétés de stabilité adaptées à la raideur des équations différentielles algébriques semi-explicites d'index 2. L'ensemble de cette stratégie est implémentée dans le code de calcul scientifique **mrpy**. Ce dernier est écrit en Python, et repose sur la librairie PETSc, écrite en C, pour le traitement des opérations d'algèbre linéaire. Nous évaluons l'efficacité algorithmique de cette stratégie par la simulation numérique d'un transport de scalaire passif dans un écoulement incompressible sur maillage adaptatif. Ce travail présente donc un nouveau solveur hydrodynamique d'ordre élevé pour les écoulements incompressibles, avec adaptation de maillage par multirésolution et contrôle d'erreur, qui peut être étendu aux écoulements à faible nombre de Mach.

Title : Space adaptive methods with error control based on adaptive multiresolution for the simulation of low-Mach reactive flows

Keywords : Multi-scale problems; incompressible Navier-Stokes equations; Adaptive multiresolution; Error control; Collocated finite-volume schemes; Differential Algebraic Equations; High-order additive Runge-Kutta methods; scalar transport

Abstract : We address the development of new numerical methods for the efficient resolution of stiff Partial Differential Equations modelling multi-scale time/space physical phenomena. We are more specifically interested in low Mach reacting flow processes, that cover various real-world applications such as flame dynamics at low gas velocity, buoyant jet flows or plasma/flow interactions. It is well-known that the numerical simulation of these problems is a highly difficult task, due to the large spectrum of spatial and time scales caused by the presence of nonlinear The adaptive spatial discretization is coupled to a new 3rd-order additive Runge-Kutta method for the incompressible Navier-Stokes equations, combining a 3rd-order, A-stable, stiffly accurate, 4-stage ESDIRK method for the algebraic linear part of these equations, and a 4th-order explicit Runge-Kutta scheme for the nonlinear convective part. This numerical strategy is implemented from scratch in the in-house numerical code mrpy. This software is written in Python, and relies on the PETSc library, written in C, for linear algebra operations. We assess the capabilities of this

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

mechanisms taking place into dynamic fronts. In this general context, this work introduces dedicated numerical tools for the resolution of the incompressible Navier-Stokes equations, an important first step when designing an hydrodynamic solver for low Mach flows. We build a space adaptive numerical scheme to solve incompressible flows in a finite-volume context, that relies on multiresolution analysis with error control. To this end, we introduce a new collocated finite-volume method on adaptive rectangular grids, with an original treatment of the spurious pressure and velocity modes that does not alter the precision of the discretization technique.

new hydrodynamic solver in terms of speed and efficiency, in the context of scalar transport on adaptive grids. Hence, this study presents a new high-order hydrodynamics solver for incompressible flows, with grid adaptation by multiresolution, that can be extended to the more general low-Mach flow configuration.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Remerciements

Je souhaite tout d'abord remercier mes directeurs de thèse, Christian Tenaud et Marc Massot, pour la confiance qu'ils m'ont accordée en acceptant que je travaille avec eux sur cette thèse. J'estime avoir participé à une véritable aventure, d'une intensité intellectuelle extraordinaire, et cela n'aurait pas été possible sans de tels mentors et exemples. Je leur suis particulièrement reconnaissant pour l'opportunité que j'ai eue de pouvoir être chercheur invité à la NASA (Ames Research Center, institution que je remerice d'ailleurs de m'avoir accueilli !) pendant l'été 2018. Ce fut une expérience inoubliable, qui aura sans doute de nombreuses répercussions sur mon avenir.

Je remercie aussi Laurent Séries et Loïc Gouarin, pour toute l'aide qu'ils m'ont accordée pour la bonne prise en main des nombreux outils informatiques dont j'ai eu besoin pendant ma thèse. Ils m'ont notamment introduit à PETSc, ce qui fut une révélation en ce qui concerne l'usage moderne de la science informatique pour s'attaquer à des problèmes d'ingénierie nécessitant l'usage d'architectures massivement parallèles.

Je tiens aussi à remercier le personnel administratif des deux laboratoires dans lesquels j'ai réalisé ma thèse, l'EM2C à CentraleSupélec et le CMAP à l'École Polytechnique, avec une mention spéciale pour Noï Lavaud, Nathalie Rodrigues, Aldjia Mazari et Sébastien Turgis, pour leur disponibilité et leurs bonnes dispositions à mon égard à chaque fois que j'ai eu besoin de leur assistance. Je rermercie mes collègues de travail, notamment Quentin, Pierre, Ruben, Claire, David, Thomas, Sean, et tant d'autres, pour la bonne ambiance de travail et les moments agréables que nous avons partagés durant cette période.

Un grand merci aux personnes des deux résidences dans lesquels j'ai vécu pendant ma thèse ! Merci à ma famille, papa, maman, et mes trois petits frêres Polito, Phiphi et Xavinou, c'est à votre amour et à votre dévouement que je dois d'avoir pu en arriver là !

Je veux finir en remerciant mes amis : Merci à Nico, Amaury, Arthur, Ludo, Léopold (Poldounet !), Julien, Adrien, Adrien, Adrien, Clémentine, Clémence,

Sophie ! Merci à Alan, Ery, Rémi, Mike, Marielle, Arla, Céline, Mathilde, Gaétan, vous faites les meilleures soirées de tout Paname ! Merci à Inès, Agathe, Emma. Un grand merci à Madame Lejeune, Mamé, (et à travers elle à tous les membres de la famille Lejeune avec lesquels j'ai interagi !), pour toutes les fois où elle m'a accueilli rue Galande pour travailler sur ma thèse ! Merci à Marie et Léa. Merci à Brico et Cheickna, ça nous aura pris dix ans mais on l'a fait ! Merci à Mimi. Merci au groupe Hermès : Noémie, Francky, Romane, Claire, merci d'avoir été là quand j'étais dans le creux de la vague... Merci à tous ceux que j'aurai oublié de mentionner, merci de m'avoir accompagné et soutenu dans cette période qui a été beaucoup plus difficile que je ne l'avais imaginé, je ne serais pas parvenu au bout sans vous.

Résumé

Ce travail vise au développement de nouvelles méthodes numériques adaptatives pour la simulation numérique de phénomènes physiques multi-échelles en temps et en espace. Nous nous concentrons sur les écoulements réactifs à faible nombre de Mach, caractéristiques d'un grand nombre de configurations industrielles telles que la convection naturelle, la dynamique de fronts de flamme ou encore les décharges plasmas. La raideur associée à ce type de problèmes, que ce soit via le terme source chimique qui présente un large spectre d'échelles de temps caractéristiques ou encore via la présence de fort gradients très localisés associés aux fronts de réaction, génère des difficultés numériques considérables. Si l'on considère par exemple le cas de la combustion, l'on veut en général réaliser des simulations numériques s'étalant sur quelques secondes, alors que les échelles les plus rapides dûs aux réactions chimiques peuvent descendre jusqu'à l'ordre de la nanoseconde. Si l'on veut simuler une flamme de laboratoire, avec une description précise de la géométrie du front de flamme, ce dernier est de l'ordre du dixième de millimètre, tandis que la zone de combustion a une taille caractéristique de l'ordre de la dizaine de centimètres. Un calcul DNS en 3D d'un tel phénomène sur un maillage uniforme suffisamment fin pour décrire les plus petites échelles spatiales nécessite donc la prise en compte de plusieurs milliards de points de calcul. Il est aujourd'hui possible de réaliser des calculs aussi coûteux, mais ils demandent l'utilisation d'architectures informatiques massivement parallèles, avec des dizaines voire des centaines de milliers de processeurs en parallèle. De telles ressources sont hors de portée des principaux acteurs, aussi bien scientifiques qu'industriels, en matière de CFD. Il est donc nécessaire de concevoir des méthodes sur mesure pour traiter la raideur de telles applications, afin d'obtenir des résultats d'une grande précision avec un coût calcul raisonnable.

Une première méthode efficace pour réduire la raideur de ce type de problèmes consiste à considérer des modèles asymptotiques qui permettent de supprimer certaines des échelles les plus rapides, sans impact majeur sur la précision descriptive du modèle. Par exemple dans le cas des écoulements réactifs, de nombreux cas d'applications réels sont très bien décrits par les équations de Navier-Stokes à faible nombre de Mach, dans lesquelles sont supprimées les ondes acoustiques, qui jouent un très faible rôle dans le transport d'énergie lorsque la

vitesse d'écoulement du fluide est très faible par rapport à la vitesse du son. Ces équations se décomposent en deux principales parties : d'une part les équations décrivant l'hydrodynamique du système et qui traduisent la conservation de la masse et de la quantité de mouvement, et d'autre part les équations de transport qui traduisent la conservation des espèces et de l'énergie. Il est alors possible de concevoir deux solveurs différents pour s'attaquer à ce problème, un solveur hydrodynamique que l'on couple à un solveur d'équations de type convection-réaction-diffusion. De nombreux progrès ont été réalisés récemment pour la construction de solveurs efficaces pour les problèmes de transport, et l'on peut citer comme exemple les méthodes de séparation d'opérateurs avec pas de temps adaptatif et contrôle d'erreur, qui sont des méthodes sur-mesure pour s'attaquer au caractère multi-échelle en temps. Les techniques d'adaptation dynamique de maillages quant à elles font partie des principales familles d'algorithmes conçus pour s'attaquer à l'aspect multi-échelle en espace. La première génération de méthodes d'adaptation dynamique de maillages fortement plébiscitées dans la communauté du calcul scientifique est née au début des années 80, des travaux de M. Merger. Ce sont les techniques d'"Adaptive Mesh Refinement (AMR)", dont certaines des implémentations les plus récentes sont réalisées sur architectures massivement parallèles, notamment au Lawrence Berkeley National Laboratory, suivant les travaux pionniers de J. Bell et M. Day. La seconde génération de méthodes d'adaptation dynamique de maillages apparaît quant à elle au milieu des années 90, et repose sur la décomposition en ondelettes. Elle permet de faire de l'adaptation dynamique de maillage avec contrôle d'erreur par rapport à une solution obtenue sur maillage uniforme. C'est cette méthode qui a été retenue par les équipes CFD de l'entreprise SpaceX.

Dans ce cadre général, nous introduisons de nouvelles méthodes numériques pour la résolution des équations de Navier-Stokes incompressibles, une étape importante dans la réalisation d'un solveur hydrodynamique pour les écoulements à faible nombre de Mach. Nous construisons un solveur volumes finis avec adaptation de maillage par l'analyse de multirésolution, qui permet un contrôle a priori des erreurs générées par l'adaptation de maillage. Pour ce faire, nous développons un nouveau schéma de volumes finis collocalisé, avec un traitement original des modes de pression et de vitesse parasites qui n'affecte pas la précision de la discrétisation spatiale. Cette dernière est couplée à un nouveau schéma de Runge-Kutta additif d'ordre 3 pour les écoulements incompressibles, qui présente des propriétés de stabilité adaptées à la raideur des équations différentielles algébriques semi-explicites d'index 2. L'ensemble de cette stratégie est implémentée dans le code de calcul scientifique `mrpy`. Ce dernier est écrit en Python, et repose sur la librairie PETSc, écrite en C, pour le traitement des opérations d'algèbre linéaire. Nous évaluons l'efficacité algorithmique de cette stratégie par la simulation numérique d'un transport de scalaire passif dans un écoulement incompressible sur maillage adaptatif. Ce travail présente

donc un nouveau solveur hydrodynamique d'ordre élevé pour les écoulements incompressibles, avec adaptation de maillage par multirésolution et contrôle d'erreur, qui peut être étendu aux écoulements à faible nombre de Mach.

Contents

List of Figures

List of Tables

General Introduction

Numerous physical phenomena result from the interaction of several strongly coupled sub-processes that involve a wide spectrum of spatial and time scales spanning over several order of magnitudes, due to the presence of highly nonlinear mechanisms taking place into dynamic fronts. Let us start with examples originating from our specific field of study, multicomponent reacting flows. The first one is related to combustion flow processes. Classical configurations of interest for scientific investigations are the propagation of premixed methane-air flame [NWK98a], [NWK99], [Gio99], or axisymmetric methane-air diffusion flame [MTS+98], [NG09], [DB00b]. Methane and air are typically injected into a combustion chamber whose characteristic size is of the order of ten centimeters. The injection area is few millimeters wide, and the (laminar) flame thickness is around a tenth of millimeters. These phenomena involve various species submitted to nonlinear chemical reaction processes, diffusive and convective processes, and with heat release that affects the thermodynamical state of the flow. In [NW97], Najm and Wyckoff studied the interaction of a premixed stoichiometric methane-air flame, with a counter-rotating vortex pair in 2D, under atmospheric pressure conditions. The initial flame thickness is 0.06 cm, and the initial laminar burning speed is around 20 cm/s. The vortex pair center-to-center distance is 0.25 cm, and the maximum flow velocity is 20 m/s. This means that the flame time scale is around 3 ms, and the convective time scale is around 0.13 ms. They used a subset of the detailed methane-air chemistry mechanism, where the shortest reactive time scales are of the order of nanoseconds.

The second example that we will consider comes from plasma / flow interactions. Plasma sources at atmospheric pressure have been recently developed for different engineering applications. One way to produce them is to use nanosecond repetitively pulsed discharges in an insulating fluid (for example air, or another gas mixture) [DBMB15], [DBM+12], [TB14]. The region of interest in these experiments is situated between an anode and a cathode (two parallel plates for example). A constant high electric field is repetitively applied over this region, during a few nanoseconds, with a period of a few microseconds. This electric field accelerates the electrons, that in turn ionize the ambient gas and cause an electron avalanche [VPB94] . The electron avalanche entails fast

variations of the net charge density that create an induced electric field. This causes nonlinear ionizing waves that drive the formation of thin plasma filaments, also known as streamers discharges. Here, the effective ionization and the dielectric relaxation times scales are of the order of the picosecond, and the discharge propagation over few centimeters takes place during few nanoseconds. The streamer discharges act upon the gas flow momentum, the time scale of which is around few milliseconds. As for the spatial scales, the Debye length at atmospheric pressure (involved in the electron avalanche) can be of micrometric scale, whereas the space between the elecrodes is about a few centimeters.
A good understanding of such complex phenomena is a highly challenging task that is nonetheless essential for a great number of scientific and engineering applications. Fortunately, a large amount of time and energy has been deployed over the past decades in order to build a hierarchy of comprehensive mathematical models to describe such reactive strongly multiscale flows. They resulted in various systems of Partial Differential Equations (PDEs) in the space and time dimensions, encompassing all the physically relevant scales. It is thus possible, theoretically, to gain insights regarding these physical processes by solving the PDEs modelling them with a sufficiently accurate resolution. This is the goal of the technique of Direct Numerical Simulations (DNS). However, the numerical resolution of reactive flows is extremely demanding in terms of computational resources, due to their multi-scale properties. Let us exemplify this fact with the combustion process mentioned earlier. In a typical numerical simulation, one discretizes the physical domain, in this case the combustion chamber. If we consider a uniform discretization, an accurate description of the flame requires at least ten computational points spanning the flame thickness. This means that the typical mesh size is 10^{-5} meter, and the computational domain will consist of 10^{12} mesh points (in a three-dimensional space). Such tremendous requirements rendered DNS combustion out of reach for years. Nevertheless, by harnessing the capabilities of massively parallel computing architectures, researchers have been able to perform computations at this scale. For example, a DNS solver, $s3d$, has been developed at the Lawrence Berkeley National Laboratory [CCdS+09], in order to produce high-fidelity simulations of turbulent flames. It was lately used to perform a petascale DNS of premixed hydrogen combustion [HCK+12], with nearly 7×10^9 grid points, on $120,000$ process cores.
The *Hybrid* code, developed by Larsson *et al.* at the University of Maryland[1], gives us another example of petascale computations on massively parallel architectures. In [BMBL+13], Bermejo-Moreno *et al.* used this code for the direct numerical simulation of isotropic turbulence and its interaction with shock waves. They solved the compressible Navier-Stokes equations on a parallel system with approximately 2 million cores, for a maximum number of 4.12×10^{12}

[1] http://terpconnect.umd.edu/~jola/supercomputing.html

simulated grid points and a memory usage around 1.6 PB. Thus, we are now able to perform the DNS of reactive flows with the most refind model of the hierarchy, that is by solving the compressible Navier-Stokes equations coupled to complex chemistry and detailed transport. This represents a significant scientific and engineering accomplishment, but since these simulations entail very high computational costs, either in terms of CPU time to solution, memory trace or data handling, they rely on massively parallel computational resources that are not available to most industrial players or laboratories at the moment. Making large scales simulations accessible on regular computational architectures necessitates a change in paradigm, through the use of asymptotic limits of models for specific applications, and tailored algorithms especially designed to tackle multiscale phenomena, with lower memory trace and shorter time-to-solution.

Flows occuring at a low Mach number, where the flow velocity is relatively small compared to the local speed of sound, are a typical case where it is possible to use an asymptotic limit of the fully compressible Navier-Stokes equations. These flows cover a large number of real-world applications and engineering configurations, such as natural convection [Get98, MGDV02], flame dynamics at low gas velocity [NWK98a, NWK99, RPB14], or buoyant jet flows [BT69, HK05]. In those cases, we can assume that the density relative variations due to the pressure can be neglected, and that the soundwaves have a little impact on the dynamics of the overall system. Although the fully compressible Navier-Stokes equations still give the most accurate description of the flow dynamics, it becomes increasingly difficult to solve reacting flows with this model as the Mach number decreases [GV99, DJOR16]. This is due to the fact that the compressible solver has to take into account the time scales related to the acoustic waves, thus imposing severe restrictions on the time step to ensure the stability of the computation. It is thus more natural to describe such flows with a different mathematical model, and Madja & Sethian [MS85] were the first to propose a model for low-Mach number flows. They effectively suppress the acoustic waves for the physics at hand, by performing an asymptotic development of the fully compressible equations in the Mach number parameter. For a mixture gas model for example, the final equations can be decomposed into (i) a set of advection-diffusion-reaction equations for the conservation of the species and the energy conservation, (ii) momentum balance equations for the flow velocity, and (iii) a mass conservation equation that translates into an algebraic constraint on the divergence of the velocity field. It is thus possible to decouple the spatial and time scales related to the transport phenomena of the species and the temperature or enthalpy on one hand, and those related to the flow dynamics on the other hand.
The low-Mach number approximation thus filters the acoustic waves from the

Navier-Stokes equations, but they also result in a change of the mathematical structure of reactive flows. While the fully compressible Navier-Stokes equations are hyperbolic-parabolic equations, the low-Mach number formulation is a set of parabolic-hyperbolic-elliptic equations. As a consequence, the fully compressible equations and the low-Mach number equations are two structurally different mathematical models, and this dichotomy results in strong differences in the numerical methods deployed to solve them. For the latter case, a sound algorithmic strategy, that takes advantage of the natural decomposition of low-Mach number equations detailed above in a clever manner, consists in *splitting* the resolution of the complex chemistry and detailed transport on one hand, and the resolution of the flow dynamics on the other hand. The former part is a set of advection-diffusion-reaction equations, three processes that are very different physical mechanisms, occuring over a large range of time and spatial scales. Therefore this set of equations can also be resolved with an operator-splitting strategy. These two splitting procedures have been the basis of a large number of numerical schemes to solve reactive and non-reactive flows described by the low-Mach number formulation [DB00a, SRN10, NWK98a, NWK99, NK05, Nic00, Sha12]. Time operator-splitting constitutes a very good example of algorithms tailored to efficiently solve multiscale low-Mach number flows. But they necessarily entail splitting errors, with regard to fully coupled solvers, and these errors have been seldomly treated in the literature. We believe that splitting error control is an important point of improvement for operator-splitting strategies to solve reactive flows, because these errors can alter the numerical accuracy of the calculated solutions.

Another great strategy to reduce the time to solution and memory trace of problems with a wide range of spatial scales, consists in resorting to dynamically adaptive grid techiques, that confine the use of fine mesh cells to computational regions where the smallest scale processes are taking place, while employing coarser mesh cells elsewhere. This entails important memory space savings, so that rather large computational domains can be solved with common computational resources. Adaptive grid techniques are particularly efficient when simulating physical phenomena with localized wave fronts moving across the domain. The computational domain is then characterized by a highly inhomogeneous spatial scales distribution, and it is possible to confine the use of extremely refined mesh points to a very small part of the domain. The first widely adopted adaptive grid techniques belong to the family of Adaptive Mesh Refinement (AMR), pioneered in the eighties with the work of Berger [BO84]. The goal was initially to solve multidimensional, time dependent shock hydrodynamic problems [BC89]. Since then, several numerical simulation applications have been performed with variants of the original procedure [DB00a, SRN10, FPG15, PHB+98, ABC+98, NG09, BS99, GTG15, Pop03],

and they have even been performed on parallel architectures. We can cite a work of Bell *et al.* [BDS+05], where they performed the DNS of a laboratory-scale turbulent V-flame of premixed methane-air with detailed chemical kinetics, using AMR to resolve the flame and turbulent structures, on a parallel computer with 128 processors. For a state-of-the-art parallel implementation of AMR techniques for multidimensional reactive flow simulations, one can refer to the **AMReX**[2] code developed at Lawrence Berkely National Lab [ZAB+19]. This code was successfully used for the DNS of various combustion configurations [DTB+15, ADB15], and for astrophysical hydrodynamics simulations [ABB+10, ZHA+11, ZHA+12]. One can also refer to the **canoP**[3] code, an open-source project led by the « Maison de la simulation » research laboratory, that leverages the parallel AMR features of the p4est library [BWG11]. It was used for example for the simulation of multiphase flows (see [Dru17] and references therein). We will give more details on key aspects of AMR techniques in this work, but for now we mention that AMR algorithms are generally plagued with two drawbacks that have a negative impact on the reliability of their results: (i) the refinement process by which the code decides to employ finer or coarser meshes usually relies on heuristic ad-hoc criteria, and (ii) it is often not possible to control the ineluctable error between the adapted grid computation and a computation performed on a uniform fine grid[4]

Multiresolution-based adaptive grid algorithms, the second generation of adaptive grid techniques born in the nineties, overcome these two downsides. Relying on the mathematical properties of wavelets basis [DeV98], they allow to build local regularity indicators of a function. Then, by using these indicators, it is possible to build a hybrid discretization of this function, while controlling the error between the adaptive representation and a uniformely refined representation [Har94a], [Har95], [CKMP03]. The first numerical simulations with

[2] https://ccse.lbl.gov/AMReX/index.html

[3] https://gitlab.maisondelasimulation.fr/canoPdev/canoP

[4] In [NFG17], Narechania, Fréret and Groth use adjoint-based error estimation to direct the output-based Adaptive Mesh Refinement procedure for the numerical simulation of inviscid and viscous compressible flows. The adjoint-based a posteriori error estimates are defined by solution-dependent engineering functionals. One can also consult the web page of the Cart3D code with the joint work of M. Aftosmis at NASA (collaboration with M. Berger) for steady and unsteady compressible flows: https://www.nas.nasa.gov/publications/software/docs/cart3d/pages/.

However, the approach entails the resolution of a large sparse linear system, and this procedure has to be done several times before reaching a precise enough adaptive grid, thus requiring a rather heavy overhead compared to traditional AMR techniques, especially in the unsteady cases. The user must have in mind a dedicated quantity, the error on which he or she wants control, and the estimate does not provide an access to the error on the primal variable flow fields but really focuses on such an output-based mesh refinement. To the extent of our knowledge, such techniques, even if quite interesting for the compressible problems mentioned above, are not well adapted for our combustion/plasma based strongly unsteady applications.

adaptive multiresolution appeared in the nineties with the seminal papers of Harten [Har94a], [Har95], and since then they have been employed in numerous cases to compute reacting flows. In [Ten10], Tenaud conducted numerical simulations on viscous compressible flows with high frequency oscillations in supersonic regimes, using a fully adaptive multiresolution finite-volume technique. It demonstrates the capability of such schemes to perform dynamic grid adaptation within user-defined tolerance. Multiresolution adaptive techniques were also used in [For16] for the numerical simulation of shock-bubble interaction. And we finally mention that these techniques are the basis for the numerical solver developed at the company SpaceX for their rocket engine design [JL15].

In this context, Duarte, in his thesis [Dua11], introduced a new generation of time-space adaptive numerical schemes with error control to tackle low-Mach reacting flows. This work focused on the computationally effective treament of the advection-diffusion-reaction transport equations[5]. Their efficient resolution was then carried out with a second-order Strang splitting scheme [Str68], with dynamic splitting time steps. The novelty of this approach compared to previous splitting methods was that the subproblems were solved with dedicated high-order one-step methods, in order to ensure that the temporal error of the scheme was only related to the splitting scheme: the 4^{th} order stabilized explicit ROCK4 solver for the diffusion part [Abd02], the 5^{th} order implicit RADAU5 solver for the reactive part [HW96] and the 3^{rd} order OSMP3 for the advective part [DT09]. A shifted embedded Strang splitting scheme was then used to determine adaptive time steps, while controlling the errors due to the operator splitting technique. A particular case of interest in his work was the simulation of reaction fronts, that is low-Mach reactive flows with the presence of reactive waves. In such configurations, a very high chemical activity occurs in highly localized fronts that move throughout the computational domain; these conditions are typical of plasma discharges [EMB+06] or human ischemic strokes [DDD+13]. Therefore, in order to capture the chemical waves in these applications, the adaptive operator splitting strategy was coupled to an adaptive multiresolution finite-volume scheme, resulting in a highly efficient time-space adaptive numerical scheme for advection-diffusion-reaction processes (see *, e.g.,* [DDT+13] for the performance evaluations of this numerical scheme for the propagation of premixed flames, and the ignition of diffusion flames).

The present work was originally motivated by the desire to further complement this strategy toward a low-Mach solver with adaptation in time and space and

[5] It represented a first step toward a sound mathematical framework to combine (i) the optimality in cost reduction for low Mach reacting flows obtained by tailored numerical methods such as adaptive grid techniques and operator-splitting, and (ii) accuracy control in the space and time discretization.

error control. Several building blocks were still missing; first the hydrodynamics and velocity fields were given analytically and not resolved in [Dua11, DDT+13]. Providing a hydrodynamics solver for the Navier-Stokes equations in the low-Mach variable density limit, within the proposed framework, is a major step forward since several key issues were to be resolved in order to complete such a program: (i) the proposed solver has to fit within a finite volume framework for the multiresolution with collocated variables, (ii) since the strategy relies on sub-step solver of high order in time with error control, the time-integration has to be high-order, that is at least third order, thus eliminating the traditionally used time splitting methods, (iii) since implicit methods are envisioned, a specific and efficient linear algebra solver has to be designed. The second building block is related to data structure for both sequential and parallel implementation, optimization and efficiency. The original implementation was conducted in Fortran [TD11b] and relied on tree of pointer recursive navigation. It was more dedicated to a proof of concept than to an efficient and optimal implementation. A piece of work lead by T. Dumont, in collaboration with M. Duarte and INTEL (T. Guillet), allowed the use of space filling curve (z-curve or « courbe de Lebesgue » in French) and Morton index in order to reach an implementation on shared-memory architecture (Xeon-Phi) with close to ideal scaling [DDD+15]. Even if such a strategy is also used by the close approach of cell-based AMR [Dru17, BWG11], it became however clear that room for improvement existed in order to obtain better data locality and efficient treatment of prediction operator and its intrinsic recursive nature, all the more on distributed memory architecture for massively parallel computing. Since the question of the data structure has been only partially tackled during the thesis, we will come back to this piece of work in the General Conclusion and Prospects section and we directly focus on the hydrodynamic solver.

Generally, the first step in designing a numerical scheme to solve the low-Mach Navier-Stokes equations consists in the numerical resolution of the *incompressible* Navier-Stokes equations in the velocity $\boldsymbol{u} = (u_i(x,y,z,t))_{i=1,\cdot,3}$ and pressure $p(x,y,z,t)$ unknowns. Indeed, the incompressible Navier-Stokes equations share most of the features of the mathematical structure of the low-Mach Navier-Stokes equations, so that when one has been able to circumvent the difficulties related to the resolution of the former, one can extend his method to the resolution of the latter in a rather straightforward fashion. We restate them here:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}^t \otimes \boldsymbol{u}) + \nabla p - \nu \, \Delta \boldsymbol{u} = \boldsymbol{f} \quad \text{in } \Omega \times \mathbb{R} \\ \nabla \cdot \boldsymbol{u} = 0 \quad \text{in } \Omega \times \mathbb{R} \end{cases} \tag{1}$$

with a homogeneous Dirichlet boundary condition for $\boldsymbol{u}$ and the initial condi-

tion:

$$\boldsymbol{u}(\cdot, 0) = \boldsymbol{u}_{\text{ini}} \quad \text{in} \quad \Omega$$

where ν is the cinematic viscosity of the fluid and $\boldsymbol{f}$ is a source term. We make the following assumptions:

1. $\nu \in]0, +\infty[$,
2. Ω is an open bounded domain of $\mathbb{R}^3$,
3. $\boldsymbol{u}_{\text{ini}} \in L^2(\Omega)^3$,
4. $\nabla \cdot \boldsymbol{u}_{\text{ini}} = 0$ in Ω,
5. $\boldsymbol{f} \in L^2(\Omega \times \mathbb{R})^3$.

These equations are among the most famous sets of PDEs, and their study and better comprehension have been a major research topic in various branches of mathematics, physics and engineering over the past century, and with good reason. Since their discovery in the nineteenth century, researchers have not yet been able to prove that they always possess smooth solutions. The *Navier-Stokes existence and smoothness* problem was even made one of the seven **Millenium Prize problems** by the **Clay Mathematics Institute** in May 2000[6]. To be more precise, we know that satisfying solutions exist in two dimensions [Lad69], but we have not yet been able to prove that given any initial divergence-free velocity field $\boldsymbol{u}_{\text{ini}}$, there exist (or do not exist, for that matter) smooth functions $\boldsymbol{u}$ and p that satisfy equations (1) in three dimensions (the relevant case for real-world applications and flows) for an infinite time interval, with bounded energy. Still, this situation did not prevent CFD practitioners from attempting to build efficient and accurate approximate solvers for these equations. In fact, quite the contrary actually happened, to the point that an *overwhelming* quantity of articles, scientific papers and books have been devoted to that specific topic. A full review of the matter is far beyond the scope of this thesis, and we will limit our discussion in this introduction to two well-known issues that appear when trying to solve (1) with a spatial discretization handled via the finite-volume technique. The first one is the apparition of the pressure and / or velocity spurious modes. This problem is mainly related to the Stokes part of the equation, and it appears even on their steady version (we get this system of PDEs by removing the time derivative of the velocity, and the convective term in the momentum equations). Upon spatial discretization of the steady Stokes equations, we end up with a linear system on the (discrete) velocity and pressure variables. The linear system is a saddle-point problem [Saa03], [BGL05], and depending on the particular choice of the finite-volume discretization scheme, its matrix may have more kernel modes than the continuous problem. These

[6] https://www.claymath.org/millennium-problems/navier-stokes-equation

spurious / parasite kernel modes will appear when trying to solve the linear system to obtain the velocity and the pressure, and pollute them. The original solution for finite-volume schemes applied to the incompressible Navier-Stokes equations is to *stagger* the velocity and the pressure [HW65]. Although this solution completely solves the difficulty in case of uniform Cartesian grids, its extension to more complex grids and geometries is a delicate task [RP13], [GTG15], [CEH09]. Hence, we can expect difficulties when trying to apply such a remedy to the spatial discretization of equations (1) combined to a multiresolution adaptive scheme. The second issue is encountered after the spatial discretization. The semi-continuous problem in the time variable obtained is not a simple set of Ordinary Differential Equations (ODEs), but a Differential Algebraic Equation (DAE). We know since the seminal article of Petzold that "DAEs are not ODEs", and they are incredibly more difficult to efficiently solve than ODEs [HW96]. In particular, a large range of numerical schemes, which give high-order solutions when applied to ODEs, can only yield low-order solutions when applied to the semi-discretized Navier-Stokes equations. Thus, a straightforward implementation of classical numerical methods (among them the famous prediction-projection schemes [Cho68], [Tem69]) is ill-advised if the goal is to build efficient numerical solvers based on the operator-splitting strategy mentioned earlier, where each subproblem has to be solved with high-order methods. For instance, there has not been any successfull attempt to obtain prediction-projection schemes with an higher-than-second-order precision for the incompressible Navier-Stokes equations [GMS06]. We will give a more detailed account of these issues, and other difficulties encountered when solving the incompressible Navier-Stokes equations, throughout this thesis.

We are now able to express the purpose of the present work. It falls within the general framework of the development of a new generation of dedicated numerical tools for the simulation of multi-scale reacting flows; in particular, the numerical resolution of low-Mach regime reacting flows with adaptive time-operator splitting and adaptive multiresolution finite-volume schemes, with error control. To this end, the following study introduces mathematical and numerical concepts used to build high-order numerical solutions to the incompressible Navier-Stokes equations with adaptive multiresolution in a finite-volume context. We design a new collocated finite-volume scheme based on the variational formulation of equations (1), that allows us to implement a fully adaptive multiresolution finite-volume scheme for incompressible flows. To the best of our knowledge, such a scheme did not exist in the literature when we started this thesis. We also investigate a set of already existing high-order Runge-Kutta schemes for the numerical integration of the semi-discretized Navier-Stokes equations. We review their mathematical properties, in order to determine the best suited methods to obtain 3^{rd} order (or higher) numerical

solutions to these equations. We also introduce a new class of 3^{rd} order *Additive Runge-Kutta (ARK)* methods that take advantage of the structure of the semi-discretized incompressible Navier-Stokes equations, and show that they possess better properties than existing schemes. At the beginning of this study, we did not have at our disposal flexible enough computational tools to be able to build and test various spatial and temporal schemes to solve incompressible flows coupled to an adaptive multiresolution strategy. That is why a large amount of time during this thesis was dedicated to build from scratch a software to implement and test our new techniques. This resulted into an academic in-house code called `mrpy`, written in Python.

This thesis is divided into seven chapters. Chapter 1 is devoted to space adaptive multiresolution techniques for dynamic adapted grids. We give a short introduction to the notion of nonlinear approximation, and expose the AMR technique in this mathematical framework. Then we give the mathematical background behind the wavelets theory upon which are constructed multiresolution schemes, and end up with our specific adaptive multiresolution strategy.
Chapter 2 deals with our spatial discretization scheme for the incompressible Navier-Stokes equations on grids obtained by adaptive multiresolution. We start with a short review of different solutions proposed over the years for the discretization of these equations on non-uniform grids, with a special focus on the schemes combined to an AMR strategy for grid adaptation. We then build our specific collocated scheme.
In Chapter 3 we give some insights regarding the mathematical structure of the semi-discretized incompressible Navier-Stokes equations. We chose to first present some characteristics of stiff Ordinary Differential Equations, because for our cases of interest, DAEs can be viewed as ODEs with *infinite stiffness*. We also give a brief account of projection methods, with an emphasis on their limitations.
Chapter 4 considers the time integration of Hessenberg index 2 DAEs by one-step Runge-Kutta methods. We present the requirements of such methods to yield high-order solutions to the Navier-Stokes equations, and exhibit some 3^{rd} order schemes. We expose our construction process for a new class of additive Runge-Kutta methods well suited for these equations. We assess the order of these schemes with various test cases of well-known incompressible flows.
Chapter 5 is about the algorithmic implementation of the adaptive multiresolution strategy in the `mrpy` code. We present here our new fully adaptive multiresolution scheme to solve the incompressible Navier-Stokes equations, and discuss some practical issues regarding the data structure and code implementation.
In Chapter 6 we put together the results of Chapters 2, 4 and 5 to present a new high-order numerical scheme to solve the incompressible Navier-Stokes

coupled to a finite volume adaptive multiresolution strategy. Special care is given to the treatment of the velocity spurious modes.
Finally Chapter 7 considers the application of our new space adaptive hydrodynamic solver to the transport of a passive scalar. We assess the performance of multihierarchy multiresolution scheme compared to a simulation performed on a uniform grid. This study has been recently published in the Journal of Computational and Applied Mathematics [NMST19].

Publications

1. "**M.-A. N'Guessan**, M. Massot, L. Séries and C. Tenaud", "High order time integration and mesh adaptation with error control for incompressible Navier-Stokes and scalar transport resolution on dual grids", *Journal of Computational and Applied Mathematics*, 2019

Book chapters

1. **M.-A. N'Guessan**, L. Séries, C. Tenaud and M. Massot, "A high-order Runge-Kutta method coupled to a multiresolution strategy to solve the incompressible Navier-Stokes equations", NASA Technical Memorandum, Proceedings of the Heliophysics Modeling and Simulation summer 2018 Summer Program, NASA Ames Research Center, 2018

2. H.Leclerc, **M.-A. N'Guessan**, L. Séries, L. Gouarin and M. Massot, "Parallel dedicated data structures and adaptive multiresolution implementations: application to the resolution of multi-scale PDEs", NASA Technical Memorandum, Proceedings of the Heliophysics Modeling and Simulation summer 2018 Summer Program, NASA Ames Research Center, 2018

2018 Nasa summer program

During the summer 2018, I had the chance to be invited to the Heliophysics Modeling and Simulation summer program at NASA Ames Research Center. It was a great opportunity to exchange with the scientists of the NASA advanced Supercomputing division. I specifically worked on high-order Runge-Kutta schemes for Hessenberg index 2 Differential Algebraic Equations, and a new data structure for the parallel implementation of adaptive multiresolution algorithms in collaboration with Hugo Leclerc, Loïc Gouarin and Laurent Séries, during this period. The results of this work were published in two proceedings to the NASA technical Memorandum.

This PhD was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

Chapter 1

Space Adaptive Multiresolution for Multi-Scale Evolutionary PDEs

This chapter describes the core concepts underlying the adaptive grid technique that we will implement in this work. Within the framework of the efficient resolution of multiscale PDEs, the key aspects to consider for the choice of a specific space adaptive strategy are (i) its ability to detect and track local space scales, (ii) the ability to control the errors between the solution obtained on the adaptive grid and the solution obtained on a uniform refined grid and (iii) the computational overhead generated by its implementation. We begin this chapter by a small introduction to approximation theory, and its link to adaptive grid techniques. We then give a general description of adaptive mesh refinement (AMR) schemes, before introducing the mathematical theory behind the multiresolution analysis. We conclude this chapter with a practical implementation of adaptive multiresolution, that will be the basis for the fully adaptive multiresolution scheme for the numerical simulation of incompressible flows.

1.1 Approximation theory and adaptive grids strategies

A lot of physical phenomena are subject to a highly inhomogeneous spatial distribution due to the large range of spatial scales of the quantities involved. Examples of such situations are brain strokes [DDD+13], plasma discharges at atmospheric pressure [DBM+12, TB14] or reactive flows phenomena such as combustion [DDT+13, BDS+05, DB00b, SRN10, NWK98a, NWK99, Che11, CCdS+09, HCK+12]. The numerical simulation of these phenomena is an ex-

tremely difficult task, among other reasons because of the huge number of degrees of freedom necessary to acurately describe them by uniform grid computations. Indeed, in the presence of localized fronts of steep spatial gradients into a smooth domain, the sharp regions require a mesh size that can be order of magnitudes smaller than what is necessary in the smooth regions. Thus an excessive computational effort is spent in smooth regions if they are discretized with highly refined grids. What is more, it often happens that the localized fronts move throughout the domain, so that it is not possible in advance to know where refinement is necessary. This situation motivated over the years the development of *adaptive mesh techniques*, that computational strategies that dynamically adapt the grid in order to track singularities and use fine meshes only in these parts of the domain.
We believe that the appropriate mathematical setting to discuss and evaluate the preformance of adaptive grids strategies is that of **approximation theory**. Generally speaking, it is the branch of functional analysis that deals with the problem of approaching a rather complex *target function* by simpler functions called *approximants*. Improving the quality of the approximation is usually done by increasing the complexity of the approximants, and approximation theory deals with the trade-off between accuracy and complexity. In numerical simulation, in the finite-volume scheme context, we consider physical quantities as functions of space and time variables, and we try to approach them by piecewise constant functions that lie in finite-dimensional spaces. Finding the best numerical approximation (*i.e.* the one involving the least possible degrees of freedom without sacrificing precision) is a part of the more general goal of approximation theory. We will thus give some elements of approximation theory, before diving into adaptive grids techniques.

1.1.1 Nonlinear approximation

Let u be a real function over an open domain $\Omega \subset \mathbb{R}^d$, where d is an integer. We are interested in the approximation of u by *piecewise constants* functions on Ω. We will use the following case for illustration purpose: Let $\Omega =]0,1[$, and u be the following function:

$$u(x) = \tanh(50 \cdot |x - \tfrac{1}{2}|) \quad \forall x \in \Omega \tag{1.1}$$

This function is represented in figure (1.1).
Next for $n \in \mathbb{N}^*$ an integer, we define a partition of the segment Ω into n disjoint intervals. We can consider for example partitions consisting of dyadic intervals: for $l \in \mathbb{N}^*$, we define the grid Ω_l as the union of cells $K_i^l =]2^{-l}i, 2^{-l}(i+1)[$ for $i \in \{0, 1, 2, \ldots, 2^l - 1\}$ (see figure (1.2)).

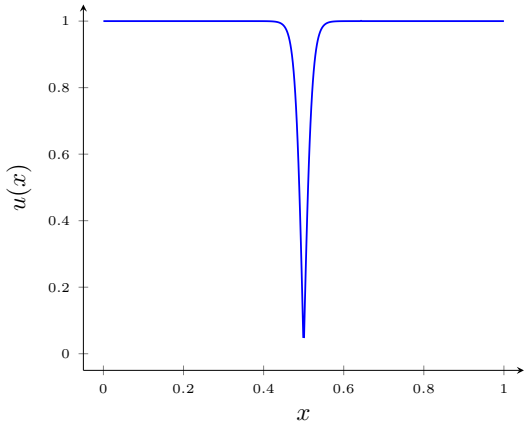


Figure 1.1: $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$

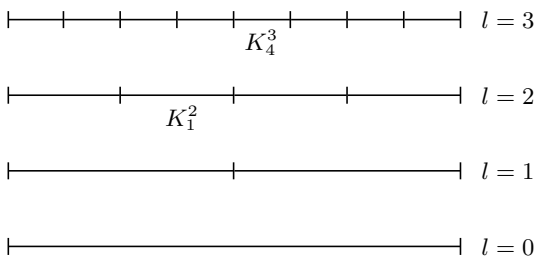


Figure 1.2: Dyadic grids on the segment $]0, 1[$

Let φ_i^l be the characteristic functions on each of these cells:

$$\varphi_i^l(x) = \begin{cases} 1 & \text{if } x \in K_i^l \\ 0 & \text{otherwise} \end{cases}$$

Let V_l be the space of piecewise constant functions for this partition. Then the φ_i^l form an orthogonal basis of V_l, and for each $F \in V_l$ there exists a unique set of real coefficients c_i^l for $i \in \{0, 1, 2, \ldots, 2^l - 1\}$ so that:

$$F = \sum_{i=0}^{2^l-1} c_i^l \varphi_i^l$$

For $0 < p \leq +\infty$, we can look for the best approximation of u by elements F of the approximation space V_l in the L_p-norm. We define the error in approximating a function $u \in L_p(\Omega)$ by functions in V_l by:

$$e(u, V_l)_p = \inf_{F \in V_l} \|u - F\|_{L_p} \tag{1.2}$$

In the case $p = 2$, since $L_2(\Omega)$ is an Hilbert space, the best approximation F is obtained by the projection of u onto the subspace V_l of L_2:

$$F = \sum_{i=0}^{i=2^l-1} \langle u, \varphi_i^l \rangle \varphi_i^l$$

where $\langle \cdot, \cdot \rangle$ is the canonical scalar product on L_2. We speak in this case of a *linear approximation*, because the approximation space is a linear space. For our function defined in (1.1), we show in figures (1.3-1.6) the best approximation in the L_2-norm for different values of the grid level l.
We considered a uniform partition of Ω, but this discussion is still valid if the partition Π consists of n disjoint intervals I_k for $k \in \{0, 1, \ldots, n-1\}$ of different size. In any case, we can define the mesh length of a partition by:

$$\delta_\Pi = \max_{0 \leq k < n} \operatorname{diam}(I_k)$$

one of the topics of interest in approximation theory is to determine the properties of functions that are well approximated by functions in a specific approximation space. In our case, it could be for example to search for the functions u for which $e(u, V_l)_{+\infty} = \mathcal{O}(\delta_{\Omega_l}^\alpha)$ where $0 \leq \alpha$ is a real number. It turns out that when $0 \leq \alpha \leq 1$, we know exactly which functions are characterised by this quality of approximation: it is the Lipschitz space $\operatorname{Lip} \alpha$ [DeV98]. We recall here the definition of this space: for $0 \leq \alpha \leq 1$ and $M > 0$ a real number, we

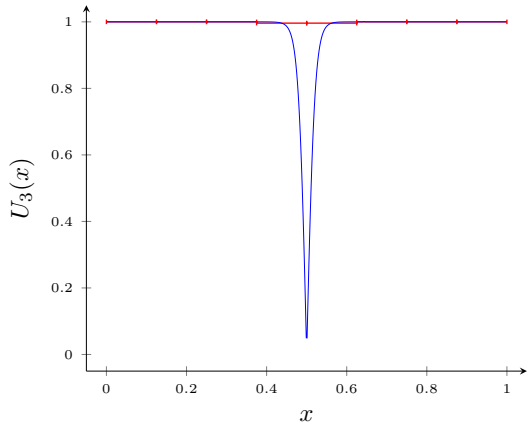


Figure 1.3: Best piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ in V_3. Blue line: exact function; red line: approximate function

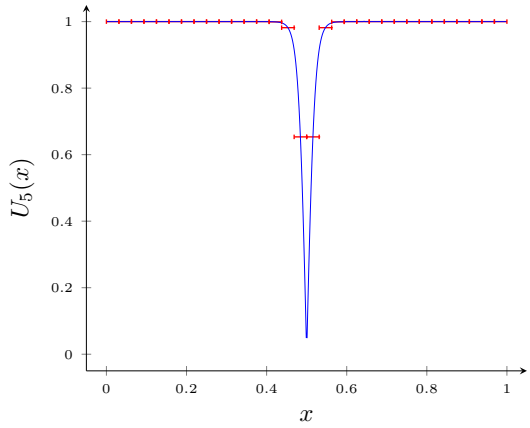


Figure 1.4: Best piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ in V_5. Blue line: exact function; red line: approximate function

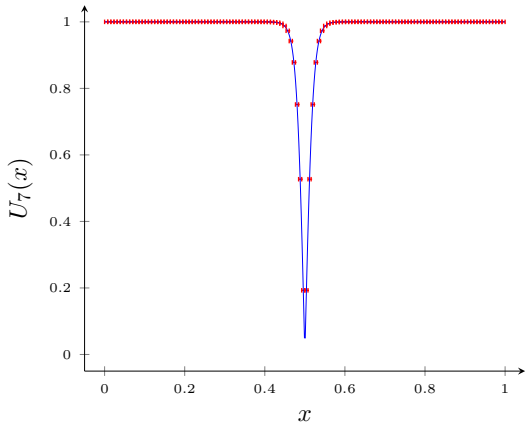


Figure 1.5: Best piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ in V_7. Blue line: exact function; red line: approximate function

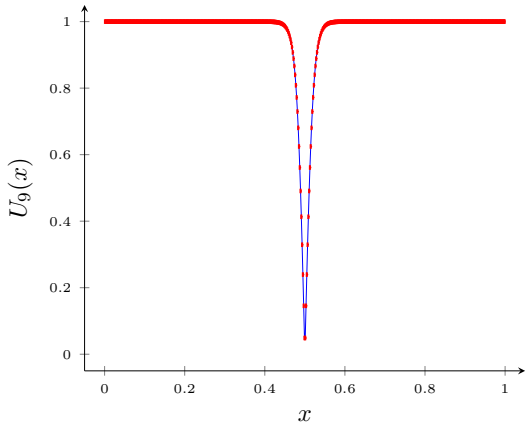


Figure 1.6: Best piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ in V_9. Blue line: exact function; red line: approximate function

define $\mathrm{Lip}_M \alpha$ the set of all functions u on Ω such that:

$$|u(x) - u(y)| \leq M|x - y|^\alpha$$

We define $\mathrm{Lip}\,\alpha$ as the union of the sets $\mathrm{Lip}_M \alpha$ for all $M \in \mathbb{R}_+^*$.
After these considerations on linear approximation we introduce *nonlinear approximation*. Let us consider a partition Π of ω by intervals belonging to the grids Ω_l but without the requirement that all intervals are elements of the same grid. We then denote by $V(\Pi)$ the set of functions that are piecewise constants on each intervals from Π. We then define the space Σ_n to be the union of sets $V(\Pi)$ with $\#\,\Pi = n$ for n a positive integer, where $\#\,\Pi$ is the cardinality of Π. Each function in Σ_n is piecewise constant with at most n different pieces. For $u \in L_p(\Omega)$, $0 < p \leq +\infty$, we can look for the best approximant of u by functions in Σ_n, and as with the linear approximation, we define the approximation of u in the approximation space Σ_n by:

$$\varepsilon_n(u)_p = \inf_{F \in \Sigma_n} \|u - F\|_{L_p} \tag{1.3}$$

This is a nonlinear approximation, because the space Σ_n is *nonlinear*: it is not stable under linear combination operations. But what we lose in simplicity over linear approximation space, is compensated by the flexibility of nonlinear approximants. Indeed, in the case $p = +\infty$, it turns out that $\varepsilon_n(u)_{+\infty} = \mathcal{O}(n^{-1})$ if and only if u is of bounded variation [DeV98]. If we take for example the function $u(x) = x^{\frac{1}{2}}$, it is in $\mathrm{Lip}\,\frac{1}{2}$ and in no higher-order Lipschitz space. If we want to approximate it by functions from the V_l spaces to a precision of $\mathcal{O}(10^{-3})$, then we need a mesh length for Ω_l of order $\mathcal{O}(10^{-6})$ or less, and we have to divide Ω in at least $\mathcal{O}(10^6)$ pieces. Whereas an approximation from Σ_n will require $\mathcal{O}(10^3)$ pieces, because this function is of course of bounded variation. In the case of nonlinear approximation of this type, we speak of *n-term approximation*, because we want to find a good approximation of u by functions characterised by at most n different real coefficients. This difference in approximation quality is due to the fact that in nonlinear approximation, we can design a partition that depends on the target function, which is not the case in linear approximation. Coming back to our function u defined by (1.1), we see with figures (1.3-1.6) that the linear approximation from V_3 is sufficient to represent the target function near the boundaries of the domain, where the function is almost constant, whereas we need an approximation from V_9 to properly represent the function at the center of the domain, where u presents a particularly sharp singularity. Clearly, we can get a better nonlinear approximation by taking elements from V_9 at the center of the domain, and combining them with elements from V_3 near the boundaries.
From this point, the intuitive idea behind the use of nonlinear approximation for adaptive mesh refinement techniques is quite straigthforward: we use piecewise

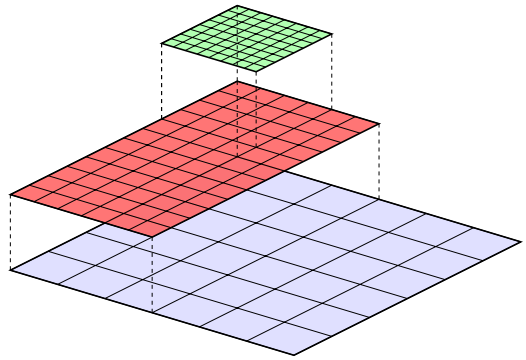


Figure 1.7: Hierarchy of embedded grids in $2D$

constant functions on coarse intervals where the function is smooth enough, and we use piecewise constant functions on finer intervals in the region where the target function is not smooth. We then have to tackle the following questions:

- how can we automatically measure the smoothness of the target function?
- how do we find the most suitable nonlinear approximation depending on the target function?
- how do we measure (and ultimately control) the approximation error?

In the numerical simulation community, the first strategy proposed was that of *Adaptive Mesh Refinement* (AMR) [BO84]. In the following section, we will give a brief account of this technique.

1.1.2 Adaptive Mesh Refinement Techniques

The first AMR techniques were developed in the eighties, and aimed at improving Computational Fluid Dynamics (CFD) numerical simulations. A lot of reactive flows real-world applications are characterised by the presence of highly localized small-scale processes in a smooth larger domain. The goal was then to place finer meshes into the regions with steep spatial gradients, or discontinuities, over a coarse grid covering the remaining of the domain. The general framework for AMR methods was set by Berger & Colella [BC89], by the use of a hierarchy of embedded grids (see figure (1.7) for an example in two dimensions.).
Bell *et al.* [BDS+05] for example showed the high efficiency of this method on structured meshes, for the resolution of viscous incompressible flows. Let us sketch here the main components of this adaptive grid technique, in relation with our initial approximation problem.
We consider a function u over an open bounded domain $\Omega \in \mathbb{R}^d$ with $d = 1$, 2 or 3. We want to build a nonlinear approximation of u by piecewise constant

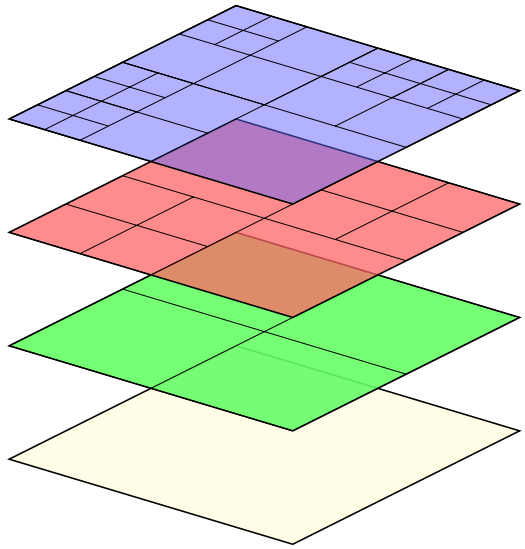


Figure 1.8: Example of cell refinement process illustrating cell-based AMR

functions. We start by dividing Ω into a coarse uniform grid, with logically rectangular meshes (intervals in $1D$, rectangles in $2D$ and parallelepipeds in $3D$). We approach u into each cell by a constant, so as to minimize an error expressed into a given norm (it could be the mean value of u on each mesh for example). Then, we use a local error estimation procedure, that we will detail later, into each cell to determine the quality of the approximation. If we are not satisfied with the given value, we proceed to a *refinement* of the approximation: we divide the cell into equal sized meshes (for example into two identical intervals in $1D$, four identical rectangles in $2D$ and eight identical parallelepipeds in $3D$), and we recompute an approximation of u into these newly created meshes. This cell refinement procedure can be recursively applied to the finer meshes created, by subdividing them into smaller identical cells with a fixed ratio, until we reach, hopefully only into a small proportion of Ω, a maximum user-prescribed level of refinement. The same local error estimation procedure can be used in the opposite direction to determine whether the level of spatial resolution can be loosened, in an operation called *coarsening*: if some neigboring nodes over-resolve u, we can combine them into a coarser mesh (for example combine two neighboring intervals in 1d into two times bigger interval), and we approximate u into this new mesh by the average of the combined meshes approximations. In two dimensions for example, we can end up with grid shapes as in figure (1.8).

Thus, with the *refining* and *coarsening* operations, it is possible to build an adaptive grid. As we said earlier, AMR techniques are generally used to solve

PDEs, and they are coupled with spatial discretization schemes to evaluate spatial derivatives. In a finite volume context for example, these spatial derivatives are approximated by fluxes neighboring cells. Their computation is greatly simplified if neighboring cells do not differ too much in size: that is why most AMR implementations follow a $2:1$ balance criterion [Pop03, BWG11], imposing that the levels of refinement of two neighboring cells differ at most by one unit.

We now come back to the local error estimation procedure. Generally they consist of heuristics to determine the local smoothness of the function. One possibility is to compute a scaled discretized gradient in each cell, and to compare them to a user-prescribed tolerance (see [Dei05]): we refine the mesh if the scaled gradient is above the tolerance. Another error estimation procedure can be based upon a Richardson extrapolation performed using the fine and corresponding coarser local approximation (see *, e.g.,* [Ber82, BO84, BC89]). In both cases, we see that the local error estimation procedure is quite straightforward.

Many AMR softwares have been developed over the past 30 years in order to perform multi-dimensional simulations of physical phenomena. In general, adaptive grids techniques require the use of sophisticated data structures, and elaborate algortihms to deal with mesh connectivity, in order to perform efficient calculations. We will only mention here the quadtrees/octrees data structures, that have been recently implemented for this purpose [Pop03, BWG11, GTG15]. More AMR software libraries can be found in [Dua11] and references therein. The term *octree* refers to a recursive tree structure where each node is either a leaf or has 8 children. This corresponds to three dimension domains, and the analogous in two dimensions is named a *quadtree*, where nodes have four children. They can be associated with $3D$ and $2D$ parallelepipedic domains. The root node corresponds to a parallelepipedic domain that is recursively subdivided according to the tree structure. This tree data structure is well suited to the recursive refinement procedure described above. The HPC, parallel distributed processing software library *p4est* [BWG11] is one of the most efficient implementations of octree data structure for numerical simulations coupled to AMR capabilities. It was recently used in [Dru17, War19] to compute low-mach two phase flows, or magnetic reconnection for partially ionized plasma.

We see clearly now how AMR techniques answer the questions asked at the end of section (1.1). The smoothness of the target function we want to approximate is measured by heuristic computations of the local gradient for example, that we compare to a prescribed tolerance; and the nonlinear approximation is found by the subsequent use of the *refining* and *coarsening* operations, that allocate finer or coarser meshes whithin the domain depending on the local

error approximation procedure. But this strategy lacks rigorous foundations: the local regularity indicators for the target functions are ultimately a simple heuristic parameters, and we cannot evaluate the performance of the nonlinear approximation. Also, we cannot determine the quality of the approximation error, namely between the AMR approximation and a uniform approximation of the function based on the most refined grid. This is precisely the theoretical gap that we wish to fill by using *adaptive multiresolution* strategies, based on the pioneering work of Harten [Har94a, Har95].
In what follows, the theoretical framework of multiresolution techniques will be detailed. We will then describe the space adaptive multiresolution scheme conceived as an adaptive mesh refinement method for time dependent PDEs, in particular, the family of *fully adaptive multiresolution scheme* introduced by Cohen *et al.* in [CKMP03]. Some reviews on such topics can be found in [Har94b, CDD04, Pos05].

1.2 Wavelet theory and multiresolution analysis

Wavelet coefficients offer a simple, and effective approach to decompose a target function into a series of piecewise constant functions. They provide us with local regularity indicators of the function, that translate into the size and scale of the coefficients of the decomposition. The n-term approximation problem is settled then in a rather straigthforward operation: we just have to retain the n terms in the wavelet series that are largest relative to the error measuring norm of the approximation. Before introducing the wavelets used in our adaptive multiresolution strategy, we think it best to start with an exposition of the simplest wavelet decomposition, the *Haar system*.

1.2.1 The Haar system and nonlinear approximation

We come back to the problem introduced at the beginning of this chapter in section (1.1.1). We look for a good approximation of a real function u over the open domain $\Omega =]0,1[$. We consider again the partition of Ω by the dyadic grids Ω_l. Let φ be the characteristic funtion of Ω. Since the dyadic grid are *nested*, the linear spaces V_l form a ladder: $V_l \subset V_{l+1} \subset V_{l+2} \subset \cdots$. Hence we can define the orthogonal complement W_l of V_l in V_{l+1}. W_0 for example is spanned by the following function:

$$H(x) = \chi_{]0,\frac{1}{2}[} - \chi_{]\frac{1}{2},1[} = \begin{cases} 1 & \text{if } 0 < x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} < x < 1 \end{cases} \tag{1.4}$$

H is the *Haar function*. More generally, the space W_l is spanned by the following (normalized) shifted dilates of H:

$$\psi_i^l(x) = 2^{l/2} H(2^l x - i), \qquad i = 0, \dots, 2^l - 1, \tag{1.5}$$

The function ψ_i^l is a scaled version of H, with the scale 2^{-l} on the interval K_i^l. By definition, $V_{l+1} = V_l \oplus^\perp W_l$, so that we can write:

$$V_l = W_{l-1} \oplus^\perp V_{l-1} = W_{l-1} \oplus^\perp W_{l-2} \oplus^\perp V_{l-2} = \cdots, \tag{1.6}$$

and we see that $\varphi \cup \left(\psi_i^k\right)_{k=0,\dots,l-1, i=0,\dots,2^k-1}$ is an orthonormal basis of V_l. We can compare this basis with the more classical orthonormal basis of V_l obtained by normalizing the functions φ_i^l:

$$\phi_i^j(x) = 2^{j/2} \varphi(2^l x - i), \qquad i = 0, \dots, 2^l - 1, \tag{1.7}$$

We will consider the special case where $u \in L_2(\Omega)$. If we denote by P_l the projector in L_2 onto the subspace V_l, then we know that $P_l\, u$ is the best approximation of u by functions from V_l. We can write:

$$P_l u := \sum_{i=0}^{2^l-1} \langle f, \phi_i^l \rangle \phi_i^l \tag{1.8}$$

or alternatively:

$$P_l u = \langle u, \varphi \rangle \varphi + \sum_{k=0}^{l-1} \sum_{i=0}^{2^k-1} \langle f, \psi_i^k \rangle \psi_i^k \tag{1.9}$$

Clearly, $\overline{\cup V_j} = L^2(\Omega)$ (for the canonical norm on L_2), so that $P_l u$ is convergent in $L^2(\Omega)$, *i.e.*

$$\lim_{l \to +\infty} \|u - P_l u\|_{L^2} = 0, \tag{1.10}$$

and we can take the limit in (1.9) to obtain:

$$u = \langle u, \varphi \rangle \varphi + \sum_{k \geq 0} \sum_{i=0}^{2^k-1} \langle u, \psi_i^k \rangle \psi_i^k \tag{1.11}$$

Hence $\varphi \cup \left(\psi_i^k\right)_{k \geq 0, i=0,\dots,2^k-1}$ is an orthonormal basis of $L_2(\Omega)$, called *the Haar basis*.

Now let denote by $d_i^k := \langle u, \psi_i^k \rangle$ the coefficients of u in the Haar basis. Contrary to the classical basis, in the Haar basis the information we add by going from the resolution in V_l to the resolution in V_{l+1} is orthogonal, *i.e.* independent of the information we have in V_l. And going from $P_l u$ to $P_{l+1} u$ is easier: we just have to add the *detail* corresponding to the space W_l, $(d_i^l)_{i=0,\dots,2^l-1}$. d_i^l is the *wavelet coefficient* of u at scale 2^{-l} and position $2^{-l} i$, W_l is known as a *wavelet space*, spanned by the *wavelets* $\left(\psi_i^l\right)_{i=0,\dots,2^k-1}$. The details can also be

interpreted as *local regularity indicators* for the function u. First, we can apply the Parseval's identity to (1.11) to obtain:

$$\|u\|^2_{L_2(\Omega)} = |\langle u, \varphi\rangle|^2 + \sum_{k\geq 0}\sum_{i=0}^{2^k-1} |\langle u, \psi_i^k\rangle|^2 \tag{1.12}$$

which means that the series of terms (d_i^k) converges absolutely; hence the wavelet coefficients tend to zero as l tends to infinity. We can estimate the size of the details with the following heuristic reasoning. Let us assume that u is C^1, then we have:

$$|d_i^l| = \inf_{c\in\mathbb{R}} |\langle f - c, \psi_i^l\rangle| \leq \inf_{c\in\mathbb{R}} \|f - c\|_{L^2(K_i^l)} \leq 2^{-l}\|u'\|_{L^2(K_i^l)}, \tag{1.13}$$

by using a formal Taylor series expansion, and noticing that $\|\psi_i^l\|_{L^2(K_i^l)} = 1$, and that the Haar wavelets ψ_i^l are orthogonal to any constant $c \in \mathbb{R}$, *i.e.*they have *first order vanishing moments*:

$$\langle c, \psi_i^l\rangle = 0. \tag{1.14}$$

The decay of the wavelet coefficients is directly influenced by the local smoothness of u. Consequently, the coefficients d_i^l get small at fine scales when $u|_{K_i^l}$ is sufficiently smooth, whereas high gradients involve more significant values.

We can use all this characteristics of the wavelet decomposition to build an efficient n-term approximation of u. Indeed, we just have to take the n biggest wavelet coefficients; we see from (1.11) that this approximation offers the best n-term approximation of u in $L_2(\Omega)$ by piecewise constants functions. In addition, because the details are local regularity indicators, we are assured that coefficients corresponding to the regions where u is less smooth will be automatically captured by this approximation. Finally, we can easily estimate the error of this approximation by (1.12): it is simply the norm of (the sum of) the details we discard.

The last advantage of the Haar basis that we want to present here is the fact that it is easy to compute the wavelet coefficients. There is indeed the following *two-scales* relations between the canonical basis and the Haar basis:

$$\left.\begin{aligned} \phi_i^l &= \frac{1}{\sqrt{2}}(\phi_{2i}^{l+1} + \phi_{2i+1}^{l+1}), \quad & \psi_i^l &= \frac{1}{\sqrt{2}}(\phi_{2i}^{l+1} - \phi_{2i+1}^{l+1}), \\ \phi_{2i}^{l+1} &= \frac{1}{\sqrt{2}}(\phi_i^l + \psi_i^l), & \phi_{2i+1}^{l+1} &= \frac{1}{\sqrt{2}}(\phi_i^l - \psi_i^l), \end{aligned}\right\} \tag{1.15}$$

which leads to a *change of basis*:

$$\sum_{i=0}^{2^{l+1}-1} \langle u, \phi_i^{l+1}\rangle \phi_i^{l+1} = \sum_{i=0}^{2^l-1} \langle u, \phi_i^l\rangle \phi_i^l + \sum_{i=0}^{2^l-1} \langle u, \psi_i^l\rangle \psi_i^l, \tag{1.16}$$

or equivalently,

$$\sum_{i=0}^{2^{l+1}-1} c_i^{l+1}\phi_i^{l+1} = \sum_{i=0}^{2^l-1} c_i^l \phi_i^l + \sum_{i=0}^{2^l-1} d_i^l \psi_i^l, \tag{1.17}$$

where

$$\left.\begin{aligned} c_i^l &= \frac{1}{\sqrt{2}}(c_{2i}^{l+1} + c_{2i+1}^{l+1}), \quad & d_i^l &= \frac{1}{\sqrt{2}}(c_{2i}^{l+1} - c_{2i+1}^{l+1}), \\ c_{2i}^{l+1} &= \frac{1}{\sqrt{2}}(c_i^l + d_i^l), & c_{2i+1}^{l+1} &= \frac{1}{\sqrt{2}}(c_i^l - d_i^l). \end{aligned}\right\} \tag{1.18}$$

The representation in term of the fine scales can be retrieved from the coarse scale averages by adding the detail, lost through the coarse projection. A recursive change of basis based on these two-scale coefficients (1.18) yields a telescopic transform known as the *fast wavelet transform* $\mathcal{W}$.
As a consequence, for a given $L > l_0$, a function $u_L \in V_L$ can be written either on the standard canonical basis:

$$u_L = \sum_{i=0}^{2^L-1} c_i^L \phi_i^L, \tag{1.19}$$

or on a *wavelet* or *multi-scale basis*:

$$u_L = \sum_{i=0}^{2^{l_0}-1} c_i^{l_0}\phi_i^{l_0} + \sum_{j=l_0}^{L-1}\sum_{i=0}^{2^j-1} d_i^j \psi_i^j, \tag{1.20}$$

The change of representation from (1.19) to (1.20) is performed by the wavelet decomposition, where $\mathcal{W}$ transforms a linear combination of fine scale box functions with an array of coefficients $\mathbf{c}_L$, into a linear combination of coarse scale box functions with coefficient array $\mathbf{c}_0$ and Haar wavelets with array of detail coefficients $\mathbf{d}_l$ for each dyadic level $j < J$:

$$\mathcal{W} : \mathbf{c}_L \to \mathbf{d}^L := (\mathbf{c}_0, \mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{L-1})\,. \tag{1.21}$$

In the same way and based on the same relations (1.18), the inverse transform $\mathcal{W}^{-1} : \mathbf{d}^L \to \mathbf{c}_L$, turns the wavelets coefficients into the single scale L. Due to the telescopic structure of these computations and because the relations (1.18) involve only finite coefficients $(\pm 1/\sqrt{2})$ called usually *masks*, the number of operations required by both transforms is $\mathcal{O}(2^L)$.

With this introduction to the Haar system, we are ready to tackle the modern version of piecewise constant approximations by multiresolution analysis, that rely on the smoother biorthogonal wavelets introduced by Daubechies [Dau88, Dau92], and Cohen *et al.* [CDF92],

1.2.2 Biorthogonal Wavelets

We will introduce the concepts of biorthogonal wavelets in one dimension, for the Hilbert space $L_2(\mathbb{R})$. We begin our discussion by constructing a *multiresolution analysis* of this space. Multiresolution is related to spaces invariant under shift dilation operations. If f is a function defined on $\mathbb{R}$ and $j \in \mathbb{Z}$ is an integer, then $f(\cdot - j)$ is the shift of f by j. And for $a > 0$ a real number, $f(a\cdot)$ is the dilate of f by a. Let φ be a compactly supported function in $L_2(\mathbb{R})$. We define V_0 as the closure in $L_2(\mathbb{R})$ of the set of all finite linear combinations of the shifts of φ. We say that V_0 is the *principal shift invariant (PSI) space* generated by φ. Now for $k \geq 0$ an integer, we define the space $V_k := V_k(\varphi)$ as the dilate of V_0 by 2^k. A function g is in V_k if and only if $g = f(2^k\cdot)$ with $f \in V_0$. Then the space V_k is invariant under the shifts $j2^{-k}$, $j \in \mathbb{Z}$. In other words, the spaces V_k are scaled versions of V_0. We require that the spaces V_k satisfy the *ladder property*, that is for $k \geq 0$ an integer, we have: $V_k \subset V_{k+1} \subset V_{k+2} \subset \cdots$. This is equivalent to requiring that $V_0 \subset V_1$, *i.e.*$\varphi \in V_1$. We also assume that the shifts $\varphi(\cdot - j)$, $j \in \mathbb{Z}$ are a Riesz basis for V_0, that is the shifts $\varphi(\cdot - j)$ span V_0 (that is the case by definition of V_0) and for all $(c_k)_{k\in\mathbb{Z}} \in \ell^2(\mathbb{Z})$, there exist positive constants, $0 < c < C < \infty$, such that

$$c\|c_k\|_{\ell^2}^2 \leq \left\| \sum_{k\in\mathbb{Z}} c_k \varphi(\cdot - k) \right\|_{L^2}^2 \leq C\|c_k\|_{\ell^2}^2, \tag{1.22}$$

and hence there is a unique representation of $f_j \in V_j$ in this basis: $f_j = \sum_{k\in\mathbb{Z}} x_k \phi(\cdot - k)$. Finally, we ask that the union of the V_k spaces is dense in $L_2(\mathbb{R})$, $\overline{\cup V_k} = L^2(\mathbb{R})$. Then $(V_k)_{k\in\mathbb{N}}$ is a *multiresolution analysis* of L_2 [Dau88, Dau92, Mal89]. The Haar system introduced in section (1.2.1) gives us an example of a multiresolution analysis built with this specific design. The function φ is called the *scaling function* in multiresolution theory.

Now we introduce *the dual scaling* function of φ, $\tilde{\varphi}$, and we assume that we are also able to build a multiresolution analysis of L_2 with this function, as in the preceding paragraph. Duality means here that we have:

$$\langle \varphi(\cdot - j), \tilde{\varphi}(\cdot - k) \rangle = \delta_{jk}, \quad (j,k) \in \mathbb{Z}^2. \tag{1.23}$$

with δ_{jk} the Kronecker symbol. $\tilde{\varphi}$ is also assumed to be a compactly supported function in $L_2(\mathbb{R})$. Because we assume that spaces V_k and $\tilde{V}_k$ possess the ladder

property, φ and $\tilde{\varphi}$ are refinable, and we can write them in the following way:

$$\varphi(x) = \sum_{k\in\mathbb{Z}} a_k \varphi(2x-k), \quad \tilde{\varphi}(x) = \sum_{k\in\mathbb{Z}} \tilde{a}_k \tilde{\varphi}(2x-k), \tag{1.24}$$

with finitely supported masks $(a_k)_{k\in\mathbb{Z}}$, $(\tilde{a}_k)_{k\in\mathbb{Z}}$, because the scaling functions are compactly supported in L_2. With this *dual pair* of scaling functions, we can define a non-orthogonal projection

$$Pf := \sum_{k\in\mathbb{Z}} \langle f, \tilde{\varphi}(\cdot - k)\rangle \varphi(\cdot - k) \tag{1.25}$$

onto V_0. By using dilate operations, we obtain the corresponding projectors P_k that map L_2 onto V_k, $k \in \mathbb{N}$. If we denote by $Q := P_1 - P_0$ the projector onto a subspace W of V_1, then W is called a wavelet space. As seen above with the Haar system, the functions in W represent the details we have to add to the resolution in V_0 to obtain the finer resolution in V_1. A fundamental result of wavelet theory is that W is also a PSI space generated by the function:

$$\psi(x) = \sum_{k\in\mathbb{Z}} b_k \tilde{\varphi}(2x-k) \tag{1.26}$$

with $b_k = (-1)^k \tilde{a}_{1-k}$. The shifts $\psi(\cdot - j)$, $j \in \mathbb{Z}$, are also a Riesz basis for W, and their *dual wavelets* take the form $\tilde{\psi}(\cdot - j)$, where $\tilde{\psi}$ is obtained from φ via the formula:

$$\tilde{\psi}(x) = \sum_{k\in\mathbb{Z}} \tilde{b}_k \varphi(2x-k) \tag{1.27}$$

with $\tilde{b}_k = (-1)^k a_{1-k}$. We then have:

$$Qf := \sum_{k\in\mathbb{Z}} \langle f, \tilde{\psi}(\cdot - k)\rangle \psi(\cdot - k) \tag{1.28}$$

By dilation, we obtain the spaces W_j, the projectors Q_j and the representation

$$Q_j f := \sum_{k\in\mathbb{Z}} 2^j \langle f, \tilde{\psi}(\cdot - k)\rangle \psi(\cdot - k) \tag{1.29}$$

Since $\overline{\cup V_k} = L^2(\mathbb{R})$, $P_k f \to f$ as $k \to +\infty$, we have:

$$f = \sum_{k\geq 0} \big(P_{k+1} f - P_k f\big) = \sum_{k\geq 0} \sum_{j\in\mathbb{Z}} 2^k \langle f, \tilde{\psi}(2^k \cdot - j)\rangle \psi(2^k \cdot - j) \tag{1.30}$$

(1.30) is called the *biorthogonal wavelet decomposition* of the function $f \in L_2(\mathbb{R})$ [Dau92, DeV98]. The pairs (φ, ψ) and $(\tilde{\varphi}, \tilde{\psi})$ are usually called the *primal* scaling function and wavelet, and the *dual* scaling function and wavelet, respec-

tively.

One of the main advantages of biorhorgonal wavelets is that they have better approximation properties than the Haar wavelet (1.4). To show this, we will first normalize our biorthogonal wavelet bases, by defining:

$$\varphi_k^j = 2^{j/2}\varphi(2^j \cdot -k), \qquad k \in \mathbb{Z}, \tag{1.31}$$

$$\tilde{\varphi}_k^j = 2^{j/2}\tilde{\varphi}(2^j \cdot -k), \qquad k \in \mathbb{Z}, \tag{1.32}$$

$$\psi_k^j = 2^{j/2}\psi(2^j \cdot -k), \qquad k \in \mathbb{Z}, \tag{1.33}$$

$$\tilde{\psi}_k^j = 2^{j/2}\tilde{\psi}(2^j \cdot -k), \qquad k \in \mathbb{Z}, \tag{1.34}$$

Then we can re-write the decomposition (1.30) as:

$$f = \sum_{k\in\mathbb{Z}} \langle f, \tilde{\varphi}_k^0\rangle \varphi_k^0 + \sum_{j\in\mathbb{N}}\sum_{k\in\mathbb{Z}} \langle f, \tilde{\psi}_k^j\rangle \psi_k^j, \tag{1.35}$$

akin to the Haar decomposition (1.11). The wavelets can be designed to have specific regularity properties; they can be spline functions for example, with an explicit analytical expression given by piecewise polynomials [Dua11], so that they have bounded derivatives over their support domain. In addition, they can be chosen to have *vanishing moments* properties (as in (1.14)). We say that the wavelet ψ_k^j has N vanishing polynomials moments if, for any polynomial $P \in \mathbb{P}_{N-1}$ we have:

$$\langle P, \psi_k^j\rangle_{\Sigma_{j,k}} = 0, \tag{1.36}$$

where $\Sigma_{j,k} := \operatorname{supp}\psi_k^j$. And consequently we obtain approximations of order N [CDD04],

$$\begin{aligned} |\langle f, \psi_k^j\rangle| &= \inf_{P\in\mathbb{P}_{N-1}} |\langle f - P, \psi_k^j\rangle| \\ &\leq \inf_{P\in\mathbb{P}_{N-1}} \|f - P\|_{L^2(\Sigma_{j,k})} \|\psi_k^j\|_{L^2(\Sigma_{j,k})} \\ &\leq C 2^{-jN} |f|_{W_2^N(\Sigma_{j,k})}, \end{aligned} \tag{1.37}$$

using *Cauchy-Schwarz's inequality*, the fact that

$$\|\psi_k^j\|_{L^2(\Sigma_{j,k})} = 1 \tag{1.38}$$

and based on a standard estimate on *local polynomial approximation* (see, *e.g.*, [DS84]):

$$\inf_{P\in\mathbb{P}_n} \|f-P\|_{L^2(\Omega)} \leq C(\operatorname{diam}\Omega)^n |f|_{W_2^n(\Omega)}. \tag{1.39}$$

The semi-norm associated with the *Sobolev space* $W_2^n(\Omega)$:

$$W_2^n(\Omega) := \left\{ f \;\middle|\; \partial^\alpha f \in L^2(\Omega),\, |\alpha| \leq n \right\}, \tag{1.40}$$

is given by $|f|_{W_2^n(\Omega)} := \left(\sum_{|\alpha|=n} \|\partial^\alpha f\|^2_{L^2(\Omega)}\right)^{1/2}$. We also assume that the measure of $\Sigma_{j,k}$ is $\mathcal{O}(2^{-j})$. Hence the details decrease as $j \to +\infty$, and their size is an indicator of the local regularity of the function f. If the supports $\tilde{\Sigma}_{j,k}$ of the dual wavelets do not overlap *too much* [CKMP03], then we can apply the nonlinear approximation strategy developed for the orthogonal Haar system to the decomposition (1.35): the different wavelet components of f give *independent* information of f, and they decay exponentially, and in faster way in regions where f is smoother. Taking the n largest coefficients in (1.35) gives us a best n-term approximation of f. What is more, by the Riesz property of the wavelets, it can be shown that:

$$c \sum_{j=-1}^{\infty} \sum_{k\in\mathbb{Z}} \|\langle f, \tilde{\psi}_k^j\rangle\|^2 \leq \|f\|_{L^2}^2 \leq C \sum_{j=-1}^{\infty} \sum_{k\in\mathbb{Z}} \|\langle f, \tilde{\psi}_k^j\rangle\|^2 \tag{1.41}$$

where $\psi_k^{-1} := \varphi_k^0$ and $\tilde{\psi}_k^{-1} := \tilde{\varphi}_k^0$. We still have a tight relation between the norm of f and the size of the wavelet coefficients, so that we are still able to control the approximation error.
Let us now introduce the adaptive multiresolution strategy.

1.3 Adaptive Multiresolution Strategy

We come back to the approximation of u by *piecewise constant* functions on Ω. For $j = 0, 1, \ldots, J$, from the coarsest to the finest grid, we build regular disjoint partitions (cells) $(\Omega_\gamma)_{\gamma\in S_j}$ of an open subset $\Omega \subset \mathbb{R}^d$, such that each Ω_γ, $\gamma \in S_j$, is the union of a finite number of cells Ω_μ, $\mu \in S_{j+1}$, and thus S_j and S_{j+1} are consecutive embedded grids. The index j refers thus to the scale level and we denote

$$|\gamma| := j \quad \text{if} \quad \gamma \in S_j, \tag{1.42}$$

with the abbreviated notation $\Omega_\gamma := \Omega_{j,k}$, where $k \in \mathbb{Z}^d$. For instance, we can consider the univariate dyadic intervals in 1D, $d = 1$:

$$\Omega_\gamma = \Omega_{j,k} := [2^{-j}k, 2^{-j}(k+1)),\ \gamma \in S_j := \{\, (j,k) \mid j \in \mathbb{N}_0,\, k \in \mathbb{Z} \,\}. \tag{1.43}$$

The same follows for higher dimensions.

1.3.1 Multiresolution Analysis

We denote $\mathbf{U}_j := (u_\gamma)_{\gamma \in S_j}$ as the spatial representation of u on the grid S_j, where u_γ represents the cell-average of $u : \mathbb{R}^d \to \mathbb{R}$ in Ω_γ:

$$u_\gamma := |\Omega_\gamma|^{-1} \int_{\Omega_\gamma} u(\mathrm{x})\, \mathrm{dx}, \quad \mathrm{x} \in \mathbb{R}^d. \tag{1.44}$$

Data at different levels of discretization are related by two inter-level transformations which are defined as follows:

1. The *projection operator* P_{j-1}^{j}, which maps $\mathbf{U}_j$ to $\mathbf{U}_{j-1}$. It is obtained through exact averages computed at the finer level by

$$u_\gamma = |\Omega_\gamma|^{-1} \sum_{\substack{|\mu|=|\gamma|+1 \\ \Omega_\mu \subset \Omega_\gamma}} |\Omega_\mu| u_\mu. \tag{1.45}$$

   since the grids are nested, this projection operator is *exact* and *unique* [Pos05].

2. The *prediction operator* P_j^{j-1}, which maps $\mathbf{U}_{j-1}$ to an approximation $\hat{\mathbf{U}}_j$ of $\mathbf{U}_j$. We want it to satisfy at least two basic constraints [CKMP03]:

   (a) The prediction is local, *i.e.*$\hat{u}_\mu$ depends on the values u_γ on a finite stencil R_μ surrounding Ω_μ, where $|\mu| = |\gamma| + 1$.

   (b) The prediction is *consistent* with the projection in the sense that

$$P_{j-1}^{j} \circ P_j^{j-1} = \mathrm{Id}. \tag{1.46}$$

   i.e., one can retrieve the coarse cell averages from the predicted values:

$$u_\gamma = |\Omega_\gamma|^{-1} \sum_{\substack{|\mu|=|\gamma|+1 \\ \Omega_\mu \subset \Omega_\gamma}} |\Omega_\mu| \hat{u}_\mu; \tag{1.47}$$

   In particular, this property implies that the stencil R_μ must contain the unique index γ such that $|\mu| = |\gamma| + 1$ and $\Omega_\mu \subset \Omega_\gamma$.

With these operators, we define for each cell Ω_μ the *prediction error* or *detail* as the difference between the exact and predicted values:

$$d_\mu := u_\mu - \hat{u}_\mu, \tag{1.48}$$

or in terms of inter-level operations:

$$d_\mu = u_\mu - P_{|\mu|}^{|\mu|-1} \circ P_{|\mu|-1}^{|\mu|} u_\mu. \tag{1.49}$$

The consistency assumption (1.47) and the definitions of the projection operator (1.45) and of the *detail* (1.48), imply

$$\sum_{\substack{|\mu|=|\gamma|+1 \\ \Omega_\mu \subset \Omega_\gamma}} |\Omega_\mu| d_\mu = 0. \tag{1.50}$$

We can then construct as shown in [CKMP03], a *detail vector* defined as $\mathbf{D}_j = (d_\mu)_{\mu \in \nabla_j}$, where the set $\nabla_j \subset S_j$ is obtained by removing for each $\gamma \in S_{j-1}$, one $\mu \in S_j$ such that $\Omega_\mu \subset \Omega_\gamma$, in order to avoid redundancy from expressions (1.48) and (1.47), and to get a one-to-one correspondence:

$$\mathbf{U}_j \longleftrightarrow (\mathbf{U}_{j-1}, \mathbf{D}_j), \tag{1.51}$$

issued by operators P_{j-1}^j and P_j^{j-1}. For instance, in the univariate dyadic case (1.43) the detail vector is given by $\mathbf{D}_j = (d_{j,k})_{k \in \mathbb{Z}}$ with $d_{j,k} = u_{j,k} - \hat{u}_{j,k}$. By iteration of this decomposition, we finally obtain a *multi-scale representation* of $\mathbf{U}_J$ in terms of $\mathbf{M}_J = (\mathbf{U}_0, \mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_J)$:

$$\mathcal{M} : \mathbf{U}_J \longmapsto \mathbf{M}_J, \tag{1.52}$$

and similarly, its inverse $\mathcal{M}^{-1}$.

1.3.2 Wavelet Representation

We will restrict to the case where P_j^{j-1} is linear, because we then have

$$\hat{u}_\mu := \sum_\gamma c_{\mu,\gamma} u_\gamma, \tag{1.53}$$

which means that $\mathcal{M}$ and $\mathcal{M}^{-1}$ are *changes of basis*. Based on the theoretical studies exposed in section 1.2, we can then identify a wavelet representation [CKMP03] by defining for $\mathbf{U}_j$ the *dual scaling wavelet* $\tilde{\varphi}_\gamma$ in (5.1):

$$u_\gamma := \langle u, \tilde{\varphi}_\gamma \rangle, \tag{1.54}$$

such that

$$\tilde{\varphi}_\gamma := |\Omega_\gamma|^{-1} \chi_{\Omega_\gamma}, \tag{1.55}$$

and where according to (1.53)

$$d_\mu := u_\mu - \hat{u}_\mu = \langle u, \tilde{\varphi}_\mu \rangle - \sum_\gamma c_{\mu,\gamma} \langle u, \tilde{\varphi}_\gamma \rangle = \langle u, \tilde{\psi}_\mu \rangle, \tag{1.56}$$

defines the *dual wavelet* $\tilde{\psi}_\mu$:

$$\tilde{\psi}_\mu := \tilde{\varphi}_\mu - \sum_\gamma c_{\mu,\gamma}\tilde{\varphi}_\gamma. \tag{1.57}$$

The *multiresolution representation* $\mathbf{M}_J$ can be then written as

$$\mathbf{M}_J = (d_\lambda)_{\lambda\in\nabla^J} = (\langle u, \tilde{\psi}_\lambda\rangle)_{\lambda\in\nabla^J}, \tag{1.58}$$

which corresponds exactly to $\mathbf{d}^J$ in the definition of the wavelet transform (1.21), where we have defined $\nabla^J := \bigcup_{j=0}^J \nabla_j$ with $\nabla_0 := S_0$, and where $d_\lambda = u_\lambda$ and $\tilde{\psi}_\lambda = \tilde{\varphi}_\lambda$ if $\lambda \in \nabla_0$. With this representation, the multiresolution representation will be of order N if for all $u \in \mathbb{P}_{N-1}$ and for all $\lambda \in \nabla^J$, we have

$$\langle u, \tilde{\psi}_\lambda\rangle = d_\lambda = 0, \tag{1.59}$$

that is, if the wavelet $\tilde{\psi}_\lambda$ has N vanishing moments.
Next we build a prediction operator such that the associated dual wavelet is of order N. A standard procedure defines P_j^{j-1} based on polynomial interpolations of order N. For instance, for the univariate dyadic case (1.43), considering a centered stencil $(u_{j,k-M},\dots,u_{j,k+M})$ and the unique polynomial of degree $2M$ such that

$$2^j \int_{\Omega_{j,l}} p_{j,k}(x)\,\mathrm{d}x = u_{j,l}, \quad l = k-M,\dots,k+M, \tag{1.60}$$

we can define the prediction approximation taking into account the consistency property (1.47) [CKMP03]:

$$\hat{u}_{j+1,2k} = 2^{j+1}\int_{\Omega_{j+1,2k}} p_{j,k}(x)\,\mathrm{d}x, \qquad \hat{u}_{j+1,2k+1} = 2^{j+1}\int_{\Omega_{j+1,2k+1}} p_{j,k}(x)\,\mathrm{d}x. \tag{1.61}$$

This procedure is exact for polynomials of degree $2M$, *i.e.* it has accuracy order $N = 2M+1$.
For a 1D configuration, we can define an *interpolation stencil* $R_{j,k}$[1] (1.62):

$$R_{j,k} = \{\,(j-1, \lfloor k/2\rfloor + l) \mid |l| \le M\,\}, \tag{1.62}$$

to approximate the values at grid level $j+1$: $\hat{u}_{j+1,2k}$ and $\hat{u}_{j+1,2k+1}$, contains the parent-cell $u_{j,k}$ and its nearest M neighbors. The centered polynomial interpolations of accuracy order $N = 2M+1$ might be written for the 1D case

[1]Symbol $\lfloor\ \rfloor$ denotes the floor function, which maps a real number to the largest integer smaller than or equal to the given real number.

as

$$\begin{cases} \hat{u}_{j+1,2k_1} = u_{j,k_1} + \displaystyle\sum_{d_1=1}^{M} \xi_{d_1} \left(u_{j,k_1+d_1} - u_{j,k_1-d_1} \right), \\ \hat{u}_{j+1,2k_1+1} = u_{j,k_1} - \displaystyle\sum_{d_1=1}^{M} \xi_{d_1} \left(u_{j,k_1+d_1} - u_{j,k_1-d_1} \right), \end{cases} \tag{1.63}$$

where $k = k_1 \in \mathbb{Z}$, and the coefficients ξ_{d_1} are given in Table 1.1 up to $M = 4$. The case $N = 3$ is given by (1.64):

$$\hat{u}_{j+1,2k} = u_{j,k} + \frac{1}{8}(u_{j,k-1} - u_{j,k+1}), \qquad \hat{u}_{j+1,2k+1} = u_{j,k} + \frac{1}{8}(u_{j,k+1} - u_{j,k-1}). \tag{1.64}$$

N	M	ξ_1	ξ_2	ξ_3	ξ_4
1	0	0	0	0	0
3	1	$-1/8$	0	0	0
5	2	$-22/128$	$3/128$	0	0
7	3	$-201/1024$	$11/256$	$-5/1024$	0
9	4	$-3461/16384$	$949/16384$	$-185/16384$	$35/32768$

Table 1.1: Prediction operator. Coefficients for polynomial interpolations of order $N = 2M + 1$ [Har94a].

The case $N = 1$, $M = 0$ corresponds to the Haar wavelets introduced in section 1.2.1.

Extensions to multi-dimensional interpolations is straightforward based on the 1D configuration (1.63). Defining the expression Q^M as

$$Q^M(k_1, u_{j,k}) = \sum_{d_1=1}^{M} \xi_{d_1} \left(u_{j,k_1+d_1} - u_{j,k_1-d_1} \right), \tag{1.65}$$

the 2D polynomial interpolation, proposed by Bihari & Harten [BH96], reads

$$\begin{aligned} \hat{u}_{j+1,(2k_1+p,2k_2+q)} = u_{j,(k_1,k_2)} + (-1)^p Q^M(k_1, u_{j,(\cdot,k_2)}) + (-1)^q Q^M(k_2, u_{j,(k_1,\cdot)}) \\ - (-1)^{(p+q)} Q_2^M(k_1, k_2, u_{j,(k_1,k_2)}), \end{aligned} \tag{1.66}$$

The integers p and q are equal to either 0 or 1 depending on the child-cell considered, and Q^M (1.65) is used in both dimensions. The operator Q_2^M, derived from a tensor product is given by

$$Q_2^M(k_1,k_2,u_{j,(k_1,k_2)}) = \sum_{d_1=1}^{M}\xi_{d_1}\sum_{d_2=1}^{M}\xi_{d_2}(u_{j,k_1+d_1,k_2+d_2} - u_{j,k_1-d_1,k_2+d_2} - u_{j,k_1+d_1,k_2-d_2} + u_{j,k_1-d_1,k_2-d_2}). \quad (1.67)$$

In the same way, 3D interpolations are defined by introducing the operator Q_3^M:

$$\begin{aligned} Q_3^M(k_1,k_2,k_3,u_{j,(k_1,k_2,k_3)}) = \sum_{d_1=1}^{M}\xi_{d_1}\sum_{d_2=1}^{M}\xi_{d_2}\sum_{d_3=1}^{M}\xi_{d_3}(&u_{j,k_1+d_1,k_2+d_2,k_3+d_3} \\ &- u_{j,k_1-d_1,k_2+d_2,k_3+d_3} - u_{j,k_1+d_1,k_2-d_2,k_3+d_3} \\ &- u_{j,k_1+d_1,k_2+d_2,k_3-d_3} + u_{j,k_1-d_1,k_2-d_2,k_3+d_3} \\ &+ u_{j,k_1-d_1,k_2+d_2,k_3-d_3} + u_{j,k_1+d_1,k_2-d_2,k_3-d_3} \\ &- u_{j,k_1-d_1,k_2-d_2,k_3-d_3}). \end{aligned} \quad (1.68)$$

Hence,

$$\begin{aligned} \hat{u}_{j+1,(2k_1+p,2k_2+q,2k_3+r)} = u_{j,(k_1,k_2,k_3)} &+ (-1)^p Q^M(k_1,u_{j,(\cdot,k_2,k_3)}) + (-1)^q Q^M(k_2,u_{j,(k_1,\cdot,k_3)}) \\ &+ (-1)^r Q^M(k_3,u_{j,(k_1,k_2,\cdot)}) \\ &- (-1)^{(p+q)} Q_2^M(k_1,k_2,u_{j,(\cdot,\cdot,k_3)}) \\ &- (-1)^{(p+r)} Q_2^M(k_1,k_3,u_{j,(\cdot,k_2,\cdot)}) \\ &- (-1)^{(q+r)} Q_2^M(k_2,k_3,u_{j,(k_1,\cdot,\cdot)}) \\ &+ (-1)^{(p+q+r)} Q_3^M(k_1,k_2,k_3,u_{j,(k_1,k_2,k_3)}), \end{aligned} \quad (1.69)$$

As before, p, q, and r are equal to either 0 or 1.

1.3.3 Data Compression and Tree-Structured Data

One of the main interests of carrying out such a multi-scale decomposition is that this new representation (1.58), defines a whole set of regularity estimates all over the spatial domain, and thus a data compression might be achieved. Given a set of index $\Lambda \subset \nabla^J$, we define a truncation operator $\mathcal{T}_\Lambda$, that leaves unchanged the component d_λ if $\lambda \in \Lambda$, and replaces it by 0, otherwise. In

practice, we are interested in sets Λ obtained by *thresholding*:

$$\lambda \in \Lambda \quad \text{if} \quad |d_\lambda| \geq \varepsilon_{|\lambda|}, \tag{1.70}$$

with the level-dependent threshold values $(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_J)$. Data compression is then achieved by discarding the cells whose *details* are not into Λ according to (1.70). Applying $\mathcal{T}_\Lambda$ on the multi-scale decomposition $\mathbf{M}_J$ of $\mathbf{U}_J$ amounts to building an approximation $\mathcal{A}_\Lambda \mathbf{U}_J$, where the operator $\mathcal{A}_\Lambda$ is given by

$$\mathcal{A}_\Lambda := \mathcal{M}^{-1} \mathcal{T}_\Lambda \mathcal{M}. \tag{1.71}$$

Taking into account that

$$u = \sum_{j \in \mathbb{N}_0} \sum_{|\lambda|=j} \langle u, \tilde{\psi}_\lambda \rangle \psi_\lambda, \tag{1.72}$$

it can be seen that for a given J, the array $\mathbf{\Psi}_{J,\lambda}$ with $|\lambda| \leq J$, corresponds to the cell averages of the primal wavelet ψ_λ at level J, *i.e.*$\mathbf{\Psi}_{J,\lambda} = (\langle \psi_\lambda, \tilde{\phi}_\gamma \rangle)_{\gamma \in S_J}$. We can thus define the normalized norm ℓ^1 by

$$\|\mathbf{U}_J\| := 2^{-dJ} \sum_{\lambda \in S_J} |u_\lambda|, \tag{1.73}$$

which corresponds to the L^1-norm of a piecewise constant function. For $\mathbf{\Psi}_{J,\lambda}$, this yields

$$\|\mathbf{\Psi}_{J,\lambda}\| \leq C \|\psi_\lambda\|_{L^1} \leq C 2^{-d|\lambda|}. \tag{1.74}$$

And for the thresholded representation of $\mathbf{U}_J$ after applying $\mathcal{A}_\Lambda$ [CKMP03]:

$$\|\mathbf{U}_J - \mathcal{A}_\Lambda \mathbf{U}_J\| = \left\| \sum_{\lambda \neq \Lambda} d_\lambda \mathbf{\Psi}_{J,\lambda} \right\| \leq C \sum_{\lambda \neq \Lambda} |d_\lambda| 2^{-d|\lambda|} = C \sum_{|d_\lambda| \leq \varepsilon_{|\lambda|}} |d_\lambda| 2^{-d|\lambda|}, \tag{1.75}$$

where we see that the approximation error is bounded by the sum of the discarded details. Taking into account that $|d_\lambda| 2^{-d|\lambda|} \leq \varepsilon_{|\lambda|} 2^{-d|\lambda|}$, and considering a level-wise threshold parameter:

$$\varepsilon_j := 2^{dj} \eta, \tag{1.76}$$

the next bound follows[2]

$$\|\mathbf{U}_J - \mathcal{A}_\Lambda \mathbf{U}_J\| \leq C \, \#(\nabla^J) \eta = C \, \#(S_J) \eta \leq C 2^{dJ} \eta, \tag{1.77}$$

with the cautious assumption that all the d_λ such that $\lambda \notin \Lambda$, are equal to

[2] $\#(\,)$ denotes the cardinality of a set.

$\varepsilon_{|\lambda|}$, although many of them might be much smaller. The latter estimate (1.77) justifies the choice $\eta = 2^{-dJ}\varepsilon$ in order to have

$$\|\mathbf{U}_J - \mathcal{A}_\Lambda \mathbf{U}_J\| \leq C\varepsilon, \tag{1.78}$$

with the level-dependent threshold values proposed by Harten [Har94a, Har95]:

$$\varepsilon_j = 2^{d(j-J)}\varepsilon, \quad j \in [0, J], \tag{1.79}$$

where ε becomes the threshold value for the finest level J.
However, we do not delete all useless details at this point because we want to preserve the ability to perform the computations of the prediction operator. The set Λ must conserve a *graded tree* structure in order to guarantee the availability of cell values within the local prediction stencil (1.62, 1.63). In order to define such a structure, we first introduce the following terminology:

- If $\Omega_\mu \subset \Omega_\lambda$ with $|\mu| = |\lambda| + 1$, we say that Ω_μ is a *child* of Ω_γ, and that Ω_γ is the *parent* of Ω_μ.
- By the definition of ∇_j, if Ω_λ has $N(\Omega_\lambda)$ children, $N(\Omega_\lambda) - 1$ of them are in $\nabla := \bigcup_{j\geq 0} \nabla_j$. We call these cells the *detail children* of Ω_λ.
- Moreover, we define the *leaves* $\mathrm{L}(\Lambda)$ of a *tree* Λ as the set of Ω_λ with $\lambda \in \mathrm{L}(\Lambda)$ such that Ω_λ has no children in Λ.
- Finally, we define Ω_λ as a *root* when it belongs to the coarsest grid, that is, $\lambda \in S_0$ or $|\lambda| = 0$, in which case, we denote λ as λ_0.

A set of indices $\Lambda \in \nabla$ is a tree if the following holds [CKMP03]:

- The fundamental level $\nabla_0 = S_0$ is contained in Λ.
- If Ω_μ and Ω_υ are detail children of the same Ω_λ, then $\mu \in \Lambda$ if $\upsilon \in \Lambda$.
- If Ω_λ is such that its detail children are in Λ, then the parent of Ω_λ has the same property.

For the 1D dyadic configuration (1.43), Λ is a tree if $\nabla_0 \in \Lambda$ and

$$(j, k) \in \Lambda \Rightarrow (j - 1, \lfloor k/2 \rfloor) \in \Lambda. \tag{1.80}$$

The set $R(\Lambda)$ contains the tree Λ plus the missing cells Ω_λ in the construction of ∇_j. A tree Λ is thus graded if for all $\mu \in R(\Lambda)$, the prediction stencil R_μ is contained in $R(\Lambda)$. Coming back to the dyadic example, Λ is a graded tree if

$$(j, k) \in R(\Lambda) \Rightarrow (j - 1, \lfloor k/2 + l \rfloor) \in R(\Lambda), \quad |l| \leq M. \tag{1.81}$$

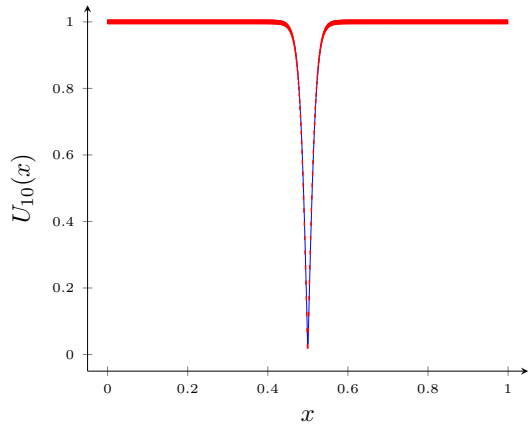


Figure 1.9: Best piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ in V_{10}. Blue line: exact function; red line: approximate function

Defining Λ_ε as the smallest graded tree containing Λ given by (1.70), we introduce the corresponding tree approximation operator $\mathcal{A}_\varepsilon := \mathcal{A}_{\Lambda_\varepsilon} = \mathcal{M}^{-1}\mathcal{T}_{\Lambda_\varepsilon}\mathcal{M}$, following (1.71). Since $\Lambda \subset \Lambda_\varepsilon$, it follows directly that

$$\|\mathbf{U}_J - \mathcal{A}_\varepsilon \mathbf{U}_J\| \leq C\varepsilon. \tag{1.82}$$

1.3.4 Numerical example

We end this chapter with a numerical example of the adaptive multiresolution strategy applied to the function u (1.1) introduced at the beginning of section (1.1.1). We will detail in Chapter 5 the algorithms and the main issues concerning the practical implementation of the multiresolution schemes we just described. Here, we will just give the parameters we used for the nonlinear approximation of function (1.1) by multiresolution.
We chose as the maximum grid level the value $J = 10$, meaning that we have 1024 meshes on the finest grid. The (best) linear approximation in V_{10} can be seen in figure (1.9).
We use the predictor with $N = 3$ vanishing moments (1.64), and the thresholding parameter is set at $\varepsilon = 10^{-5}$. We can see in figure (1.10) the results of this procedure.
We obtain exactly what we expected; an approximation of u on a grid adapted to its smoothness, with the finest meshes at the center of the domain, and *much* coarser meshes near the boundaries. And qualitatively, we see that this approximation is a very good representation of the function. The reduction in the number of meshes used can be measured with the compression rate τ. We define it as:

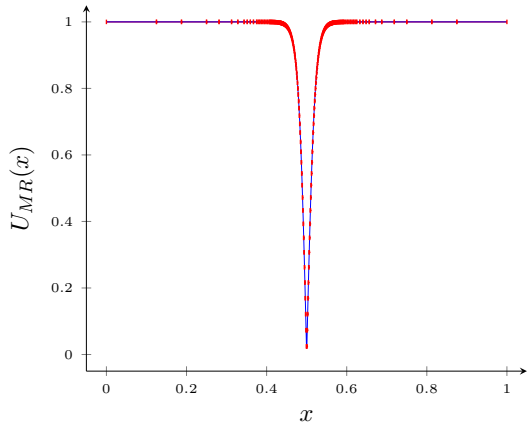


Figure 1.10: Piecewise constant approximation in the L_2-norm of $u(x) = \tanh\left(50 \cdot \left|x - \frac{1}{2}\right|\right)$ obtained by adaptive multiresolution, with $J = 10$ and $\varepsilon = 10^{-5}$. Blue line: exact function; red line: approximate function

$$\tau = 1 - \frac{\#(L(\Lambda))}{2^J} \tag{1.83}$$

In this case we obtain the impressive value: $\tau = 80.27\%$, showcasing the high efficiency of the multiresolution strategy to effectively build adaptive grids.

Chapter 2

A new collocated finite-volume scheme for the incompressible Navier Stokes equations on adaptive multriesolution grids

In the previous chapter we have considered the specific implementation of the adaptive multiresolution technique that we wish to apply to incompressible flows. We are now concerned with the spatial discretization of the incompressible Navier-Stokes equations on the particular set of non-uniform meshes generated by the use of this adaptive strategy. The elliptic nature of this set of equations makes it more difficult to provide accurate finite volume methods for their resolution, and we will start this chapter with a small overview of some finite volume schemes devised in recent years to solve incompressible flows on general non-matching grids, with a focus on methods coupled to AMR. In the second part of this chapter, we describe our new collocated scheme.

2.1 Overview of finite volume schemes

The incompressible fluid flow considered is fully described by two variables, the velocity vector $\boldsymbol{u} = (u_i(x, y, z, t))_{i=1,\dots,d}$ and the pressure field $p(x, y, z, t)$. The time variable t varies between 0 and T. The flow momentum and mass balance equations read:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}^t \otimes \boldsymbol{u}) + \nabla p - \nu \, \Delta \boldsymbol{u} = \boldsymbol{f} \quad \text{in} \quad \Omega \times]0, T[\\ \nabla \cdot \boldsymbol{u} = 0 \quad \text{in} \quad \Omega \times]0, T[\end{cases} \tag{2.1}$$

with a homogeneous Dirichlet boundary condition for $\boldsymbol{u}$ and the initial condi-

tion:

$$\boldsymbol{u}(\,\cdot\,,0) = \boldsymbol{u}_{\text{ini}} \quad \text{in} \quad \Omega$$

where ν is the cinematic viscosity of the fluid and $\boldsymbol{f}$ is a source term. We make the following assumptions:

1. T is the finite duration of the flow,
2. $\nu \in]0, +\infty[$,
3. $\boldsymbol{u}_{\text{ini}} \in L^2(\Omega)^d$
4. $\boldsymbol{f} \in L^2(\Omega \times]0, T[)^d$.

It follows from the implicit nature of the pressure variable p in (2.1) and the algebraic constraint stating that the velocity must be divergence-free that the velocity lies in the kernel of the continuous divergence, and that the pressure lies in the orthogonal of the gradient operator which happens to be the image of the divergence operator [GS00]. Their discrete counterparts should follow these properties and lie in the kernel of the *discrete* divergence operator for the velocity, and the orthogonal of the kernel of the *discrete* gradient for the pressure. But it can occur that, depending on the spatial discretization, the discrete gradient operator has more kernel modes than present in the continuous case, thus producing *spurious* pressure modes that pollute the approximate solution one wishes to obtain [GS00, RC83, SLI10].
The classical approach to avoid this problem is to discretize the equation on *staggered* grids [HW65]. This strategy has become the standard finite volume scheme for incompressible flows, and has served as the basis of many engineering applications thanks to its simplicity and efficiency [Pat80, KM85]. What is more, the strong mathematical properties of this scheme allowed convergence proofs for the Stokes [Nic92, CEGH14, CK98] and Navier-Stokes [NW96, CEGH14, GHLM16] equations, on rectangular Cartesian grids. Nonetheless, the extension of this scheme to complex geometries is not obvious at all [ZSK94]. Various solutions have been proposed to apply this MAC scheme in the context of grid adaptation with AMR techniques [ABC+98, PHB+98, Pop03, GTG15, HB97, MC00, MG06]. In [ABC+98] Almgren *et al.* devised a method to solve the time-dependent, variable density incompressible Navier-Stokes equations on a hierarchy of grids. Their spatial discretization was based on a single-grid (meaning a uniform Cartesian grid) scheme (first designed by Almgren, Bell and Szymczak [ABS96]), where the velocity variables were cell-centered; at each timestep, they first extrapolate face-centered velocities from the cell-centered ones, that they make divergence-free with an exact projection method using MAC-like spatial derivatives. They use these face-centered velocities to compute advective terms, and then compute the new cell-centered velocities with an approximate projection method, where the discrete Laplacian operators for the pressure Poisson equation is an approximate one, easier to invert

than the real discrete divergence-gradient operator. Since the computational domain is composed of patches of grids with different refinement levels, they can apply the uniform-grid scheme on each one of these grids, and then use a special treament to deal with the coarse-fine boundaries. Martin [MC00] later used a similar strategy to solve incompressible flows with a hierarchical structured grid AMR approach [BO84].
Later, Popinet [Pop03] developed an incompressible Euler equations solver coupled with a quad/octree implementation of AMR. The main difference with the aforementioned AMR schemes being that instead of using a single-grid approach, he made spatial derivatives suited to deal locally with coarse-fine interfaces. These two approaches both suffer from at least the two following issues: (i) the discrete velocity field is not divergence-free (even though in their schemes the face-centered discrete velocity fields used for advection are divergence-free) and this is a cause of kinetic energy conservation error [MLVM98], and (ii) the treatment of coarse-fine interfaces is rather complex. Finally, in [GTG15] Guittet *et al.* also solved the incompressible Navier-Stokes equations on adaptive non-graded quad/octrees, with a MAC layout of the velocity and pressure field. Here, special interpolations are needed after every refining/coarsening operations, to preserve the MAC layout of the data, introducing additional spatial errors.

In our case, the multiresolution algorithms require inter-level operations between hierarchically embedded grids and result in non-uniform adaptive grids, as seen in Chapter 1, and we wish to avoid the computational overhead necessary to maintain different staggered grids throughout the whole computation. Thus we discretize (2.1) on *collocated* grids and we need to deal with the spurious modes.
The main strategies derived over the years to suppress spurious pressure modes on collocated grids have resorted to the addition of a dissipative term to the original set of equations ([RC83]). The method designed by Rhie and Chow [RC83] for example modifies the discrete Laplacian of the pressure Poisson equation obtained in a typical prediction-projection scheme to solve (2.1). In [FL06], Felten and Lund showed that with this correction, the cell-center discrete velocities are not strictly solenoidal, and hence cause a kinetic energy conservation error in the inviscid limit. We wish to avoid this uncontrollable accuracy loss.
In light of these observations, Shashank *et al.*[SLI10] came up with a subtle solution: they discretize (2.1) in a uniform Cartesian grid with the standard second-order accurate central difference finite volume approximation, and then build the discrete Laplacian operator of the pressure Poisson equation. They identify the spurious pressure modes, and filter the polluted pressure field. This solution conserves the discrete mass, momentum and kinetic energy in the in-

viscid limit [SLI10]. We will follow these guidelines when dealing with the spurious modes in our collocated scheme.

All the finite volume schemes presented above share the same design principle based on the *classical* interpretation of (2.1): (i) they first divide the computational domain Ω in finite number of control volumes; then (ii) they integrate the momentum and balance equations of (2.1) over each control volume, and turn these volume integrals into surface integrals over the boundary of the control volume thanks to the divergence theorem; finally (iii) the continuous variables $\boldsymbol{u}$ and p are approximated by constants in the control volume, the fluxes integrals are approximated by interpolations of the constant values in each control volume, or by a finite difference approach through the use of Taylor expansions. This viewpoint presents several advantages, one of the most important being the fact that it is fairly easy to understand and implement for CFD practitioners. But it comes in our opinion with two major drawbacks:

1. Since it is based on the classical interpretation of the PDE at hands, it implicitly make assumptions on the initial data, or the regularity of the solutions; for example the presence of the Laplace operator in (2.1) implies that the velocity possesses second-order spatial derivatives, when we know [Tem77] that a broad range of velocity solutions of (2.1) do not possess such smoothness

2. The use of finite difference restricts its application to simple geometries or mesh shapes

This situation led some researchers to develop new guideline principles for the design of finite volume schemes [CET06, CEH09, EGH10, EGH00, CEGH14]. These new methods rely on the variational formulation of (2.1).
Various aspects of this philosophy were presented in [EGH00], where the authors successfully applied it to discretize elliptic, hyperbolic and parabolic problems in one, two or three dimensions, proving convergence of the approximate solutions to the continuous PDE when the characteristic size of the mesh tends to zero. In [CET06, CEH09], colocated finite volume schemes were designed to solve the incompressible Navier-Stokes equations on general $2D$ or $3D$ conforming polygonal meshes, meaning that the computational domain Ω is discretized into general polygonal convex control volumes, with the only requirement that the straight line joining the centers of two adjacent control volumes is perpendicular to their common edge. Then in [CEH09], the preceding scheme was expanded to deal with general non-conforming grids, which is a feature shared by the adaptive grids produced by the multiresolution strategy introduced in Chapter 1.
We followed the design principles of [CEH09] to build a colocated finite volume scheme adapted to our grids, where we made sure that the spurious pressure

modes do not affect the computation of the velocity field; if needed, we filter the non-physical kernel modes in the pressure, but only once, at the end of the computation [GS00, SLI10]. The following section is devoted to the presentation of this finite volume method.

2.2 A new spatial discretization

We consider a rectangular (*resp.* rectangular parallelepiped) domain $\Omega =]0, b_x[\times]0, b_y[$ in $2D$ (*resp.* $\Omega =]0, b_x[\times]0, b_y[\times]0, b_z[$ in 3D, with $(b_x, b_y, b_z) \in \mathbb{R}_+^*$). We denote $\partial\Omega = \overline{\Omega}\backslash\Omega$ its boundary. We will treat the case where the meshes K_γ^l of the multiresolution strategy also have a rectangular shape. Hence we define, for $l \in \mathbb{N}^*$, the uniform discretization of Ω as follows:

$$\Omega_l = \{\,]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\mid i, j \in \{0, 1, \ldots, 2^l - 1\}\,\}$$
$$K_{i,j}^l =]2^{-l}i, 2^{-l}(i+1)[\times]2^{-l}j, 2^{-l}(j+1)[$$

in $2D$, and:

$$\Omega_l = \{\,]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\times]2^{-l}b_z k, 2^{-l}b_z(k+1)[\mid i, j, k \in \{0, 1, \ldots, 2^l - 1\}\}$$
$$K_{i,j,k}^l =]2^{-l}i, 2^{-l}(i+1)[\times]2^{-l}j, 2^{-l}(j+1)[\times]2^{-l}b_z k, 2^{-l}b_z(k+1)[$$

in $3D$, where the K_γ^l are identical meshes of Ω_l, the uniform discretization of Ω at level l. We then define a space discretization of Ω:

Definition 2.2.1 (Space discretization)**.** *A discretization $\mathcal{D}$ of Ω is defined as the triplet $\mathcal{D} = (\mathcal{M}, \mathcal{E}, \mathcal{P})$, where:*

1. *$\mathcal{M}$ is a finite family of nonempty connected open disjoint subsets of Ω such that $\overline{\Omega} = \overline{\bigsqcup_{K\in\mathcal{M}} K}$. There exists $\mathbb{L} \in \mathbb{N}^*$ such that every mesh $K \in \mathcal{M}$ belongs to a uniform discretization Ω_l, with $l \leq \mathbb{L}$. Thus every mesh $K \in \mathcal{M}$ has the form K_γ^l defined above, and we denote by l_K its level. For any $K \in \mathcal{M}$, we denote by ∂K its boundary, $m(K)$ its measure and h_K its diameter. We define the size $h_\mathcal{D}$ of $\mathcal{D}$ as: $h_\mathcal{D} = \max\{\, h_K \mid K \in \mathcal{M}\,\}$*

2. *$\mathcal{E}$ is a finite family of disjoint subsets of $\overline{\Omega}$ such that, for all $\sigma \in \mathcal{E}$, σ is a nonempty open subset of a hyperplane of $\mathbb{R}^d$. We assume that for all $K \in \mathcal{M}$, there exists a subset $\mathcal{E}_K$ of $\mathcal{E}$ such that $\partial K = \bigcup_{\sigma\in\mathcal{E}_K} \overline{\sigma}$. For any $\sigma \in \mathcal{E}$, we denote by $\mathcal{M}_\sigma = \{K \in \mathcal{M}, \sigma \in \mathcal{E}_K\}$. We assume that for all $\sigma \in \mathcal{E}$, either $\mathcal{M}_\sigma$ has exactly one element and then $\sigma \subset \partial\Omega$ (the set of these interfaces, called boundary interfaces, is denoted by $\mathcal{E}_{\text{ext}}$) or*

$\mathcal{M}_\sigma$ has exactly two elements (the set of these interfaces, called interior interfaces, is denoted by $\mathcal{E}_{\text{int}}$). In this latter case, if K and L are the meshes in $\mathcal{M}_\sigma$, σ is further specified as $\sigma_{K|L}$. For all $\sigma \in \mathcal{E}$, we denote by $m(\sigma)$ its (Lebesgue) measure. For all $K \in \mathcal{M}$ and $\sigma \in \mathcal{E}_K$, we denote by $\boldsymbol{n}_{K,\sigma}$ the unit vector normal to σ outward to K

3. *$\mathcal{P}$ is the set of $\boldsymbol{x}_K$ points, where for all $K \in \mathcal{M}$, $\boldsymbol{x}_K$ is the center of K. We denote by $d_{K,\sigma}$ the Euclidian distance between $\boldsymbol{x}_K$ and σ, for $\sigma \in \mathcal{E}_K$, and $D_{K,\sigma}$ the cone with vertex $\boldsymbol{x}_K$ and basis σ*

4. *For all $K \in \mathcal{M}$, we denote by $\mathcal{N}_K$ its set of neighbours, i.e. the cells $L \in \mathcal{M}$ such that the intersection of $\mathcal{E}_K$ and $\mathcal{E}_L$ contains exactly one edge. For all $K \in \mathcal{M}$, the following must hold: for any mesh $L \in \mathcal{N}_K$, L's and K's levels differ by at most one unit ($l_L \in l_K - 1, l_K$ in the general case, or $l_L \in l_K - 1, l_K, l_K + 1$ if $l_K \neq \mathbb{L}$)*

Remark 1. *The adaptive grids that we can obtain with the multiresolution strategy described in Chapter 1 are included into the set of discretizations $\mathcal{D}$ described above.*

Given a discretization $\mathcal{D} = (\mathcal{M}, \mathcal{E}, \mathcal{P})$, let us denote by $\#(\mathcal{D})$ the cardinal of $\mathcal{M}$, so that we can introduce:

$$X_\mathcal{D} = \mathbb{R}^{\#(\mathcal{D})} = \{\, v = (v_K)_{K\in\mathcal{M}} \mid v_K \in \mathbb{R} \,\} \tag{2.2}$$

We start by designing a *discrete gradient* on $X_\mathcal{D}$. Let $\boldsymbol{e}_i$ for $i \in \{1, \dots, d\}$ be the basis vectors of $\mathbb{R}^d$. Let $v \in X_\mathcal{D}$, $K \in \mathcal{M}$ and σ so that $K \in \mathcal{M}_\sigma$. We define $\nabla_{K,\sigma} v$ as follows:

1. If $\sigma \in \mathcal{E}_{\text{ext}}$ and $\boldsymbol{n}_{K,\sigma} = \boldsymbol{e}_i$:

$$\nabla_{K,\sigma} v = \frac{1}{2} m(\sigma) v_K \boldsymbol{n}_{K,\sigma}$$

2. If $\sigma \in \mathcal{E}_{\text{ext}}$ and $\boldsymbol{n}_{K,\sigma} = -\boldsymbol{e}_i$:

$$\nabla_{K,\sigma} v = -\frac{1}{2} m(\sigma) v_K \boldsymbol{n}_{K,\sigma}$$

3. If $\sigma \in \mathcal{E}_{\text{int}}$, we denote by L the neigbour of K in $\mathcal{M}_\sigma$.

$$\nabla_{K,\sigma} v = \frac{1}{2} m(\sigma)(v_L - v_K) \boldsymbol{n}_{K,\sigma}$$

Next, for $i \in \{1, \dots, d\}$, we define $\nabla_K^{(i)} v$, as follows:

$$\nabla_K^{(i)} v = \frac{1}{m(K)} \sum_{\sigma \in \mathcal{E}_K} \nabla_{K,\sigma} v \cdot \boldsymbol{e}_i \tag{2.3}$$

We only presented here the discrete gradient with Dirichlet boundary conditions on the velocity, but we also devised a similar operator for Neumann boundary conditions on the velocity.

2.3 Approximation of the Navier-Stokes problem

We come back to the initial problem described in section 2.1, that is, finding approximate solutions to (2.1) on multiresolution adapted grids.
Let $\mathcal{D}$ be a discretization of Ω in the sense of definition 2.2.1. Let $\boldsymbol{U}(t)$ and $P(t)$ be vector fields on $X_{\mathcal{D}}^d$ and $X_{\mathcal{D}}$ respectively, for each $t \in]0,T[$. We will say that $\left[\boldsymbol{U}(t) = (u_{K,i}(t))_{i=1,\dots,d,K\in\mathcal{M}}, P(t) = (p_K(t))_{K\in\mathcal{M}}\right]$ is a semi-discrete approximate solution to (2.1) if we have:

$$\begin{cases} m(K)\dfrac{\partial u_{K,i}}{\partial t} - \nu \displaystyle\sum_{\substack{\sigma\in\mathcal{E}_{\text{int}} \\ \mathcal{M}_\sigma=(K,L)}} \frac{m(\sigma)}{d_\sigma}(u_{L,i}-u_{K,i}) + \sum_{\substack{\sigma\in\mathcal{E}_{\text{ext}} \\ \mathcal{M}_\sigma=K}} \frac{m(\sigma)}{d_\sigma} u_{K,i} \\ \quad + \displaystyle\sum_{j=1}^{d} \nabla_K^{(j)}(u_{K,i}u_{K,j}) + \sum_{\substack{\sigma\in\mathcal{E}_{\text{int}} \\ \boldsymbol{n}_{K,\sigma}=\pm\boldsymbol{e}_i \\ \mathcal{M}_\sigma=(K,L)}} \frac{1}{2} m(\sigma)(p_K+p_L)\boldsymbol{n}_{K,\sigma}\cdot\boldsymbol{e}_i \\ \quad - \displaystyle\sum_{\substack{\sigma\in\mathcal{E}_{\text{ext}} \\ \boldsymbol{n}_{K,\sigma}=\pm\boldsymbol{e}_i \\ \mathcal{M}_\sigma=(K)}} \frac{1}{2} m(\sigma) p_K \\ \quad = \displaystyle\int_K f_i \quad \text{for each} \quad i = 1,\dots,d \\ \text{and} \\ \displaystyle\sum_{j=1}^{d} \nabla_K^{(j)} u_{K,j} = 0 \end{cases} \tag{2.4}$$

for all $K \in \mathcal{M}$.
The different equations (2.4) amount to a nonlinear system that can be written in matrix form (for example the different divergence constraints by mesh can be written in a more compact form $D \cdot \boldsymbol{U} = 0$ where D is a *divergence* matrix). Our discretization of the pressure gradient amounts to formally defining the *gradient* matrix as $-D^t$, i.e. the discrete gradient is the negative of the adjoint of the discrete divergence [CET06, CEH09, GTG15]. This way, we make sure

that our discretization mimics this property of the continuous PDE. In addition, we do not then have to specify boundary conditions for the pressure, which can be a tricky operation [GS87].
We re-write (2.4) in the following way which is more easily recognizable as a finite-volume scheme:

$$\begin{cases} m(K)\dfrac{\mathrm{d}u_{K,i}}{\mathrm{d}t} + \underbrace{\nu \sum_{\sigma\in\mathcal{E}_K} F_{K,\sigma}(u_i)}_{\text{Diffusion}} + \underbrace{m(K)\partial_K^{(i)} P}_{\text{Pressure gradient}} + \underbrace{C_K^{(i)}(\boldsymbol{U})}_{\text{Convection}} \\ \quad = \displaystyle\int_K f_i(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = m(K)F_{K,i} \\ \displaystyle\sum_{L\in\mathcal{N}_K} \Phi_{K|L}(\boldsymbol{U}) = 0 \end{cases} \tag{2.5}$$

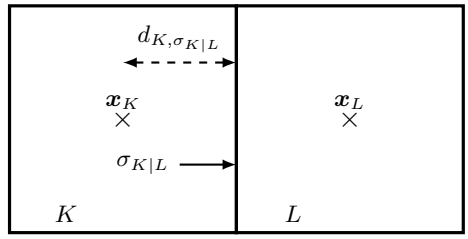
where $\nu\sum_{\sigma\in\mathcal{E}_K} F_{K,\sigma}(U_i)$ are approximations of the diffusive fluxes of the quantity u_i through the set of boundaries $\mathcal{E}_K$ of K. $\sum_{L\in\mathcal{N}_K}\Phi_{K|L}(\boldsymbol{U})$ are an approximation of the mass fluxes through the boundaries of K, with $\mathcal{N}_K$ its set of neighbors. We only need to distinguish between 3 cases for the fluxes computation: the case where the meshes K and L are at the same level ($l_K = l_L$), the case where L has level $l_L = l_K + 1$, and finally the case where L has level $l_L = l_K - 1$. The three cases are explicited in (figure 2.1).

We now bring the attention of the reader to two weaknesses of our scheme:

- the first one is related to the computation of the diffusion fluxes between cells of different sizes. The computation of the fluxes cannot be consistent at these non-conforming edges, and this should affect the precision of the numerical approximation of the solutions to the Navier-Stokes equations on our non-uniform grids. This problem was already adressed in [EGH00], where they observed that "the error which results from this lack of consistency can be controlled if the number of atypical edges is not too large". Indeed, we did not observe such a lack of precision in most of our simulations (results Chapter 6); we believe that this is due to the fact that in general, with the adaptation by multiresolution, we have few non-conforming edges between meshes of different sizes compared to the edges between same-size grids

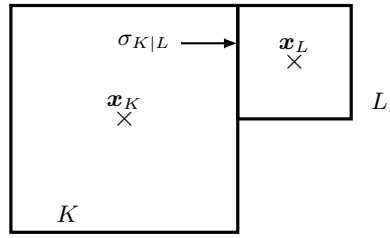
- The second one comes from the fact that our discrete divergence (and hence our discrete pressure gradient) cannot prevent the apparition of spurious pressure modes. We designed a strategy (see Chapter 6) to preclude the apparition of spurious velocity modes; but the spurious pressure modes can nonetheless affect the accuracy of the approximate velocity variables. A proven technique to suppress these pressure modes relies on

Case $l_L = l_K$



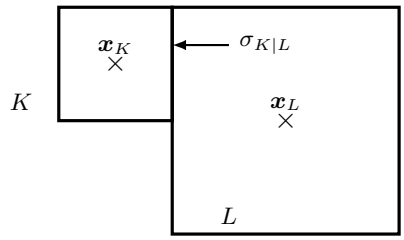
$$F_{k,\sigma_{K|L}}(U_i) = m(\sigma_{K|L})\frac{u_{K,i} - u_{L,i}}{d_{K,\sigma_{K|L}} + d_{L,\sigma_{K|L}}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{K,1} + u_{L,1}}{2}$$

Case $l_L = l_K + 1$



$$F_{k,\sigma_{K|L}}(U_i) = m(\sigma_{K|L})\frac{u_{K,i} - u_{L,i}}{\frac{d_{K,\sigma_{K|L}}}{2} + d_{L,\sigma_{K|L}}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{K,1} + u_{L,1}}{2}$$

Case $l_L = l_K - 1$



$$F_{k,\sigma_{K|L}}(U_i) = m(\sigma_{K|L})\frac{u_{K,i} - u_{L,i}}{d_{K,\sigma_{K|L}} + \frac{d_{L,\sigma_{K|L}}}{2}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{K,1} + u_{L,1}}{2}$$

Figure 2.1: Computation of the fluxes depending on the interface case

a nonconsistent stabilization technique à la Brezzi-Pitkäranta [BP84]. A next step on our work would be to design and implement such a technique for adaptive multiresolution applied to the resolution of the incompressible Navier-Stokes equations.

Chapter 3

Numerical schemes to solve the semi-discretized incompressible Navier-Stokes equations: considerations on the projection methods

In this chapter we consider the numerical integration of the semi-discretized equations obtained after the spatial discretization scheme described in Chapter 2. These are Differential Algebraic Equations, mainly due to the presence of the divergence-free constraint in the Navier-Stokes equations, and they are notoriously harder to solve than classical ODEs. We begin this chapter with an introduction to the mathematical theory behind DAEs, where we will focus on their *stiffness*, and the phenomenon of *order reduction*. We will then give an overview of the recent advances in projection methods, emphasizing their shortcomings to produce high-order numerical solutions for incompressible flows.

3.1 Differential algebraic equations

After the spatial discretization, we now have to solve the following differential equations in the time parameter for the velocity and the pressure variables:

$$\begin{cases} \Gamma \dfrac{dU_i}{dt} = \nu L_i U_i + D_i^t P - \left(\displaystyle\sum_{j=1,\dots,d} D_j U_i U_j \right) + \Gamma S_i(t) \\ \displaystyle\sum_{i=1,\dots,d} D_i U_i = S_{\text{div}}(t) \end{cases} \tag{3.1}$$

Here Γ, L_i and D_i are square matrices of size $\#(\mathcal{M}) \times \#(\mathcal{M})$. Γ is the diagonal mass matrix, so that $\Gamma_{i(K),i(K)} = m(K)$ for the mesh $K \in \mathcal{M}$. The L_i operators are the Laplacian matrices for the diffuse terms (these matrices are symmetric by construction), and the D_i are the divergence matrices described above. The vectors $S_i(t)$ include the discretized source terms F_i and the possible boundary conditions, and the vector $S_{\text{div}}(t)$ includes the boundary conditions for the divergence constraint.
The $(U_i)_{i=1,\dots,d}$ and P variables are vectors that now only depend on the time parameter, and the appropriate mathematical theory to study the numerical (and analytical) solutions of the set of equations (3.1) is that of *differential algebraic equations*. DAEs are ordinary differential equations constrained by algebraic equations. In our case, the discretized momentum equations are ODEs, and the $(U_i)_{i=1,\dots,d}$ and P variables are subjected to the discretized divergence-free velocity constraint. Understanding that there are key differences between ODEs and DAEs is crucial to be able to properly solve the latter. Even though the pursuit of theortical and numerical solutions to ODEs started almost 400 years ago (with the work of Newton, namely), the research of techniques and methods to effectively solve DAEs really gained traction in the last 40 years or so. DAEs generally entail much more numerical difficulties to be solved than ODEs, and some of the properties of numerical schemes, when applied to the latter, are lost when applied to the former. In fact, one of the first and most important papers on this subject is entitled: "Differential Algebraic Equations are not ODEs" [Pet82], and describes, among other things, classes of DAEs problems that cannot even be solved by numerical strategies designed for stiff ODEs. We shall thus start with an introduction to the DAE theory, with a special focus on the relationship between (stiff) ODEs and (semi-explicit) DAEs, which proved to be very useful in our search for effective high-order integration schemes for the (discretized) incompressible Navier-Stokes equations.
The most general form for DAEs that we will be considering in this section is the following [Pet82, HW96]:

$$F(u', u, x) = 0 \tag{3.2}$$

Where $u : \mathbb{R} \to \mathbb{R}^n$ is a vector field in the x parameter, $F : \mathbb{R}^n \to \mathbb{R}^n$ is a (sufficiently differentiable) function with same dimension as u, and of course u' is the derivative of u. We also require that the jacobian matrix $\partial F/\partial u'$ be nonsingular for all argument values in some appropriate domain. (3.2) can be viewed as an *implicit* ODE, and in principle we should be able to formally express u' in terms of x and u, to obtain an explicit ODE of the form:

$$u' = f(x, u) \tag{3.3}$$

In practice however, this transformation is not easy to obtain, and even when

it is, it can be completely useless for the numerical resolution of (3.2). This transformation generally is conducted via (subsequent) differentiations of F with regard to x, and it gives us insights regarding the mathematical structure of a DAE, by its *differentiation index* [GP82, LP86a, LP86b]:

Definition 3.1.1 (Differentiation index of a differential algebraic equation)**.** *Equation (3.2) has* differentiation index m *if* m *is the minimal number of analytical differentiations*

$$\begin{aligned} F(u', u, x) &= 0 \\ \frac{dF(u', u, x)}{dx} &= 0 \\ \frac{d^2F(u', u, x)}{dx^2} &= 0 \\ &\vdots \\ \frac{d^mF(u', u, x)}{dx^m} &= 0 \end{aligned}$$

required to solve u' *uniquely in terms of* u *and* x*, that is, to extract by algebraic manipulations an explicit ODE system of the form (3.3) from (3.2).*

Before giving some useful remarks regarding the index of a DAE, we give some concrete examples of index 1 and index 2 systems, on the important special case of *semi-explicit, Hessenberg* DAEs, which are explicit ODEs systems with constraints: here (3.2) can be expressed as a system of ODEs of the form (3.3) for a part of the components of u', coupled with a system of algebraic constraints involving part or all of the components of u only. These cases are easier to consider than the more general case (3.2), for the following reasons, among others:

1. For a large class of problems described by ODEs of the form (3.3), provided with initial values, there exist theorems guaranteeing the existence and uniqueness of solutions. This is not the case for general DAEs of the form (3.2), but for some of the semi-explicit DAEs, we can have such theorems

2. In the case of semi-explicit DAEs, we can explicitly separate the components of u between *differential variables* and *algebraic variables* [AP98]. The differential variables u_i are the ones for which the original equation (3.2) contain explicit expressions for u_i' of the form $u_i' = f(u, x)$, and the algebraic variables are the remaining ones. Generally, differential and algebraic variables have different mathematical natures, hence they require different numerical treatments. Having them explicitly identified in a DAE system simplifies its numerical integration

3. For semi-explicit DAEs, obtaining the explicit ODEs for the algebraic variables is easily done by differentiating the algebraic constraints

Hessenberg systems of index 1
These are systems of the form:

$$u' = f(u, v) \tag{3.4a}$$

$$0 = g(u, v) \tag{3.4b}$$

Where $u : \mathbb{R} \to \mathbb{R}^n$ and $v : \mathbb{R} \to \mathbb{R}^m$ are vector fields in the x parameter. We assume that g_v is nonsingular for all values of x. By using the implicit function theorem, we can solve for v in (3.4b), and by substituting v in (3.4a), we obtain an ODE in u; we can then apply classical results on ODEs for the existence of solutions to (3.4). Furthermore, by differentiating (3.4b), we obtain the implicit ODE of v':

$$v' = -g_v^{-1}(u, v)g_u(u, v)f(u, v) \tag{3.4c}$$

where we make use of the assumption that g_v is invertible. This problem is thus of differentiation index 1. We remark that if this system is provided with initial conditions, it cannot have solutions unless these initial conditions are *consistent*, i.e. they satisfy the algebraic constraint (3.4b):

$$g(u_0, v_0) = 0$$

Hessenberg systems of index 2
These are systems of the form:

$$u' = f(u, v) \tag{3.5a}$$

$$0 = g(u) \tag{3.5b}$$

Where $u : \mathbb{R} \to \mathbb{R}^n$ and $v : \mathbb{R} \to \mathbb{R}^m$ are vector fields in the x parameter. Differentiating (3.5b), we obtain the following *hidden constraint* on u and v:

$$0 = g_u(u)f(u, v) \tag{3.5c}$$

Here, we will assume that $g_u(u)f_v(u, v)$ is nonsingular for all x. In that case, we see that the system (3.5a), (3.5c) is an index 1 Hessenberg DAE. Differentiating again (3.5c) gives us the missing differential equation for v, namely:

$$v' = -(g_u f_v)^{-1}\left(g_{uu}f^2 + g_u f_u f\right) \tag{3.5d}$$

This problem has thus a differentiation index of 2. We remark that if this system is provided with initial conditions, it cannot have solutions unless these initial conditions are *consistent*, i.e. they satisfy the algebraic constraints (3.5b): $g(u_0) = 0$, *and* (3.5c): $g_u(u_0)f(u_0, v_0) = 0$. In this case, and only in this case,

the system (3.5a, 3.5b) has a locally unique solution. Hence, contrary to the index 1 systems, index 2 systems have hidden constraints that must be satisfied by the solution vector fields for all x. We also note that by differentiating the explicit algebraic constraint, we transformed the index 2 problem into an index 1 system of equations. This technique is referred to as *index reduction* [HW96], and since index 1 problems are generally easier to solve than index 2 DAEs, it could be interesting, in some cases, to solve the implicit index 1 problem instead of the original one, hoping for this set of solutions to be the good solutions to the initial problem.
We now proceed with some general remarks reagarding DAEs, before turning to the interpretation of equations (3.1) as semi-explicit DAEs:

Remark 2. *1. The index of a DAE is a measure of its singularity [LP86a, LP86b]. The higher the index, the more difficult the corresponding DAE is to solve*

2. DAEs with an index greater than 1 *are called* higher-index *DAEs [AP98]. They include some hidden constraints, which are derivatives of the explicitly stated constraints*

3. The algebraic variables are less smooth than the differential variables by one derivative [AP98]. In the case of the incompressible Navier-Stokes equations for example, the pressure is the algebraic variable (see below), and it behaves like the derivative of the velocity. This is one of the reasons why, as we will see later, if a numerical scheme applied to these equations is k-th order for the velocity, it will be $k-1$-th order for the pressure

4. Specified initial conditions for a DAE must be consistent with its algebraic constraints, otherwise this initial value problem has no *solution. This situation is even more restrictive for higher-index systems, because their hidden constraints must also be satisfied by the initial conditions. The same holds for boundary conditions*

5. Due to the preceding remark, it often happens that finding the consistent initial and/or boundary values for a specific DAE in applications is one of the most laborious parts of the numerical resolution [GS00]

6. As stated before, for higher-index systems, it is often possible to apply the index reduction strategy, and solve the lowered-index system instead of the initial one. But this strategy comes at a cost:

(a) The solutions to the lowered-index system are not submitted to the same constraints as the higher-index ones; this entails that solutions to the lowered-index system might not be solutions to the higher-index DAE

(b) Resorting to lowered-index systems can require more regularity for the variables than what is needed by the higher-index system

7. *"The most desired solution is that satisfying the original DAE system, that of highest index. This is because the implication is 'one-way': solutions of the highest index system will always satisfy all (derived) lower index systems." [GS00]*

Coming back to the discretized Navier-Stokes equations, we can derive the divergence-free constraint with regard to the time parameter to obtain (with periodic boundary conditions):

$$\sum_{i=1,\dots,d} D_i U_i' = 0 \tag{3.6}$$

Our mass matrix Γ is diagonal, so that we can easily invert it to obtain, for all $i \in \{1, \dots, d\}$

$$U_i' = \Gamma^{-1}\left(\nu L_i U_i + D_i^t P - \Big(\sum_{j=1,\dots,d} D_j U_i U_j\Big) + \Gamma S_i(t)\right)$$

We insert this equation into (3.6) to obtain the *pressure Poisson equation* (PPE):

$$\Big(\sum_{i=1,\dots,d} D_i \Gamma^{-1} D_i^t\Big) P = -\sum_{i=1,\dots,d} D_i \Gamma^{-1}\left(\nu L_i U_i - \Big(\sum_{j=1,\dots,d} D_j U_i U_j\Big) + \Gamma S_i(t)\right) \tag{3.7}$$

Deriving again (3.7) gives us the missing differential equation for P, so that equations (3.1) are an index-2 Hessenberg system of DAEs. Let's denote by K the matrix $\big(\sum_{i=1,\dots,d} D_i \Gamma^{-1} D_i^t\big)$. K is the $g_u(u) f_v(u, v)$ that appeared above when we introduced index-2 Hessenberg systems, and it is always (at least with our spatial discretization) symmetric positive. The best situation is when it is also *definite*, which corresponds to the condition "$g_u(u) f_v(u, v)$ nonsingular". But in a lot of applications, and depending generally on the boundary conditions, K will be singular, and this will generate a lot of numerical difficulties. We will come back to this later, but for the purpose of this section, we will assume that K is an invertible matrix.

The PPE coupled to the momentum equations form an index-1 Hessenberg system, that is generally referred to in the literature as the PPE formulation of the incompressible Navier-Stokes equations. We start our analysis of this formulation by noting that if $(U_i)_{i=1,\dots,d}$ and P are solutions to the PPE formulation, then only the acceleration is necessarily divergence-free: we only have (3.6) satisfied. Therefore solutions to the PPE formulation may not have a velocity that is divergence-free, so that they are not solutions to (3.1). We can be even more precise if we consider the case of non-consistent initial values $(U_{i,0})_{i=1,\dots,d}$. If

$\sum_{i=1,\dots,d} D_i U_{i,0} \neq 0$, then (3.1) is ill-posed and has no solution. But this is not the case for the PPE formulation, which can be solved. By integrating (3.6), we see that for all $t > 0$, will have:

$$\sum_{i=1,\dots,d} D_i U_i = \sum_{i=1,\dots,d} D_i U_{i,0}$$

"The initial divergence error will linger forever" [GS00]. This is bad for our multiresolution adaptive strategy. If we start with an initial divergence-free velocity field on a given grid, the application of the adaptive strategy will generally results in a new velocity field that is not divergence-free to machine accuracy (but rather, to the accuracy of our adaptive strategy; we will come back to this claim later). If we then solve, on the new adapted grid, the PPE formulation, we see that we will never be able to directly obtain a divergence-free velocity to machine accuracy. Given this, and the preceding remarks on DAEs, we believe that it is not a good strategy to try and solve the PPE formulation when integrating the Navier-Stokes equations coupled to the multiresolution strategy.
We conclude these introductory considerations on DAEs with a useful relationship between DAEs and *stiff* ODEs. Semi-explicit Hessenberg DAEs can be viewed as limits of stiff ODEs in the case of infinite stiffness [HW96, AP98, GS00, HLR89, HW99], and a rich source of DAEs is provided by limit systems of singular value perturbation ODEs [AP98]. We can illustrate this in the case of Hessenber index-1 systems of the form (3.4) with the following ODE:

$$u' = f(u, v) \tag{3.8a}$$

$$\varepsilon v' = g(u, v) \tag{3.8b}$$

If ε in (3.8b) is very small, this parameter will introduce stiffness in the ODE (3.8). In this case (3.8) is referred to as a singular perturbation problem. We also note that letting $\varepsilon \to 0$, we recover (3.4), which is referred to as the *reduced* equation for (3.8). We infer from this that a sound understanding of stiff ODEs is *necessary* in order to construct good numerical algorithms for DAEs. Thus we will make a short digression, and give some useful insights regarding stiff ODEs, before concluding this section with an example of a very simple Hessenberg index-2 DAE, showing the kind of difficulties that can occur when attempting to solve numerically these kind of problems.

3.1.1 Stiff Ordinary Differential Equations

A pragmatic characterisation of stiff problems can be found in one of the seminal works on this subject [CH52]: *stiff problems are problems which are exceedingly difficult to solve with explicit (numerical integration) methods.* We will not dig any further in the definition of stiff ODEs (there is a vast literature treating

stiffness, and we refer the interested reader to the chapter *IV* of the excellent book by Hairer and Wanner on this subject [HW96], but go directly to some examples. The first one is taken from [Dua11].
Let us consider for $t > 0$, the scalar following initial value problem:

$$\left.\begin{aligned} \mathrm{d}_t u &= -100\, u, \\ u(0) &= u_0, \end{aligned}\right\} \tag{3.9}$$

with some $u_0 \in \mathbb{R}$ and $u : \mathbb{R} \to \mathbb{R}$, and exact solution $u(t) = \mathrm{e}^{-100\,t}\, u_0$ for $t > 0$. We aim at obtaining a numerical approximation u_n of the exact solution $u(t_n)$ of (3.9) for a time discretization given by $t_0 = 0 < t_1 < \cdots < t_n < \cdots$, and $n = 0, 1, \ldots$. We first approximate the solution of (3.9) at some $t_1 = t_0 + \delta t$

$$u(\delta t) = u_0 + \int_{t_0}^{t_0+\delta t} -100\, u \, \mathrm{d}t, \tag{3.10}$$

by

$$u_1 = u_0 - 100\, \delta t u_0. \tag{3.11}$$

which implies an explicit time discretization solution of (3.10) and it is known as the *explicit Euler* method, where δt is defined as the integration time step. If we set, for instance, an initial condition $u_0 = 1$, and a relatively small time step of $\delta t = 0.5$ compared with 100, the exact and numerical solutions give, respectively, $u(0.5) = \mathrm{e}^{-50} \approx 1.9 \times 10^{-22}$ and $u_1 = -49$. And integrating over another time step δt: $u(1) = \mathrm{e}^{-100} \approx 3.7 \times 10^{-44}$ and $u_2 = 2401$. It follows then that the explicit time discretization given by (3.11) is not capable of reproducing the right dynamics given by the exact solution. However, since this solution models a rapid transition from u_0 towards a final equilibrium value, we can easily identify the associated time scale $\tau = 1/100 = 0.01$ of the transient phase and therefore, we can expect that integration time steps δt of the order or smaller than τ will be capable to track the right dynamics. For instance, for $\delta t = 0.001$, we have $u(0.001) = \mathrm{e}^{-0.1} \approx 0.904837418$ and $u_1 = 0.9$, and $u(0.002) = \mathrm{e}^{-0.2} \approx 0.818730753$ and $u_2 = 0.81$. These rapid variations or transients associated with fast scales are typical of stiff equations, but they are neither sufficient nor necessary to qualify them as stiff. Actually, an initial condition u_0 close enough to the equilibrium manifold of the solution will not develop such fast transients, and thus stiff features may not be observed.
As a first conclusion, we can deduce that an explicit time discretization scheme to solve (3.9) will generally fail to approach the right dynamics, unless we consider integration time steps smaller than the time scales disclosed by the equations. This may seem natural. Nevertheless, if we consider the counter-

part of (3.11), *i.e.*an *implicit Euler* method:

$$u_1 = u_0 - 100\,\delta t u_1, \tag{3.12}$$

and the previous $\delta t = 0.5$, we obtain the numerical approximations $u_1 = 0.019607843$ and $u_2 = 0.000384468$. Therefore, although solutions are not quite accurate, they show convergence towards the right solution with a time step several times the associated time scale, whereas for a time step larger than a given value, the explicit method will not deliver any valid result.
Before turning to the second example, we consider the *Dahlquist test equation* [Dah63], which a classical tool used to characterise the stability of numerical integration methods. We will use it here to give a first explanation on the huge difference between the implicit and explicit Euler schemes witnessed in the previous example. Let us consider the following problem:

$$\left.\begin{aligned} \mathrm{d}_t u &= \lambda u, \\ u(0) &= u_0 = 1, \end{aligned}\right\} \tag{3.13}$$

with $\lambda \in \mathbb{C}$ (a particular case was given by (3.9)). If we compute u_1 with the implicit or explicit Euler scheme for problem (3.13), we obtain:

$$u_1 = R(z)u_0, \quad z = \delta t\lambda, \tag{3.14}$$

where $R(z)$ is a rational function. For the explicit Euler method (3.11) we have:

$$R(z) = 1 + z, \tag{3.15}$$

Whereas for the implicit Euler method (3.12) yields

$$R(z) = \frac{1}{1-z}, \tag{3.16}$$

A classical analysis based on the Dahlquist test equation (3.13) allows us to define $R : \mathbb{C} \to \mathbb{C}$ given in general by (3.14), as the *stability function* of a given method. That is, $R(z)$ is the numerical solution of (3.13) given by the method itself after one time step δt. After successive applications of either Euler scheme, the numerical solution at time t_n can be written as

$$u_n = (R(z))^n\, u_0 \tag{3.17}$$

which allows us to define the *stability domain* of the method given by the set of z for which u_n remains bounded for $n \to \infty$, *i.e.*

$$S := \{\, z \in \mathbb{C} \mid |R(z)| \leq 1 \,\}. \tag{3.18}$$

For instance, considering the explicit Euler method (3.11) according to (3.15), its stability domain S is given by all $z \in \mathbb{C}$ such that

$$|1 + z| = |z - (-1)| \leq 1, \tag{3.19}$$

which is the circle of radius 1 and center -1 in the complex plane. Coming back to the previous example (3.9) with $\lambda = -100$, we can see that an explicit Euler method will remain stable as long as $z = \delta t\lambda \in S$, *i.e.*$0 \leq \delta t \leq 2/100$, which explains the previous bad results for $\delta t = 0.5$. But the stability domain given by all $z \in \mathbb{C}$ of the implicit Euler method is:

$$\left|\frac{1}{1-z}\right| \leq 1 \;\Rightarrow\; |z-1| \geq 1, \tag{3.20}$$

that is, the exterior of the circle with radius 1 and center $+1$ in the complex plane. For problem (3.9), we can then see that $R(z)$ will remain bounded for any time step $\delta t > 0$, as it is shown by $(R(z = -100\,\delta t))^n = (1 + 100\,\delta t)^{-n}$ into (3.17). This better performance of an implicit discretization for large negative λ into (3.13), characterises of stiff ODEs.

The next example of stiff configuration that we consider comes from the numerical resolution of the heat equation in one dimension by the method of lines:

$$\partial_t u - \partial_x^2 u = 0, \tag{3.21}$$

where $u : \mathbb{R} \times [0,1] \to \mathbb{R}$, $x \in [0,1]$ and we impose homogeneous Neumann conditions at the boundaries. Discretizing in space on a grid of $N_x = m$ points with second order centered finite differences, we obtain the following ODE:

$$\mathrm{d}_t U = AU, \tag{3.22}$$

with $U : \mathbb{R} \to \mathbb{R}^m$, $A \in \mathcal{M}$:

$$A = \frac{1}{\Delta x^2}\begin{pmatrix} -1 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -1 \end{pmatrix} \tag{3.23}$$

and $\Delta x = 1/(N_x + 1)$. Matrix A is diagonalizable, and its spectrum is given by:

$$\lambda_j = -\frac{4}{\Delta x^2}\sin^2\left(\frac{\pi j \Delta x}{2}\right), \quad j = 1, \ldots, N_x, \tag{3.24}$$

or alternatively,

$$\lambda_j = -4(N_x+1)^2 \sin^2\left(\frac{\pi j}{2(N_x+1)}\right), \quad j = 1, \dots, N_x, \tag{3.25}$$

for which we can identify potentially large eigenvalues increasing quadratically with the number of discretization points N_x with a maximum dispersion between $-4(N_x+1)^2$ and 0. If we operate a change of basis, we thus see that solving (3.22) amounts to solving numerous Dahlquist test-like equations (3.13), one for each eigenvalue in (3.25)). Thus if we apply the explicit Euler scheme to this problem, we need the timestep to at least scale like the square of the meshsize in order for the computation to be stable. This is a really stringent constraint on the timestep, especially when we consider that in practical applications, the characteristic spatial scales might be much smaller than the characteristic temporal scales. And once again, the implicit Euler scheme does not suffer from this problem.

We conclude this section on stiff ODEs by noting that *it is not even possible to solve the incompressible Navier-Stokes equations with fully explicit time integrators!* We show it here with the explicit Euler method applied to (3.1). Given initial (consistent) values $\boldsymbol{U}^0 = (U_i^0)_{i=1,\dots,d}$ and P^0, after one iteration of the explicit Euler method applied to the momentum equations (we cannot apply it to the continuity equation since it does not contain explicit differential expressions) we obtain, for $i \in \{1, \dots, d\}$:

$$\Gamma U_i^1 = \Gamma U_i^0 + \delta t \left(\nu L_i U_i^0 + D_i^t P^0 - \Big(\sum_{j=1,\dots,d} D_j U_i^0 U_j^0 \Big) + \Gamma S_i(0) \right)$$

There is no reason for $\boldsymbol{U}^1$ to be divergence-free, satisfying the index-2 algebraic constraint, and we cannot obtain P^1.

3.1.2 Order reduction

Another issue that can occur when applying classical integration schemes to DAES is the phenomenon of *order reduction*: if a given numerical scheme is proved to be of order $p \in \mathbb{N}^*$ when applied to approximate the solution of an ODE of type (3.3), the same numerical might be of an order $q < p$ when applied to a DAE. This makes constructing high-order integration schemes to solve DAEs more arduous than for ODEs. We illustrate this with the following example of an Hessenberg index-2 DAE, taken from [Pet82]:

$$v' = u \tag{3.26a}$$

$$v = g(x) \tag{3.26b}$$

Where $u : \mathbb{R} \to \mathbb{R}$, $v : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ is a given source term function which is C^∞. Differentiating (3.26b) gives:

$$u = g' \tag{3.26c}$$

And a another differentiation of (3.26c) gives:

$$u' = g'' \tag{3.26d}$$

Thus (3.26) is a (very simple) semi-explicit Hessenberg index-2 system. We first remark that the solution to the ODE (3.26a, 3.26d) is:

$$\begin{aligned} u &= u(0) + g'(x) - g'(0) \\ v &= v(0) + g(x) - g(0) + x\big(u(0) - g'(0)\big) \end{aligned}$$

and the solution to the derived index-1 DAE system (3.26a, 3.26c) is:

$$\begin{aligned} u &= g'(x) \\ v &= v(0) + g(x) - g(0) \end{aligned}$$

while the solution to the initial index-2 DAE (3.26a, 3.26b) is:

$$\begin{aligned} u &= g'(x) \\ v &= g(x) \end{aligned}$$

The index-1 solution is only correct if we add the particular initial value condition $v(0) = g(0)$, not needed for the index-2 system. Lowering the index of the initial system implies new constraints for the solution of the lowered-index to be that of the higher-order system. And clearly the lowered-index system can have more solutions that the higher-index one.
Let us now apply the implicit Euler scheme to (3.26a, 3.26b). If we denote again by δt the timestep, it is well known that this method as a local truncation error of order 2 when applied to classical ODE (see for example [HNW87]). Let's compute the local truncation error for our index-2 DAE above. Given initial values at some time t_0, u_0, v_0, we approximate the solutions at some $t_1 = t_0 + \delta t$ by:

$$\begin{aligned} u_1 &= \frac{v_1 - v_0}{\delta t} = \frac{g(t_1) - g(t_0)}{\delta t} \\ v_1 &= g(t_1) \end{aligned}$$

The exact solutions are:

$$u(t_1) = g'(t_1)$$
$$v(t_1) = g(t_1)$$

The error is zero for v, the differential variable. But for the algebraic variable we have:

$$\begin{aligned} u_1 - u(t_1) &= \frac{g(t_1) - g(t_0)}{\delta t} - g'(t_1) \\ &= \frac{\delta t g'(t_1) - \frac{\delta t^2}{2} g''(\eta)}{\delta t} - g'(t_1) \\ &= -\frac{\delta t}{2} g''(\eta) \end{aligned}$$

where $t_0 \leq \eta \leq t_1$. The local truncation error for the algebraic variable is only 1, and the precision order of the implicit Euler scheme has been lost when applied to this very simple index-2 system. The situation is not as bad as it seems, because it can be proven that the approximate solution for u is first-order both locally and globally [HW96]. Nevertheless, this situation will be encountered later when we apply more sophisticated integration schemes to the Navier-Stokes equations, and we will have to deal with order reduction.

3.2 The projection methods for incompressible flows

Due to their structure of index-2 DAEs, the semi-discretized Navier-Stokes equations are very difficult to solve. The first computationally efficient way to solve problem (3.1) in the primitive variables, *i.e.* in the original index-2 DAE form, was independently elaborated by Chorin [Cho68], and Temam [Tem69]. It was originally known as *the fractional-step method* (for reasons that will come to light later on when we describe this numerical strategy). Its proficiency and popularity can be measured by the tremendous amount of books, articles, engineering softwares *etc*, that relied on this method, or improved it, in order to compute incompressible flows: we can refer to [Pat80, KM85, Tur98, ABS96, HB97, BDS+05, ABC+98, PHB+98, NWK98b, NWK98a, BDG02, ZSK94, FP99], and the references therein, just to name a few. Another measure of its popularity among Computational Fluid Dynamics practitioners and researchers is the various names this method (or methods derived from it) has been referred to since its creation: splitting, prediction-projection, predictor-corrector, pressure-correction, velocity-correction *etc* rest upon the pioneering principles developed in [Cho68] and [Tem69]. However, we decided to retain a completely different numerical integration strategy, and we feel like we have to justify this choice. This is the main goal of this section. The study of the numerical properties

of *projection* schemes (mainly stability and convergence) turned out to be a prolific source of research topics; we want by no means here to produce a comprehensive summary of these discoveries, but rather to expose what, in our opinion, are the main advantages and limitations of these schemes.
We start by recasting (3.1) in a slightly different way, for aesthetic reasons. We thus want to solve:

$$\begin{cases} \dfrac{d\boldsymbol{u}}{dt} + \nabla \cdot (\boldsymbol{u}^t \otimes \boldsymbol{u}) + \nabla p - \nu \, \Delta \boldsymbol{u} = \boldsymbol{s} \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases} \tag{3.27}$$

where $\boldsymbol{u} = (u_i)_{i=1,\dots,d}$ is the (spatially) discretized velocity, p is the discretized pressure, $\boldsymbol{s}$ are the discretized source terms for the momentum equations, and the operators $\nabla \cdot (\cdot), \nabla \cdot (\cdot \otimes \cdot), \nabla(\cdot), \Delta(\cdot)$ are *linear* operators resulting from a spatial discretization (finite element, finite volume *etc*) of the Navier-Stokes equations on a given grid. It is always possible to write (3.1) in this way. We apologize if this new notations are a little bit confusing, but they allow to give a more elegant presentation of projection schemes than (3.1). We complement (3.27) with consistent initial conditions $(\boldsymbol{u}_0, p_0)$, and we consider Dirichlet boundary conditions:

$$\boldsymbol{u} = \boldsymbol{w} \quad \text{on} \quad \partial\Omega$$

We continue with a digression on *BDF* (Backward Differentiation Formula) methods to solve Hessenberg index-2 systems, as we believe this a good way to introduce fractional-step methods, even though this probably not the historical path that led to these schemes. Let us come back to the general ODE system (3.3) stated in section 3.1:

$$u' = f(x, u)$$

We want to solve this equation for $x \in [0, T]$ where $T \in \mathbb{R}$ and $T > 0$. To this purpose, we divide the interval $[0, T]$ into N intervals of length δt, we introduce the notation $x_i = i\delta t \quad \forall i \in [1, \dots, N]$. We wish to determine approximation values u_i to the exact solutions $u(x_i)$. We also introduce the notation $f_i = f(x_i, u_i)$, that we can compute when we have the approximation u_i. For $j \in \mathbb{N}$, we define the backward differences ∇^j as:

$$\nabla^0 f_n = f_n, \quad \nabla^{j+1} f_n = \nabla^j f_n - \nabla^j f_{n-1}$$

Now suppose that we know the numerical approximations $u_n, u_{n-1}, \dots, u_{n-k+1}$ for $n \in \mathbb{N}, k \in \mathbb{N}^*$ with $n > k$, and we wish to determine an approximation u_{n+1} of $u(x_{n+1})$ with these previous values. A general *multistep* method to

determine this approximation can be written as:

$$\sum_{i=0}^{k} \alpha_i u_{n-i+1} = \delta t \sum_{i=0}^{k} \beta_i f_{n-i+1} \tag{3.28}$$

where the α_i and β_i are real parameters, and we suppose that $\alpha_0 \neq 0$, in order for the impicit equation (3.28) to have a solution for u_{n+1} for δt small enough [HNW87]. The *BDF-methods* are a particular case of multistep formulas, where the coefficients β_i are such that $\beta_1 = \beta_2 = \cdots = \beta_k = 0$ and $\beta_0 = 1$, and the coefficients α_i are such that (3.28) is equivalent to [HNW87]:

$$\sum_{i=1}^{k} \frac{1}{j} \nabla^j u_{n+1} = \delta t f_{n+1} \tag{3.29}$$

For example the three first formulas are:

- k = 1: $u_{n+1} - u_n = \delta t f_{n+1}$,
- k = 2: $\frac{3}{2}u_{n+1} - 2u_n + \frac{1}{2}u_{n-1} = \delta t f_{n+1}$,
- k = 3: $\frac{11}{6}u_{n+1} - 3u_n + \frac{3}{2}u_{n-1} - \frac{1}{3}u_{n-2} = \delta t f_{n+1}$,

One definition of the *local error* of a multistep method (3.28) is the following [HNW87]:

Definition 3.2.1. *The local error of the multistep method (3.28) is defined by*

$$u(x_k) - u_k$$

where $u(x)$ is the exact solution of (3.3), and u_k is the numerical solution obtained from (3.28) by using the exact starting values $u_i = u(x_i)$ for $i = 0, 1, \ldots, k-1$.

We can now introduce a concept of order for multisteps methods.

Definition 3.2.2. *The multistep method (3.28) is said to be of* order p*, where $p \in \mathbb{N}$, if for sufficiently regular differential equations (3.3), the local error of (3.28) is $\mathcal{O}(\delta t^{p+1})$.*

The order of a multistep method is a measure of the rate of convergence of its (approximate) solution to the exact solution as $\delta t \to 0$. A classical of numerical analysis is that for $k \in \mathbb{N}^*$, the k-BDF method is of order k, when applied to a sufficiently regular ODE (3.3).
We turn next to semi-explicit Hessenberg index-2 DAEs of the form (3.5):

$$u' = f(u, v)$$

$$0 = g(u)$$

We will use existing theory [HW96] for the application of BDF-methods to this kind of problems. We assume that f and g are sufficiently differentiable, and that $g_u(u)f_v(u,v)$ is nonsingular for all x. We can then apply a general BDF method to (3.5a, 3.5b) in the following way:

$$\sum_{i=0}^{k} \alpha_i u_{n-i+1} = \delta t f(u_{n+1}, v_{n+1}) \tag{3.31a}$$

$$0 = g(u_{n+1}) \tag{3.31b}$$

where the α_i are the coefficients of the BDF method of order k. The existence and uniqueness of solutions to (3.31) are guaranteed provided δt is smaller than some threshold value δt_0 [HW96]. The following two results give us information on the convergence of BDF methods when applied to index-2 DAEs.

Lemma 3.2.1. *Let (3.5a, 3.5b) be a DAE so that $g_u(u)f_v(u,v)$ is nonsingular for all x. If the BDF method (3.31) has order p, then its local error satisfies:*

$$u(x_k) - u_k = \mathcal{O}(\delta t^{p+1}), \quad v(x_k) - v_k = \mathcal{O}(\delta t^p) \tag{3.32}$$

We see here the order reduction, charateristic of DAEs, that affects the algebraic variable. Fortunately, in the case of BDF methods, we do not lose an order for the variable v when going from the local error to the *global error*, as stated by the following theorem:

Theorem 3.2.1. *Let (3.5a, 3.5b) be a DAE so that $g_u(u)f_v(u,v)$ is nonsingular for all x. Then the k-step BDF method (3.31) is convergent of order $p = k$, if $k \leq 6$:*

$$u(x_n) - u_n = \mathcal{O}(\delta t^p), \quad v(x_n) - v_n = \mathcal{O}(\delta t^p) \quad \textit{for} \quad x_n = n\delta t \leq Const, \tag{3.33}$$

whenever the initial values satisfy

$$u(x_j) - u_j = \mathcal{O}(\delta t^{p+1}), \qquad \textit{for} \quad j = 0, \ldots, k-1$$

Proofs of these two results can be found in [HW96]. BDF methods are thus a good way to obtain high-order numerical approximations to solutions of Hessenberg index-2 DAEs. But their application to problems of type (3.5a, 3.5b) is more computationally intensive than their application to problems of type (3.3), because the implicit nature of these methods couples the resolution of (3.5a) and (3.5b), and in a lot practical situations, solving such a big (nonlinear) system is deemed too demanding.

We now come back to the Navier-Stokes equations. We denote by $D_q \boldsymbol{u}_{n+1}$ the linear combinations of the $\boldsymbol{u}_{n+1}, \boldsymbol{u}_n, \cdots \boldsymbol{u}_{n-q+1}$ terms to produce the q-step

BDF method. Applying (3.31) to (3.27) gives:

$$\frac{1}{\delta t} D_q \boldsymbol{u}_{n+1} + \nabla \cdot (\boldsymbol{u}_{n+1}^t \otimes \boldsymbol{u}_{n+1}) + \nabla p_{n+1} - \nu \, \Delta \boldsymbol{u}_{n+1} = \boldsymbol{s}_{n+1} \tag{3.34a}$$

$$\nabla \cdot \boldsymbol{u}_{n+1} = 0 \tag{3.34b}$$

With the following boundary condition:

$$\boldsymbol{u}_{n+1} = \boldsymbol{w}_{n+1} \quad \text{on} \quad \partial\Omega$$

The fractional-step methods aim at simplifying the problem (3.34) by *decoupling* the computation of $\boldsymbol{u}_{n+1}$ and p_{n+1}. This is done by *first* computing an *intermediate* velocity $\tilde{\boldsymbol{u}}_{n+1}$ by solving the momentum equations (3.34a) with a *guess*, an approximate value of the pressure p_{n+1} based upon its previous values. We name this guess $G(p_{n+1})$, and we solve:

$$\frac{1}{\delta t} D_q \tilde{\boldsymbol{u}}_{n+1} + \nabla \cdot (\tilde{\boldsymbol{u}}_{n+1}^t \otimes \tilde{\boldsymbol{u}}_{n+1}) + \nabla G(p_{n+1}) - \nu \, \Delta \tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{s}_{n+1}$$

This first part is generally called the *prediction* phase, because we compute an intermediate velocity that is not divergence-free; its a prediction of the real velocity field at time x_{n+1}. We correct this by projecting this intermediate result to a (discrete) divergence-free subspace. We call this projection $\boldsymbol{v}_{n+1}$, we compute it by looking for a scalar field ϕ so that:

$$\tilde{\boldsymbol{u}}_{n+1} - \boldsymbol{v}_{n+1} = \nabla \phi \quad \text{with} \quad \nabla \cdot \boldsymbol{v}_{n+1} = 0 \tag{3.35}$$

This part is of course called the *projection* phase (hence the name prediction-projection scheme), and it is motivated by the Helmholtz-Hodge theorem, that states that any sufficiently smooth vector field in three dimensions can be resolved into the sum of an irrotational (curl-free) vector field and a solenoidal (divergence-free) vector field. The irrotational vector field can be expressed as the gradient of a scalar field. We assume that these results are still true for our discrete spatial differentiation operators, hence the above decomposition. This problem is a (discretized) mixed Poisson equation, that we rewrite, by applying the divergence operator to (3.35):

$$\nabla^2 \phi = \nabla \cdot \tilde{\boldsymbol{u}}_{n+1}$$

The ∇^2 operator appearing here should be read: $\nabla \cdot (\nabla)$, *i.e.* the multiplication of the gradient operator by the divergence operator, it is *not* equal to the Laplacian Δ in (3.27). We solve this elliptic equation, and then compute:

$$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{u}}_{n+1} - \nabla \phi$$

We update the pressure, with a formula $p_{n+1} = p(\phi, \delta t, \boldsymbol{v}_{n+1})$ that we will precise later. This completes the numerical integration of (3.27) between x_n and x_{n+1}, and we use the *solution* $(\boldsymbol{v}_{n+1}, p_{n+1})$ to complete subsequent integration steps.

You probably noticed that something is missing in this procedure, and it is actually one of the key ingredients of fractional-step methods: what are the boundary conditions for the intermediate (non-physical at all) variables $(\tilde{\boldsymbol{u}}_{n+1}, \phi)$? The first candidate for $\tilde{\boldsymbol{u}}_{n+1}$ is of course $\boldsymbol{w}_{n+1}$, but it turns out that this is not the optimal proposition. For now, we will write:

$$\tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{w}_{n+1} + H(\cdot) \quad \text{on} \quad \partial\Omega$$

where $H(\cdot)$ is a vector field that will be precised later, and will help us to improve the precision of the prediction-projection scheme. The boundary conditions for ϕ are derived from $\nabla \cdot \boldsymbol{v}_{n+1} = 0$ on $\partial\Omega$ and the right way to express them is [GC90]:

$$\frac{\partial \phi}{\partial n} = \boldsymbol{n} \cdot (\tilde{\boldsymbol{u}}_{n+1} - \boldsymbol{v}_{n+1}) = \boldsymbol{n} \cdot H(\cdot) \quad \text{on} \quad \partial\Omega$$

We can now express a complete cycle of the procedure:

1. **Step 1: prediction phase** Solve

$$\frac{1}{\delta t} D_q \tilde{\boldsymbol{u}}_{n+1} + \nabla \cdot (\tilde{\boldsymbol{u}}^t_{n+1} \otimes \tilde{\boldsymbol{u}}_{n+1}) + \nabla G(p_{n+1}) - \nu \, \Delta \tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{s}_{n+1} \tag{3.36a}$$

   with the boundary conditions:

$$\tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{w}_{n+1} + H(\cdot) \quad \text{on} \quad \partial\Omega \tag{3.36b}$$

2. **Step 2: projection phase - Mixed Poisson equation** Solve

$$\nabla^2 \phi = \nabla \cdot \tilde{\boldsymbol{u}}_{n+1} \tag{3.36c}$$

   with the boundary conditions:

$$\frac{\partial \phi}{\partial n} = \boldsymbol{n} \cdot (\tilde{\boldsymbol{u}}_{n+1} - \boldsymbol{v}_{n+1}) = \boldsymbol{n} \cdot H(\cdot) \quad \text{on} \quad \partial\Omega \tag{3.36d}$$

3. **Step 3: projection phase - velocity and pressure updates** Compute

$$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{u}}_{n+1} - \nabla \phi \tag{3.36e}$$

$$p_{n+1} = p(\phi, \delta t, \boldsymbol{v}_{n+1}) \tag{3.36f}$$

One huge advantage of this algorithm is the *sequential* computation of the velocity, then the pressure; the name "fractional-step" comes from this; we break the simultaneous computation of the velocity and the pressure in (3.34) into two steps. In addition to this, the convective part in (3.36a) is generally treated explicitly, and (3.36a) is d independent equations on the components of $\tilde{\boldsymbol{u}}_{n+1}$. (3) implies then the resolution of $d\,\#(\mathcal{M})\times\#(\mathcal{M})$ advection-diffusion equations for the velocity components, plus 1 $\#(\mathcal{M})\times\#(\mathcal{M})$ Poisson equation for the projection. If we compare this to the fully implicit $(d+1)\,\#(\mathcal{M})\times(d+1)\,\#(\mathcal{M})$ (nonlinear) problem (3.34), we achieve a tremendous numerical computation simplification. But this simplification comes at a cost, and it is the precision of the solution obtained. Or put differently, how good of an approximation is $\boldsymbol{v}_{n+1}$, when compared to the true solution of (3.27)? This question was investigated by some researchers [GMS06, Gre90, GC90, GS87, GS00, GS03, Ran06, GQ98a, GQ98b, Gue97, GS01, She92, She96]. Before presenting the main convergence results, we begin with a heuristic study, that helps to have a quick understanding of what is happening here.

We will only consider the Stokes problem, *i.e.* we do not take into account the convective terms in the momentum equations. We begin with the simplest guess for $G(p_{n+1})$, which is 0 (original choice of Chorin [Cho68] and Temam [Tem69]). We suppose that $q > 1$, and we compare equations (3.34) and (3.36a) and we get:

$$\tilde{\boldsymbol{u}}_{n+1} - \boldsymbol{u}_{n+1} = (\boldsymbol{I} - \delta t\nu\Delta)^{-1}\delta t\nabla p_{n+1}$$

where $\boldsymbol{I}$ is the identity matrix. The matrix $(\boldsymbol{I}-\delta t\nu\Delta)$ is bounded and invertible for δt small enough, and we see that $\tilde{\boldsymbol{u}}_{n+1}$ is at best an order 1 approximation of $\boldsymbol{u}_{n+1}$! Thus the pression guess plays a great role in prediction-projection schemes. We continue our investigation by applying the divergence operator to the previous equation, and we assume that δt is small enough to consider that $(\boldsymbol{I} - \delta t\nu\Delta) \simeq \boldsymbol{I}$. We obtain:

$$\nabla^2\phi = \nabla^2\delta t p_{n+1}$$

It could be tempting to conclude that $\phi = \delta t p_{n+1}$, but if $H(\cdot) = 0$ in (3.36d) then on $\partial\Omega$ we have:

$$\frac{\partial\phi}{\partial n} \neq \frac{\partial\delta t p_{n+1}}{\partial n}$$

and this error on the boundary can propagate throughout the whole computational domain, leading to a projected velocity $\boldsymbol{v}_{n+1}$ that can still be an order 1 approximation of $\boldsymbol{u}_{n+1}$. The boundary conditions for the intermediate velocity can deteriorate the quality of the prediction-projection scheme.

With these observations in mind, we know present one of the best pressure-correction schemes to date (we remark here that ϕ acts as an approximation of the pressure, hence the name "pressure-correction"):

1. **Step 1: prediction phase** Solve

$$\frac{1}{2\delta t}(3\tilde{\boldsymbol{u}}_{n+1} - 4\boldsymbol{v}_n + \boldsymbol{v}_{n-1}) + \nabla p_n - \nu\,\Delta\tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{s}_{n+1} \tag{3.37a}$$

with the boundary conditions:

$$\tilde{\boldsymbol{u}}_{n+1} = \boldsymbol{w}_{n+1} \quad \text{on} \quad \partial\Omega \tag{3.37b}$$

2. **Step 2: projection phase - Mixed Poisson equation** Solve

$$\nabla^2\phi = \nabla\cdot\tilde{\boldsymbol{u}}_{n+1} \tag{3.37c}$$

with the boundary conditions:

$$\frac{\partial\phi}{\partial n} = 0 \tag{3.37d}$$

3. **Step 3: projection phase - velocity and pressure updates** Compute

$$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{u}}_{n+1} - \nabla\phi \tag{3.37e}$$

$$p_{n+1} = p_n + \frac{\phi - \nu\nabla\cdot\tilde{\boldsymbol{u}}_{n+1}}{2\delta t} \tag{3.37f}$$

This scheme was first proposed in a less precise form by Van Kan [VK86], and improved by Timmermans, Minev et Van de Vosse [TMVdV96]. It was proven by Guermond and Shen [GS03] to be of **order 2 for the velocity, and order 3/2 for the pressure**. So $\boldsymbol{v}_n$ is a rather good approximation of the real solution of (3.27).
However, as Gresho and Sani stated, "there is no such thing as a free lunch when solving the incompressible Navier-Stokes equations", and the fractional-step methods share some weaknesses, namely:

Remark 3.
- *There has been no successful attempt to build* good enough *fractional-step methods with an order higher than* 2 *for the velocity, let alone the pressure [GM19, GMS06]. Some authors have claimed that resorting to (3.36a-3.36f) with a third-order BDF scheme for the velocity, and a second-order approximation/guess for the pressure $G(p_{n+1})$ gave a third-order accuracy for the velocity, but under specific conditions on the timestep, that might render the whole algorithm unconditionally unstable when solving the nonlinear problem (3.27) [GMS06]. None of the other schemes devised for higher order precision for the velocity are unconditionally stable, and none of them has been proved to achieve higher-than-order-two precision for the velocity [GM19, GMS06]*

- *The projected velocity* $\boldsymbol{v}_{n+1}$ *in (3) is an approximation of* $\boldsymbol{u}_{n+1}$ *in (3.34), and it can* never *be as precised as the solution obtained by a more classical integration method, for example:*
  - *It is not possible to fix* $\boldsymbol{v}_{n+1} = \boldsymbol{w}_{n+1}$ *on* $\partial\Omega$ *in the mixed Poisson equation (3.36c, 3.36d), because that is overspecified problem [OID86, GS87]. The preferred setup is to fix the normal component of the projected velocity, with the consequence that generally,* $\boldsymbol{v}_{n+1}$ *has the wrong tangential component on the boundary. This causes the existence of a erroneous boundary layer, that can affect the flow computation elsewhere in the domain*
  - *The boundary conditions for* ϕ *in (3.37d) imply that the normal component of the pressure gradient on the boundary does not change over time [GC90]. This increases the error of the projected velocity at the boundary*
- *The pressure in (3.37f) is computed from its previous values. It implies that the pressure in (3.1) is a continuous function of time, which is not necessarily true. Another consequence of this fact is that the error in the pressure* accumulates *over time*

In light of these remarks, let us now restate the goal of our work. As we said in the Introduction, we wish to develop new dedicated algorithms for the numerical simulation of low Mach reacting flows involving a wide spectrum of spatial and temporal scales. Such phenomena can be modelled by a set of advection-diffusion-reaction equations for the conservation of the species and the energy conservation, momentum balance equations for the flow velocity and a mass conservation equation. One of the best strategies to tackle such problems, on common computational ressources, is via the use of time-space adaptive numerical tools. We deal with the wide range of spatial scales with adaptive grid techniques, abel to dynamically track reacting fronts phenomena. And we deal with the large range of time scales via operator-splitting techniques, in order to decouple processes characterized by different physical mechanisms, solving each process with a specific numerical integration method. But a key ingredient to ensure the precision of such techniques is the ability to control the errors caused by the adaptive techniques. That is why a new generation of time-space adaptive numerical tools was recently developed by Duarte *et al.* [Dua11] for the resolution of multi-scale advection-diffusion-reaction transport equations. The spatial adaptation with error control is performed by adaptive multiresolution, and they developed an adaptive dynamic splitting operator method, that rest upon the ability to control the numerical integration errors. To do so, they make sure that these errors are only due to the splitting scheme, which is possible if the subproblems are treated with high-order integration schemes.

We want to complement this particular strategy to solve low-Mach reacting flows by providing numerical tools for the hydrodynamic part. In Chapter 2 we introduced a new spatial discretization scheme for incompressible flows coupled to adaptive multiresolution in a finite-volume context. What we need now is a high-order time integration scheme for the semi-discretized incompressible Navier-Stokes equations, that can be coupled to the general time-space adaptive strategy with error control described above. We decided that prediction-projection schemes were not the ideal candidates, given their weaknesses that we highlighted in this section. We do not know of the existence of prediction-projection schemes with an order higher than 2 for the velocity. These schemes produce solutions that are an approximation of the solutions of the BDF methods, and we do not see how we can control the error between the prediction-projection solutions and the BDF solutions. And finally, even if we were to find stable third-order prediction-projection schemes, we have another problem related to the adaptive grid technique. Indeed, these are multistep methods: when computing a new iteration from timestep n to timestep $n+1$, we reach high-order by introducing in the computations the iterations at timesteps $n-k$, where k is a positive integer. With a dynamically adapting grid, we would need to "project" the solutions of the previous iterations $n-k$ onto the grid of the solution n, introducing additional errors. That is why we preferred resorting to one-step methods, and specifically Runge-Kutta methods, and we are going to present in the next chapter high-order implicit Runge-Kutta methods to solve incompressible flows.

Chapter 4

High-order implicit Runge-Kutta methods for the semi-discretized incompressible Navier-Stokes equations

This chapter presents high-order Runge-Kutta schemes for the resolution of the Hessenberg index 2 DAEs described in the previous chapter. Our goal is to obtain higher-than-second-order schemes for the numerical simulation of incompressible flows, and we start with the construction and implementation of implicit Runge-Kutta methods, with a focus on the stability requirements they have to meet in order to produce high-order and accurate solutions to index 2 DAEs. We will then present the RadauIIA method, that is 3^{rd}-order in the velocity variables, and give some details regarding its practical implementation in this work. Next, we introduce a new class of high-order *Additive Runge-Kutta* methods, tailored for the resolution of incompressible flows. We finish with the description of half-explicit Runge-Kutta methods applied to index 2 DAEs. We assess the numerical properties of these three schemes with the classical two-dimensional lid-driven cavity case.

4.1 Introduction to implicit Runge-Kutta methods

We will start by recalling the general formulation of *Runge-Kutta (RK)* methods to numerically integrate problems of type (3.3), with an initial value: $u(x_0) = u_0$. Given this initial value, and a timestep δt, we want to compute an approximate value of $u(x_0 + \delta t)$. Let s be a positive integer ($s > 0$), that is generally called *the number of stages*, a_{ij}, b_i and c_i be real numbers with $i, j = 1, \ldots, s$, a general s-stage Runge-Kutta method to determine an

approximation u_1 of $u(x_0 + \delta t)$, can be written as:

$$k_i = u_0 + \delta t \sum_{j=1}^{s} a_{ij} f\big(x_0 + c_j \delta t, k_j\big) \quad i = 1, \ldots, s \tag{4.1}$$

$$u_1 = u_0 + \delta t \sum_{i=1}^{s} b_i f\big(x_0 + c_i \delta t, k_i\big) \tag{4.2}$$

It is generally required that the c_i coefficients satisfy:

$$c_i = \sum_{j=1}^{s} a_{ij} \quad i = 1, \ldots, s$$

so that all evaluation points of f are first order approximations to the solution [HNW87]. We then only need the a_{ij} and b_i coefficients to identify a specific RK method. Unlike multi-steps methods, RK methods compute u_1 from only u_0 (or u_{n+1} form only u_n) in one step, and they are part of *one-step* methods. They are particularly suited to integrate semi-discrete PDEs with changing spatial grids (as we are doing with adaptive multiresolution). If $a_{ij} = 0$ for $i \leq j$, we are dealing with an explicit (ERK) method. In this case we can always compute the k_i and u_1. But as we stated earlier (3.1.1), explicit methods cannot be used to solve index-2 algebraic differential equations, so we will not consider them any further. If at least one $a_{ij} \neq 0$ with $i \leq j$, we are dealing with an *implicit* RK method. In this case, we have to ask ourselves whether equations (4.2) possess a solution. The most general result is obtained when f satisfies a Lipschitz condition with respect to u [HNW87], and the answer is *yes*, provided that δt is smaller than a threshold value depending on the Lipschitz constant and the a_{ij} coefficients.

If $a_{ij} = 0$ for $i < j$ and at least one $a_{ii} \neq 0$, we have a diagonal implicit Runge-Kutta method (DIRK). If all the diagonal elements of the matrix $A = (a_{ij})_{1 \leq i,j \leq s}$ of a DIRK method are equal, we have a singly diagonal implicit Runge-Kutta method (SDIRK). In the remainding cases we have an implicit Runge-Kutta method (IRK). We can further classify these techniques by their degree of difficulty in implementation. Let us assume that f is simply a linear function depending only on the differential variable u, and that u is a vector field in the x parameter from $\mathbb{R}$ to $\mathbb{R}^n$. In the case of a full IRK method, solving (4.2) amounts to solving a linear system of size $n \times s$; in the case of a DIRK method, we have to solve s linear systems of size n; and finally if we have an SDIRK, the matrix that we need to inverse at each one of the s stages is the same. Hence the IRK methods are the hardest to solve, the SDIRK the easiest and the DIRK are in between. And of course the more stages we have, the harder it is to implement an IRK, *ceteris paribus*.

It is common practice to represent a Runge-Kutta method by its Butcher

c_1	a_{11}	a_{12}	$\dots$	$a_{1,s-1}$	$a_{1,s}$
c_2	a_{21}	a_{22}		$a_{1,s-1}$	$a_{2,s}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_s	a_{s1}	a_{s2}	$\dots$	$a_{s,s-1}$	$a_{s,s}$
	b_1	b_2	$\dots$	b_{s-1}	b_s

Table 4.1: Butcher array

tableau (4.1). In what follows, we will first give some insights regarding the convergence order of IRK, and then say some words about their stability.

4.1.1 Building high-order implicit Runge-Kutta methods

A lot of work in the past 60 years has been devoted to the construction of high-order implicit Runge-Kutta methods. As stated in the previous section, more stages for an IRK means more implementation complexity; hence if two different IRK methods have the same order, we would prefer the one with less stages. We give here a definition of *order* for RK methods:

Definition 4.1.1. *A Runge-Kutta method (4.2) has* order p *if for sufficiently smooth (non-stiff) problems (3.3) we have:*

$$\|u(x_0 + \delta t) - u_1\| = \mathcal{O}(\delta t^{p+1})$$

It is not straightforward to derive implicit Runge-Kutta method with high-order. A classical result gives us an upper bound for the order p of an IRK, given its number of stages s: $p \leq 2s$. To obtain a given order, the coefficients of an IRK must satisfy *order conditions*. We will not state them here, but we refer to [HW96] for an exhaustive discussion on this subject. Building high-order IRK relies on the *simplifying assumptions*:

$$\left.\begin{aligned}
B(p): &\quad \sum_{i=1}^{s} b_i c_i^{q-1} = \frac{1}{q}, && q = 1, \dots, p; \\
C(\eta): &\quad \sum_{j=1}^{s} a_{ij} c_j^{q-1} = \frac{c_i^q}{q}, && i = 1, \dots, s, \quad q = 1, \dots, \eta; \\
D(\zeta): &\quad \sum_{i=1}^{s} b_i c_i^{q-1} a_{ij} = \frac{b_j}{q}(1 - c_j^q), && j = 1, \dots, s, \quad q = 1, \dots, \zeta.
\end{aligned}\right\} \tag{4.3}$$

The first condition $B(p)$ states that the quadrature formula $(b_i, c_i)_{i=1}^s$ is of order p. An important interpretation of the assumption $C(\eta)$ is given by the following lemma [HNW87]:

$\frac{1}{2}$	$\frac{1}{2}$
	1

Table 4.2: Butcher array of the 2-order Gauss method

$\frac{1}{2} - \frac{\sqrt{3}}{6}$	$\frac{1}{4}$	$\frac{1}{4} - \frac{\sqrt{3}}{6}$
$\frac{1}{2} + \frac{\sqrt{3}}{6}$	$\frac{1}{4} + \frac{\sqrt{3}}{6}$	$\frac{1}{4}$
	$\frac{1}{2}$	$\frac{1}{2}$

Table 4.3: Butcher array of the 4-order Gauss method

Lemma 4.1.1. *The assumption $C(\eta)$ implies that the internal stages*

$$k_i = u_0 + \delta t \sum_{j=1}^{s} a_{ij} f\big(x_0 + c_j \delta t, k_j\big) \quad i = 1, \ldots, s$$

satisfy:

$$\|u(x_0 + c_i \delta t) - k_i\| = \mathcal{O}(\delta t^{\eta+1})$$

in other words, the internal stages are of order η, and it is common practice to name η the *stage order* of the IRK. This quantity will play an important role when constructing high-order RK methods for index-2 Hesssenberg DAEs. The simplifying assumptions helped Butcher to establish the following powerful theorem [But64]:

Theorem 4.1.1. *If $B(p)$, $C(\eta)$ and $D(\zeta)$ are satisfied with $p \leq 2\eta + 2$ and $p \leq \zeta + \eta + 1$, then the method is of order p.*

Armed with this result, Butcher was among the first to discover that for any stage s, there exist IRK methods of order $2s$, *i.e.*we can always build *optimal* implicit Runge-Kutta methods. These are the *Gauss methods*, and we give the coefficients for the 1-stage and 2-stage Gauss processes in tables (4.2) and (4.3).

Sligthly less optimal processes were built by Ehle [Ehl69]: The $(2s-1)$-order *Radau IA* and *Radau IIA* methods. Tables (4.4) and (4.5) give the coefficients of the 3^{rd} order processes.
When it comes to DIRK, Nørsett & Wolfbrandt [NW77] demonstrated a useful result: if p is the order of an s-stage DIRK method, then $p \leq s + 1$. For the sake of completeness, we present in tables (4.6) and (4.7) respectively the general form of 2-stage 3-order SIDRK methods, and of 3-stage, 4-order SDIRK methods.

0	$\frac{1}{4}$	$-\frac{1}{4}$
1	$\frac{1}{4}$	$\frac{5}{12}$
	$\frac{1}{4}$	$\frac{3}{4}$

Table 4.4: Butcher array of the 3-order Radau IA scheme

$\frac{1}{3}$	$\frac{5}{12}$	$-\frac{1}{12}$
1	$\frac{3}{4}$	$\frac{1}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$

Table 4.5: Butcher array of the 3-order Radau IIA scheme

γ	γ	
$1-\gamma$	$1-2\gamma$	γ
	$\frac{1}{2}$	$\frac{1}{2}$

Table 4.6: Butcher array of the 2-stage, 3-order SDIRK methods. $\gamma = \frac{3+\sqrt{3}}{6}$

γ	γ		
$\frac{1}{2}$	$(\frac{1}{2}-\gamma)$	γ	
$(1-\gamma)$	2γ	$(1-4\gamma)$	γ
	b_1	$(1-2b_1)$	b_1

Table 4.7: Butcher array of the 3-stage, 4-order SDIRK methods.

4.1.2 Stability analysis of implicit Runge-Kutta methods

The stability properties of a numerical scheme are detrimental for its computational implementation and efficiency. As we saw in section (3.1.1), the stability restrictions of the Explicit Euler method impose the use of a timestep much smaller than required by the inner dynamics of the equation at hand, when dealing with stiff problems. IRK methods present in general better stability properties than their Explicit counterparts, but because we want to tackle DAEs, we will need an in-depth understanding of the stability properties of IRK schemes.
We come back to the Dahlquist test mentioned in section (3.1.1), $u' = \lambda u$. We want to define a *stability domain* as for ERK methods. We start with the definition of the *stability function* for IRK schemes:

Proposition 4.1.1. *The s-stage Runge-Kutta method (4.2) applied to* $u' = \lambda u$ *yields* $u_1 = R(\delta t \lambda) u_0$ *with:*

$$R(z) = 1 + z b^t (I - zA)^{-1} \boldsymbol{e} \tag{4.4}$$

where $b^t = (b_1, \dots, b_s)$, I *is the identity* $s \times s$ *matrix,* $A = (a_{ij})_{1 \leq i,j \leq s}$ *and* $\boldsymbol{e} = (\underbrace{1, \dots, 1}_{s \text{ times}})^t$.

A proof of this result can be found in [HW96]. This function is also a rational fraction, because it can be written in the following form [HW96]:

$$R(z) = \frac{det(I - zA + z\boldsymbol{e}b^t)}{det(I - zA)} \tag{4.5}$$

and we see that the numerator and denominator have a degree smaller than s.
We can now define the *stability domain* for general RK methods in exactly the same manner as we did for the Euler methods in section (3.1.1).
Next we recall the definition of *A-stability* [Dah63]. Since the solution to the equation (3.13) remains bounded exactly for $\lambda \in \mathbb{C}^-$, it is suitable for a numerical strategy to produce stable numerical solutions for $\lambda \in \mathbb{C}^-$ too, hence the following definition:

Definition 4.1.2 (A-stability)**.** *A (Runge-Kutta) method if said to be* A-stable *if its stability domain* S *satisfies*

$$\mathbb{C}^- = \{ z \mid \mathrm{Re}(z) \leq 0 \} \subset S.$$

The Implicit-Euler method, whose stability function is $R(z) = \frac{1}{1-Z}$, is clearly A-stable, which is not the case for the Explicit-Euler method. What is more, explicit Runge-Kutta methods *cannot* be A-stable, a result known as a Dahlquist *barrier* [Dah63].

Let us continue our investigation of stability properties. Prothero & Robinson [PR74] discovered that A-stability was not enough to prevent numerical instability when solving stiff nonlinear differential equations, and that the accuracy of the solution obtained was at times not even related to the order of the method used. They proposed to study the following equation:

$$u' = g'(x) + \lambda(u - g(x)) \tag{4.6}$$

with $\lambda \in \mathbb{C}$ with negative real part, and $g : \mathbb{R} \to \mathbb{R}^n$ a smooth function. The general solution to this problem takes the form:

$$u = c_0 e^{\lambda x} + g(x) \tag{4.7}$$

with $c_0 \in \mathbb{R}^n$ a given parameter. Let us apply a general s-stage Runge-Kutta method (4.2) to (4.6). if we denote u_n the solution obtained at time x_n, we compute u_{n+1} at time $x_{n+1} = x_n + \delta t$ by solving the following system:

$$\begin{cases} (I - \delta t \lambda A)\boldsymbol{k} = u_n \boldsymbol{e} + \delta t A(\boldsymbol{g}'_n - \lambda \boldsymbol{g}_n) \\ u_{n+1} = u_n + \delta t \lambda b^t \boldsymbol{k} + \delta t b^t (\boldsymbol{g}'_n - \lambda \boldsymbol{g}_n) \end{cases} \tag{4.8}$$

where $\boldsymbol{k} = (k_1, \ldots, k_s)$ and the k_i verify:

$$k_i = u_n + \delta t \sum_{j=1}^{s} a_{ij} \left[g'(x_n + c_j \delta t) + \lambda (k_j - g(x_n + c_j \delta t) \right] \quad i = 1, \ldots, s$$

and $\boldsymbol{g}_n = (g_1, \ldots, g_s)$ and the g_i verify:

$$g_i = g(x_n + c_i \delta t) \quad i = 1, \ldots, s$$

If we denote $\varepsilon_n = u_n - g(x_n)$, we can see by using (4.8) that we have the following relation:

$$\varepsilon_{n+1} = R(\delta t \lambda)\varepsilon_n + \beta(\delta t, \lambda, \boldsymbol{g}_n, \boldsymbol{g}'_n, A, b) \tag{4.9}$$

where $R(z)$ is the stability function of the IRK method considered. If λ has a very large negative real part (accute stiffness), then the true solution $u(x)$ will be quickly converging to $g(x)$ as x grows. But the numerical error ε_n behavior depends strongly on $R(z)$, and more specifically its limit as $\mathrm{Re}(z) \to -\infty$. We at least need $\lim_{\mathrm{Re}(z)\to-\infty} R(z) = 0$ if we want to have $\lim_{n\to\infty} \varepsilon_n = 0$ for λ with a very large negative real part. For example, if the stability domain of a method is *exactly* $\mathbb{C}^-$, then by the maximum principle applied to this closed half-plane, $|R(\infty)| = 1$, and the numerical error ε_n will *slowly* converge to 0. This behavior of one-step methods on this kind of stiff equations motivated the definition of the following stability property [Ehl69]:

Method	Stability function	A-stable	L-stable	stiffly accurate
4-order Gauss	$\frac{1+\frac{1}{2}z+\frac{1}{12}z^2}{1-\frac{1}{2}z+\frac{1}{12}z^2}$	Yes	No	No
3-order Radau IA	$\frac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{12}z^2}$	Yes	Yes	No
3-order Radau IIA	$\frac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{12}z^2}$	Yes	Yes	yes

Table 4.8: Stability function, A-stability, L-stability, stiff accuracy properties of some implicit Runge-Kutta methods

Definition 4.1.3 (L-stability). *A (Runge-Kutta) method if said to be* L-stable *if it is A-stable, and if in addition its stability function satisfies*

$$R(\infty) = 0$$

For example it is easy to see that if an RK method has invertible matrix A, and satisfies:

$$a_{si} = b_i \quad i = 1, \dots, s \tag{4.10}$$

then it is L-stable. This is the case for the RadauIIA (4.5) method for example. Methods satisfying (4.10) are called *stiffly accurate* [PR74], and they play an important role when solving DAEs. Finally, we summarised in table (4.8) the stability properties of some IRK methods. Proofs of these results can be found in [HW96].

4.2 Navier-Stokes equations and implicit Runge-Kutta methods

The goal of this section is to obtain a high-order IRK scheme to solve the semi-discretized Navier-Stokes equations (3.1). We want a scheme that is at least 3^{rd}-*order* for the velocity, because this variable will be used for the species transport in a low-Mach context. As we saw in section (3.1), the NS equations are semi-explicit Hessenberg index 2 DAEs, and the divergence-free constraint makes the equations *infinitely* stiff. This sets severe requirements on the numerical scheme we wish to use to solve them, let alone obtain a high-order precision solution. We will start by presenting a general way to apply implicit Runge-Kutta methods to index 2 DAEs (3.5a, 3.5b), and explain why we decided to use **the 2-stage radau IIA**; then we will give some details about the implementation of this method to solve (3.1); and we will close this section by some numerical examples.

4.2.1 Runge-Kutta methods for Hessenberg index 2 DAEs

The application of Runge-Kutta methods to DAEs is not straightforward [San13]. We will employ the ε-embedding method of [HW96] applied to index 2 Hessenberg DAEs. Let us consider the following ODE:

$$u' = f(u, v) \tag{4.11a}$$

$$\varepsilon v' = g(u) \tag{4.11b}$$

For very small values of ε, (4.11) can be viewed as an approximation of the index 2 DAE (3.5). We will apply a general IRK scheme (4.2) to (4.11), and replace ε by 0 in the nonlinear system. (4.2) applied to (4.11) gives:

$$U_{ni} = u_n + \delta t \sum_{j=1}^{s} a_{ij} f(U_{nj}, V_{nj}) \tag{4.12a}$$

$$\varepsilon V_{ni} = \varepsilon v_n + \delta t \sum_{j=1}^{s} a_{ij} g(U_{nj}) \tag{4.12b}$$

$$u_{n+1} = u_n + \delta t \sum_{i=1}^{s} b_i f(U_{ni}, V_{ni}) \tag{4.12c}$$

$$\varepsilon v_{n+1} = \varepsilon v_n + \delta t \sum_{i=1}^{s} b_i g(U_{ni}) \tag{4.12d}$$

We suppose that the matrix $A = (a_{ij})$ is invertible; we denote its inverse $W = (\omega_{ij})_{1 \le i,j \le s}$, and obtain from (4.12b)

$$\delta t g(U_{ni}) = \varepsilon \sum_{j=1}^{s} \omega_{ij} (V_{nj} - v_n) \tag{4.13}$$

We insert (4.13) into (4.12d), so that v_{n+1} is defined without reference to ε. We can now set $\varepsilon = 0$, to obtain:

$$U_{ni} = u_n + \delta t \sum_{j=1}^{s} a_{ij} f(U_{nj}, V_{nj}) \tag{4.14a}$$

$$0 = g(U_{ni}) \tag{4.14b}$$

$$u_{n+1} = u_n + \delta t \sum_{i=1}^{s} b_i f(U_{ni}, V_{ni}) \tag{4.14c}$$

$$v_{n+1} = \big(1 - \sum_{i,j=1}^{s} b_i \omega_{ij}\big) v_n + \sum_{i,j=1}^{s} b_i \omega_{ij} V_{nj} \tag{4.14d}$$

And we note that we have

$$1 - \sum_{i,j=1}^{s} b_i \omega_{ij} = R(\infty) \tag{4.14e}$$

where $R(z)$ is the stability function of the method.

The study of the convergence and stability properties of RK methods when applied to DAEs was conducted in the 1980s, and the specific order conditions for Hessenberg index 2 DAEs were first derived by Hairer, Lubich and Roche [HLR89]. This is the classical paper on the subject. It turns out that most RK methods suffer from order reduction when applied to this type of problems, and the magnitude of the order reduction depends on the *stage-order* (4.1.1), the *stability function* (4.1.2), and whether the method is *stiffly accurate* (4.1.2). We will recall here the main lemmas or theorems of [HLR89], leading us to the choice of our numerical integrator.

We start with the existence and uniqueness of a solution to (4.14). It is sufficient to prove these properties for the U_{ni} and V_{ni} in the system (4.14a, 4.14b), because u_{n+1} and V_{n+1} are then computed explcitly from these values. We denote by $\eta = u(x)$ and $\zeta = v(x)$ some initial values on the exact solution and apply one step of the Runge-Kutta method (4.14), where we drop the index n in the U_{ni} and V_{ni}. We define the local error here by the difference between the numerical solution (u_1, z_1) and the exact solution at $x + \delta t$:

$$\delta u = u_1 - u(x + \delta t), \quad \delta v = v_1 - v(x + \delta t)$$

Theorem 4.2.1 (Existence and uniqueness of the solution)**.** *Suppose that* (η, ζ) *satisfy*

$$g(\eta) = \mathcal{O}(\delta t^2), \quad g_u(\eta) f(\eta, \zeta) = \mathcal{O}(\delta t)$$

and that $g_u(u) f_v(u, v)$ *is nonsingular in a neighborhood of* (η, ζ) *If the Runge-Kutta matrix* $A = (a_{ij})$ *is invertible, then the system (4.14a, 4.14b) possesses for* $\delta t \leq \delta t_0$ *a locally unique solution which satisfies:*

$$U_i - \eta = \mathcal{O}(\delta t), \quad V_i - \zeta = \mathcal{O}(\delta t)$$

The conditions on (η, ζ) express their distance to a consistent initial solution, and they do not come as a surprise considering that the real problem (3.5) cannot have a solution if the initial values do not satisfy two algebraic constraints, one of them being hidden (3.1).

We continue with a first lemma about the local error precision, for RK methods

satisfying the conditions $B(p)$ and $C(q)$ of the Butcher simplifying assumptions (4.3).

Lemma 4.2.1. *Suppose that the Runge-Kutta method satisfies with $p \leq q+1$ and $q \leq 1$ the conditions $B(p)$ and $C(q)$ of the Butcher simplifying assumptions (4.3). Then the local error is of magnitude:*

$$\delta u = \mathcal{O}(\delta t^{q+1}), \quad P(x)\delta u = \mathcal{O}(\delta t^{q+2})$$
$$\delta v = \mathcal{O}(\delta t^{q})$$

$P(x)$ is a projection given by:

$$P(x) = I - Q(x), \quad Q(x) = (f_v(g_u f_v)^{-1} g_u)\big(u(x), v(x)\big). \tag{4.15}$$

*If in addition the Runge-Kutta method is stiffly accurate (*i.e. *satisfies $a_{si} = b_i$ for all i), then*

$$\delta u = \mathcal{O}(\delta t^{\min(p+1, q+2)})$$

We see here the importance of the stage order; methods with a small order perform particularly badly for index 2 DAEs, and they suffer from the most important order reduction, as stated by the following theorem:

Theorem 4.2.2 (Convergence for the u-component)**.** *Suppose that $g_u(u) f_v(u, v)$ is nonsingular in a neighborhood of the solution $(u(x), v(x))$ of (3.5a, 3.5b), and that the initial values are consistent. If the Runge-Kutta matrix $A = (a_{ij})$ is invertible, $|R(\infty)| < 1$ and the local error satisfy:*

$$\delta u = \mathcal{O}(\delta t^{r}), \quad P(x)\delta u = \mathcal{O}(\delta t^{r+1})$$

with $P(x)$ given by (4.15), then the method (4.14) is convergent of order r, i.e.*:*

$$u_n - u(x_n) = \mathcal{O}(\delta t^{r}), \quad \textit{for} \quad x_n = n\delta t \leq \textit{Const.}$$

If in addition we have $\delta u = \mathcal{O}(\delta t^{r+1})$, then we have $g(u_n) = \mathcal{O}(\delta t^{r+1})$.

We first note that the numerical solution u_n does not satisfy the constraint (3.5b) *exactly*. This will further increase the numerical error in low-Mach reacting flows, because this error of the flow will propagate in the transported scalars. *But* if the method is stiffly accurate, then we have the nice result that: $g(u_n) = g(U_{ns}) = 0$.
We are able to study here the convergence of the u-component alone beacause the U_{ni} in (4.14) do not depend on v_n. This can be a good feature for a numerical scheme to integrate the incompressible Navier-Stokes equations, because it can be difficult to maintain a proper / non polluted pressure variable throughout the whole computation (more on this later when we treat the spurious modes).

Together with lemma 4.2.1, this theorem shows that the precision order of an RK method, when applied to an index 2 DAE, is reduced to $q+1$ whatever the precision order p is for more regular ODEs. For examples SDIRK methods, which have a stage-order $q=1$, can only be of order 2 when applied to systems (3.5a, 3.5b). Some class of methods, however, do not suffer order reduction for the u-component, and there are the subject of the following theorem:

Theorem 4.2.3. *Suppose that the Runge-Kutta matrix $A=(a_{ij})$ is invertible and that $a_{si}=b_i$ for $i=1,\dots,s$. Then the conditions $B(p)$, $C(q)$, $D(\xi)$ with with $p\leq 2q$ and $p\leq \xi+q+1$ imply that the local error satisfies:*

$$\delta u = \mathcal{O}(\delta t^{p+1})$$

for the index 2 system (3.5a, 3.5b).

Together with theorem 4.2.2, this theorem implies that Radau IIA methods, for example, do not suffer order reduction in the u-component when applied to index 2 systems. But most methods suffer from order reduction for the v-component:

Theorem 4.2.4 (Convergence for the v-component)**.** *Suppose that $g_u(u)f_v(u,v)$ is nonsingular in a neighborhood of the solution $(u(x),v(x))$ of (3.5a, 3.5b), and that the initial values are consistent. If the Runge-Kutta matrix $A=(a_{ij})$ is invertible, $|R(\infty)|<1$ and the global error of the u-component, and the local error for the v-component satisfy:*

$$\begin{aligned} &u_n - u(x_n) = \mathcal{O}(\delta t^r), \quad \textit{for} \quad x_n = n\delta t \leq Const. \\ &g(u_n) = \mathcal{O}(\delta t^{r+1}) \quad \delta v = \mathcal{O}(\delta t^r) \end{aligned}$$

then the global error for the v-component is:

$$v_n - v(x_n) = \mathcal{O}(\delta t^r), \quad \textit{for} \quad x_n = n\delta t \leq Const.$$

The convergence of the v-component relies heavily on the precision of the u-comonent, which is not a surprise, given the fact that this variable is *instantly* linked to the differential variable via the hidden constraint (3.5c). The local and global errors for the algebraic constraint are the same, another consequence of the fact that we are dealing with a DAE. But the precision order will be limited to the stage order of the method.
Finally, we present a convergence theorem for methods with $|R(\infty)|=\pm 1$, which is the case for Gauss methods:

Theorem 4.2.5. *Suppose that $g_u(u)f_v(u,v)$ is nonsingular in a neighborhood of the solution $(u(x),v(x))$ of (3.5a, 3.5b), and that the initial values are consistent. Suppose that the Runge-Kutta matrix $A=(a_{ij})$ is invertible, and let conditions $B(p)$ and $C(q)$ be satisfied.*

Method	Stages	local error		global error	
		u	v	u	v
Gauss	s odd	δt^{s+1}	δt^{s}	δt^{s+1}	δt^{s-1}
	s even	δt^{s+1}	δt^{s}	δt^{s-2}	δt^{s+1}
Radau IA	s	δt^{s}	δt^{s-1}	δt^{s}	δt^{s-1}
Radau IIA	s	δt^{2s}	δt^{s}	δt^{2s-1}	δt^{s}
SDIRK	3	δt^{2}	δt^{1}	δt^{2}	δt^{1}

Table 4.9: Error estimates for the index 2 problem (3.5a, 3.5b)

1. *if* $R(\infty) = +1$, $q \leq 2$ *and* $p \leq q$, *then*

$$u_n - u(x_n) = \mathcal{O}(\delta t^q), \quad v_n - v(x_n) = \mathcal{O}(\delta t^{q-2})$$

2. *if* $R(\infty) = -1$, $q \leq 2$ *and* $p \leq q$, *then*

$$u_n - u(x_n) = \mathcal{O}(\delta t^{q+1}), \quad v_n - v(x_n) = \mathcal{O}(\delta t^{q-1})$$

All these different convergence results can be summarized in table (4.9), that we took from [HW96].
We wish to attain 3^{rd}-order precision for the velocity, with the least computational effort. Clearly, The **Radau IIA** method is our best choice, as it is the only one able to satisfy our requirements in only 2 stages. In addition, for stiffly accurate method like this one, $U_{ns} = u_{n+1}$ in (4.14a-4.14c), so that we save the last step of the method. And this method being L-stable, $R(\infty) = 0$ in (4.14d), and the computation of the algebraic variable is not correlated to the previous timesteps.

4.2.2 Implementation of the Radau IIA method to solve the semi-discretized incompressible Navier-Stokes equations

We want to apply the Radau IIA method (see table 4.5) to the semi-discretized incompressible Navier-Stokes equations. We derive the ε-method (4.14) described in section 4.2.1 in the following way. Given consistent velocities and pressure fields $(\boldsymbol{U^0}, P^0)$ at time t_0, we want to obtain an approximate solution of (3.1) $(\boldsymbol{U^1}, P^1)$, at time $t_0 + \delta t$ with an implicit 2-stage Runge-Kutta method. The ε-embedding method recasts equation 3.1 in the following form:

$$
\begin{aligned}
\Gamma g_i^1 &= \Gamma U_i^0 + \delta t a_{11}\Big(\nu L_i g_i^1 + D_i^t k^1 \\
&\quad - \big(\sum_{j=1,\dots,d} D_j g_i^1 g_j^1\big) + \Gamma S_i(t_0 + \delta t c_1)\Big) \\
&\quad + \delta t a_{12}\Big(\nu L_i g_i^2 + D_i^t k^2 \\
&\quad - \big(\sum_{j=1,\dots,d} D_j g_i^2 g_j^2\big) + \Gamma S_i(t_0 + \delta t c_2)\Big) \\
\Gamma g_i^2 &= \Gamma U_i^0 + \delta t a_{21}\Big(\nu L_i g_i^1 + D_i^t k^1 \\
&\quad - \big(\sum_{j=1,\dots,d} D_j g_i^1 g_j^1\big) + \Gamma S_i(t_0 + \delta t c_1)\big) \\
&\quad + \delta t a_{22}\big(\nu L_i g_i^2 + D_i^t k^2 \\
&\quad - \big(\sum_{j=1,\dots,d} D_j g_i^2 g_j^2\big) + \Gamma S_i(t_0 + \delta t c_2)\Big)
\end{aligned}
\tag{4.16a}
$$

$$
\begin{aligned}
\sum_{i=1,\dots,d} D_i g_i^1 &= S_{\mathrm{div}}(t_0 + \delta t c_1) \\
\sum_{i=1,\dots,d} D_i g_i^2 &= S_{\mathrm{div}}(t_0 + \delta t c_2)
\end{aligned}
\tag{4.16b}
$$

$$
\begin{aligned}
\Gamma U_i^1 &= \Gamma U_i^0 \\
&\quad + \delta t b_1\Big(\nu L_i g_i^1 + D_i^t k^1 - \big(\sum_{j=1,\dots,d} D_j g_i^1 g_j^1\big) + \Gamma S_i(t_0 + \delta t c_1)\Big) \\
&\quad + \delta t b_2\Big(\nu L_i g_i^2 + D_i^t k^2 - \big(\sum_{j=1,\dots,d} D_j g_i^2 g_j^2\big) + \Gamma S_i(t_0 + \delta t c_2)\Big)
\end{aligned}
\tag{4.16c}
$$

$$
P^1 = \Big(1 - \delta t \sum_{i,j=1}^{2} b_i \omega_{ij}\Big) P^0 + \sum_{i,j=1}^{2} b_i \omega_{ij} k^j \tag{4.16d}
$$

where δt is the timestep, g_i^j, k^j are intermediate variables, and $W = (\omega_{ij})_{1 \le i,j \le 2}$ is the inverse of the matrix A corresponding to the Radau IIA method (see table 4.5). Since this method is stiffly accurate, we actually have $U_i^1 = g_i^2$; and since it is L-stable, we actually have $(1 - \delta t \sum_{i,j=1}^{2} b_i \omega_{ij}) = 0$ [HW96], which simplify the computations.

We see that (4.16a, 4.16b) is a nonlinear system. The first idea that we tried to solve it was an approximate Newton-Raphson iteration method. We will develop it here for $d = 2$, the (computational) space dimension. We also consider that we are dealing with periodic boundary conditions in both space directions, so that $S_{\mathrm{div}}(t) = 0$, and that the source terms for the momentum equations are

null, so that $S_1(t) = S_2(t) = 0$. Let:

$$\begin{aligned} f_1(x,y,z) &= \nu L_1 x + D_1^t\ z - (D_1 x^2 + D_2 xy) \\ f_2(x,y,z) &= \nu L_2 x + D_2^t\ z - (D_1 xy + D_2 y^2) \\ f_3(x,y) &= D_1 x + D_2 y \end{aligned}$$

We denote: $Z = (g_{11}, g_{21}, g_{31}, g_{12}, g_{22}, g_{32})^t$, and we recast (4.16a, 4.16b) in this case as $F(Z) = 0$, where:

$$\begin{aligned} F_1(Z) &= \Gamma g_{11} - \Gamma U_1^0 - a_{11}\delta t f_1(g_{11}, g_{21}, g_{31}) - a_{12}\delta t f_1(g_{12}, g_{22}, g_{32}) \\ F_2(Z) &= \Gamma g_{12} - \Gamma U_1^0 - a_{21}\delta t f_1(g_{11}, g_{21}, g_{31}) - a_{22}\delta t f_1(g_{12}, g_{22}, g_{32}) \\ F_3(Z) &= \Gamma g_{21} - \Gamma U_2^0 - a_{11}\delta t f_2(g_{11}, g_{21}, g_{31}) - a_{12}\delta t f_2(g_{12}, g_{22}, g_{32}) \\ F_4(Z) &= \Gamma g_{22} - \Gamma U_2^0 - a_{21}\delta t f_2(g_{11}, g_{21}, g_{31}) - a_{22}\delta t f_2(g_{12}, g_{22}, g_{32}) \\ F_5(Z) &= f_3(g_{11}, g_{21}, g_{31}) \\ F_6(Z) &= f_3(g_{12}, g_{22}, g_{32}) \end{aligned}$$

We now apply Newton iteration to $F(Z) = 0$. If we start with a known value Z^k of Z at iteration k, we compute Z^{k+1} with:

$$\begin{aligned} J_Z(Z^k)\Delta Z^k &= -F(Z^k) \\ Z^{k+1} &= Z^k + \Delta Z^k \end{aligned}$$

where J_F is the jacobian matrix of F. We make the following assumption to ease the computations: $J_F(Z^k) \sim J_F(Z^0)$ during all the iteration, and of course: $Z^0 = (U_1^0, U_2^0, P^0, U_1^0, U_2^0, P^0)^t$. We also note:

$$\begin{aligned} \frac{\partial D_1 x^2}{\partial x} &= 2D_1(x) \\ \frac{\partial D_2 xy}{\partial x} &= 2D_2(y) \\ \frac{\partial D_1 xy}{\partial y} &= 2D_1(x) \\ \frac{\partial D_2 y^2}{\partial y} &= 2D_2(y) \end{aligned}$$

where $D_i(x), D_i(y)$ are matrices whose coefficients depend respectively on the vectors x and y. We can now write $\frac{1}{\delta t} J_F(Z^0)$:

$$
\frac{1}{\delta t} J_F(Z^0) = \begin{pmatrix}
\begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{11}\nu L_1 \\ +2a_{11} D_1(U_1^0) \\ +a_{11} D_2(U_2^0) \end{pmatrix} & \begin{pmatrix} -a_{12}\nu L_1 \\ +2a_{12} D_1(U_1^0) \\ +a_{12} D_2(U_2^0) \end{pmatrix} & a_{11} D_2(U_1^0) & a_{12} D_2(U_1^0) & -a_{11} D_1^t & -a_{12} D_1^t \\
\begin{pmatrix} -a_{21}\nu L_1 \\ +2a_{21} D_1(U_1^0) \\ +a_{21} D_2(U_2^0) \end{pmatrix} & \begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{22}\nu L_1 \\ +2a_{22} D_1(U_1^0) \\ +a_{22} D_2(U_2^0) \end{pmatrix} & a_{21} D_2(U_1^0) & a_{22} D_2(U_1^0) & -a_{21} D_1^t & -a_{22} D_1^t \\
a_{11} D_1(U_2^0) & a_{12} D_2(U_1^0) & \begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{11}\nu L_2 \\ +2a_{11} D_1(U_1^0) \\ +a_{11} D_2(U_2^0) \end{pmatrix} & \begin{pmatrix} -a_{12}\nu L_2 \\ +2a_{12} D_1(U_1^0) \\ +a_{12} D_2(U_2^0) \end{pmatrix} & -a_{11} D_2^t & -a_{12} D_2^t \\
a_{21} D_1(U_2^0) & a_{22} D_2(U_1^0) & \begin{pmatrix} -a_{21}\nu L_2 \\ +2a_{21} D_1(U_1^0) \\ +a_{21} D_2(U_2^0) \end{pmatrix} & \begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{22}\nu L_2 \\ +2a_{22} D_1(U_1^0) \\ +a_{22} D_2(U_2^0) \end{pmatrix} & -a_{21} D_2^t & -a_{12} D_2^t \\
D_1 & 0 & D_2 & 0 & 0 & 0 \\
0 & D_1 & 0 & D_2 & 0 & 0
\end{pmatrix} \tag{4.17}
$$

We first tried the Newton method, because of its quadratic convergence property. We were able to solve the nonlinear problem (4.16a, 4.16b) with this method, but it was very inefficient; among other reasons for this inefficiency, we can mention:

Remark 4. ***Newton method drawbacks***

- *The convergence radius of the Newton method is known to be very small; this means that we need a "good" initial value at the beginning of the Newton iterations. This practically means, in our case, that we need a very small timestep, in order for the different values to be close enough. This goes against the good stability properties of the Radau IIA method, one of the reasons we chose it in the first place: we have a stringent constraint on the timestep, that is not related to the physical properties of the problem at hand*

- *We have to rebuild the matrix (4.17) at each iteration, even if the computational grid was not modified. This adds some computational overhead, even more so in the context of adaptive multiresolution, because the connectivity between the cells is harder to obtain than when dealing with a uniform grid*

- *The matrix (4.17) is not symmetric at all; and even though the matrices L_i are symmetric, we cannot take advantage of this fact when (numerically) inverting it. This matrix is of* saddle point *type [BGL05] , i.e., it has the form:*

$$\begin{pmatrix} B & G \\ D & 0 \end{pmatrix} \tag{4.18}$$

  and thus represents a significant challenge for solver softwares

That is why we decided to resort to fixed-point iteration techniques. Even though they do not possess theoretical quadratic convergence properties, in nonsteady processes, they can require as few iteration steps as Newton methods to convergence [Tur98]. We went for a fixed-point Picard iteration method, where we write the convection terms explicitly at each iteration step. This time, we apply a function G to Z until we obtain $G(Z) = Z$. The function G is of the form:

$$G(Z) = A^{-1}\varsigma(Z)$$

where $\varsigma(Z)$ is the vector defined by:

$$\varsigma_1(Z) = \Gamma U_1^0 - a_{11}\left(\sum_{j=1}^{2} D_j g_{11} g_{j1}\right)$$
$$\varsigma_2(Z) = \Gamma U_1^0 - a_{21}\left(\sum_{j=1}^{2} D_j g_{12} g_{j2}\right)$$
$$\varsigma_3(Z) = \Gamma U_2^0 - a_{11}\left(\sum_{j=1}^{2} D_j g_{21} g_{j1}\right)$$
$$\varsigma_4(Z) = \Gamma U_2^0 - a_{21}\left(\sum_{j=1}^{2} D_j g_{22} g_{j2}\right)$$
$$\varsigma_5(Z) = 0$$
$$\varsigma_6(Z) = 0$$

and the matrix A is defined by:

$$
A = \begin{pmatrix}
\frac{1}{\delta t}\Gamma - a_{11}\nu L_1 & -a_{12}\nu L_1 & 0 & 0 & -a_{11}D_1^t & -a_{12}D_1^t \\
-a_{21}\nu L_1 & \frac{1}{\delta t}\Gamma - a_{22}\nu L_1 & 0 & 0 & -a_{21}D_1^t & -a_{22}D_1^t \\
0 & 0 & \frac{1}{\delta t}\Gamma - a_{11}\nu L_2 & -a_{12}\nu L_2 & -a_{11}D_2^t & -a_{12}D_2^t \\
0 & 0 & -a_{21}\nu L_2 & \frac{1}{\delta t}\Gamma - a_{22}\nu L_2 & -a_{21}D_2^t & -a_{12}D_2^t \\
D_1 & 0 & D_2 & 0 & 0 & 0 \\
0 & D_1 & 0 & D_2 & 0 & 0
\end{pmatrix} \tag{4.19}
$$

The Lipschitz constant of the function G decreases linearly with δt, so that we can always transform it into a contraction mapping, ensuring the existence of a fixed-point. When we reach this fixed-point Z, we have $F(Z) = 0$, and we have solved the nonlinear problem (4.16a, 4.16b). This method was much more efficient than the Newton iterations, and we believe that this is due in part to the following facts:

Remark 5. ***Fixed-point Picard iteration vs Newton iteration***

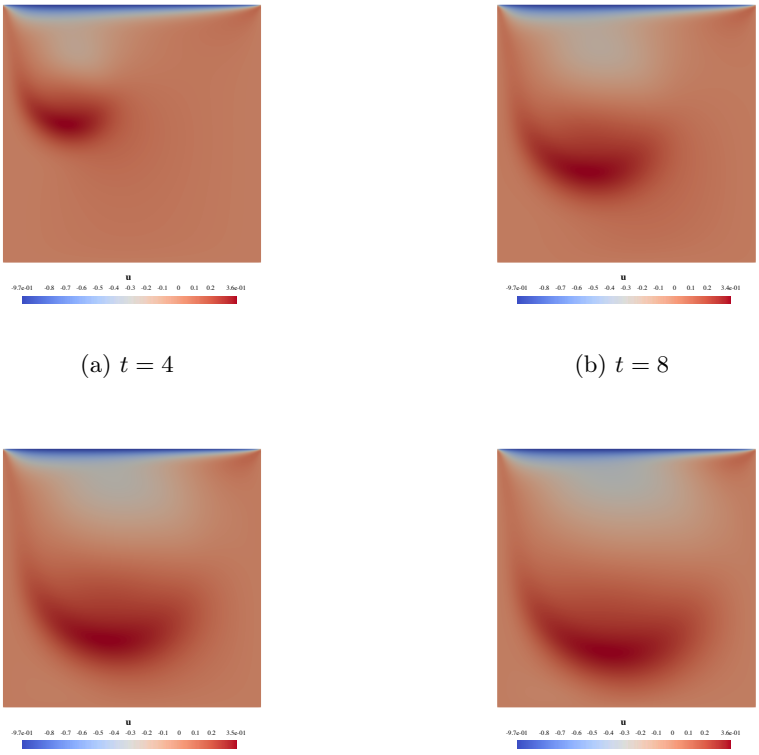
- *The convergence radius of the Picard method turned out to be much bigger than Newton method's one. The iterations converged with large timesteps in a large range of test cases*
- *Matrix (4.19) only depends on the grid connectivity, so that we do not have to recompute it at every timestep if the grid does not change*
- *Matrix (4.19) is* sparser *than matrix (4.17), making it easier to invert*
- *What is more, both matrices are of the saddle-point type (4.18), but for matrix (4.19), the block B is a block diagonal matrix, easier to invert than the block B of matrix (4.17)*
- *Matrix (4.19), is more akin to a symmetric matrix than (4.17)*

Once we have found a method to solve the nonlinear system, we end up with a linear system (of saddle-point type) that we have to solve at each timestep, in an efficient manner. We postpone to Chapter 6 the resolution of linear systems. And now we will present some numerical results to showcase the performance of the Radau IIA method.

4.2.3 Numerical simulations

We will check the order of our various Runge-Kutta schemes with the classical case of the lid-driven cavity [BP98, GGS82]. We discretize equations (2.1) on a uniform grid with the spatial scheme (2.5), in an open bounded domain Ω in $2D$, where $\Omega =]-0.5, 0.5[\times]-0.5, 0.5[$. The fluid is initially at rest within Ω ($\boldsymbol{u}_{ini}(\boldsymbol{x}, 0) = \mathbf{0}$ for $\boldsymbol{x} \in \Omega$), and we impose Dirichlet boundary conditions on the four edges of $\partial\Omega$: the velocity is set to zero on all but one of these boundaries, in our case the top one at $y = 0.5$, where the tangential velocity is set to $u_1(x, 0.5, t) = -1$. The Reynolds number is $Re = 1000$.
We then solve the semi-discretized equations (3.1) with the RadauIIA method until a steady-state is attained[1]. Figures (4.1) and (4.2) show the evolution of the horizontal and vertical components of the velocity, respectively. Figures (4.3) show the evolution of the norm of the velocity and the streamlines.

[1] We consider that we have reached convergence when the relative variation in the velocity field between timestep t^n and t^{n+1} is smaller than a user-defined threshold.



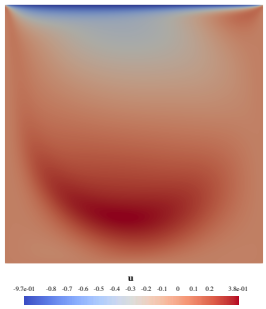
(a) $t = 4$ (b) $t = 8$

(c) $t = 12$ (d) $t = 16$

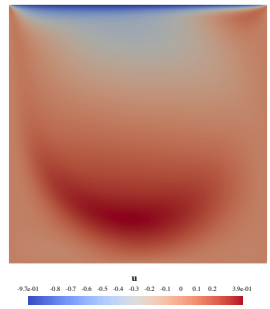
Figure 4.1: Lid-driven cavity. Evolution of the horizontal velocity component for $Re = 1000$.

Finally, figure (4.4) shows the contour of the vorticity at steady-state, and there is a perfect match with the simulations obtained by Botella & Peyret in [BP98]. These computations were made on a uniform grid, with a grid level of 8 (256×256).

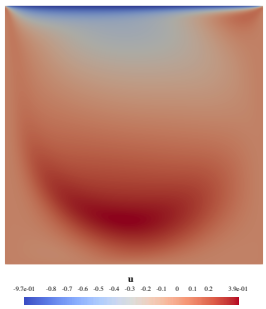
Since there are no exact solutions to the Navier-Stokes equations for the lid-driven cavity test case, we chose to check the temporal accuracy of the method in the following way. We solve the equations with the RadauIIA method until $t = 5.0$, with a very small timestep, and we compare this solution with solutions obtained at the same final time, but with larger timesteps. The computations were made on a uniform grid, with a grid level of 6 (64×64). Figures (4.5), (4.6) and (4.7) represent the order of accuracy of the RadauIIA method for the horizontal component of the velocity, the vertical one and the pressure, respectively. We observe that we effectively reach 3^{rd}-order accuracy for the



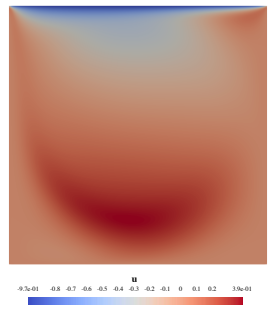
(e) $t = 24$



(f) $t = 32$

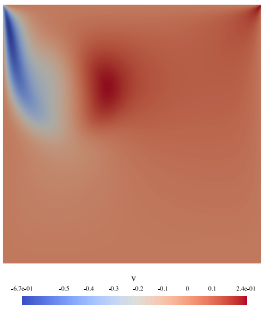


(g) $t = 36$

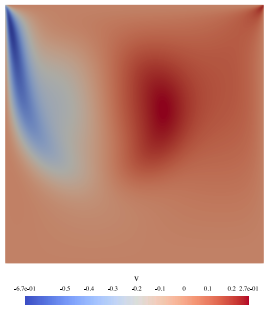


(h) $t = 40$

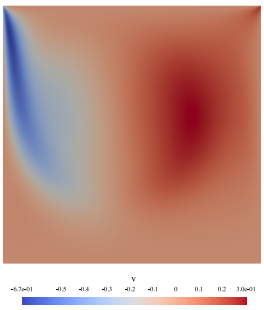
Figure 4.1: Lid-driven cavity. Evolution of the horizontal velocity component for $Re = 1000$.



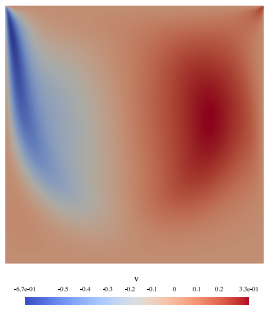
(a) $t = 4$



(b) $t = 8$

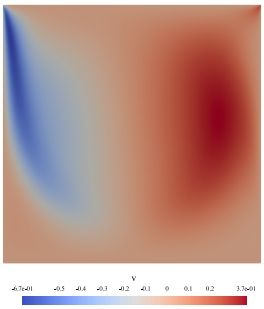


(c) $t = 12$

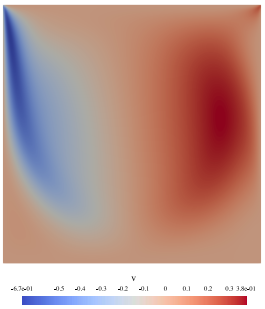


(d) $t = 16$

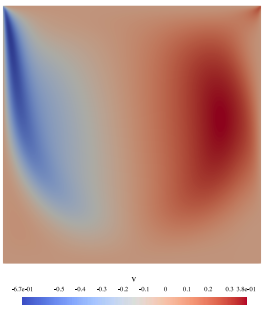
Figure 4.2: Lid-driven cavity. Evolution of the vertical velocity component for $Re = 1000$.



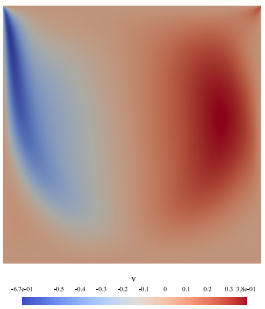
(e) $t = 24$



(f) $t = 32$

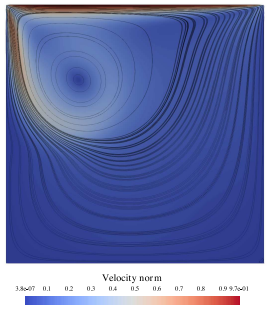


(g) $t = 36$

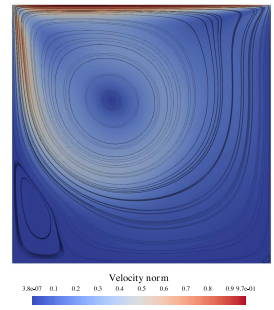


(h) $t = 40$

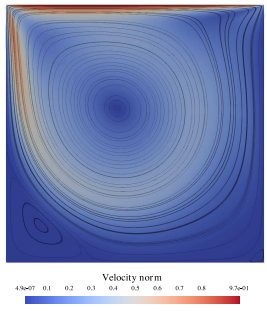
Figure 4.2: Lid-driven cavity. Evolution of the vertical velocity component for $Re = 1000$.



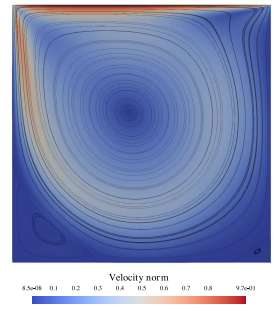
(a) $t = 4$



(b) $t = 8$

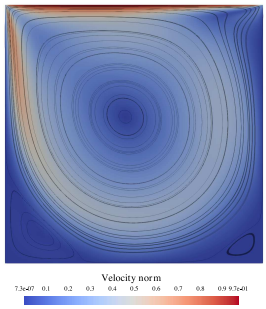


(c) $t = 12$

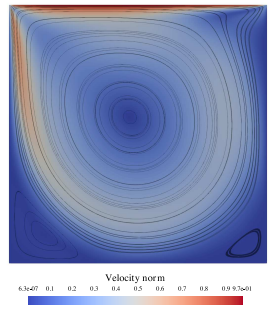


(d) $t = 16$

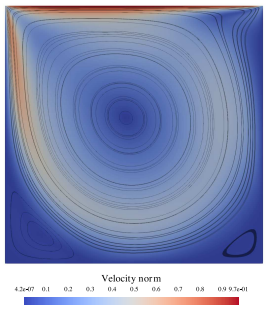
Figure 4.3: Lid-driven cavity. Evolution of the norm of the velocity and the streamlines for $Re = 1000$.



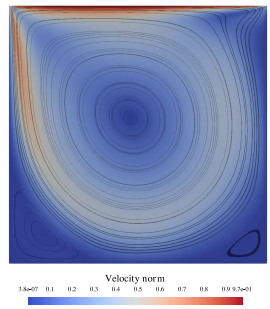
(e) $t = 24$



(f) $t = 32$



(g) $t = 36$



(h) $t = 40$

Figure 4.3: Lid-driven cavity. Evolution of the norm of the velocity and the streamlines for $Re = 1000$.

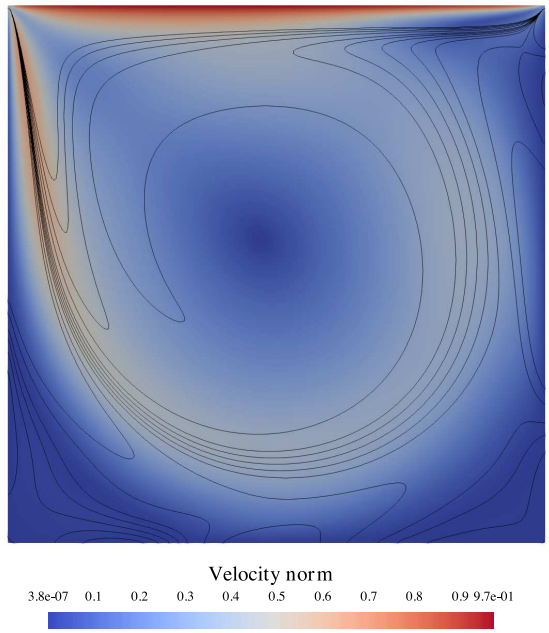


Figure 4.4: Lid-driven cavity. Norm of the velocity and Vorticity contours at steady-state for $Re = 1000$.

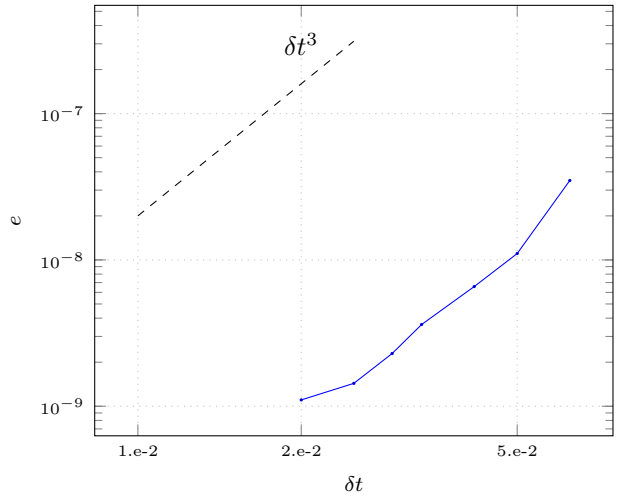


Figure 4.5: Lid-driven cavity. Temporal order of accuracy for the RadauIIA method; horizontal component of the velocity

two components of the velocity.

4.3 A new high-order additive Runge-Kutta method

In section (4.2.2) we developed an implementation of the 2-stage Radau IIA method to solve the incompressible Navier-Stokes equations. We explained why this scheme was perfectly suited to solve Hessenberg index 2 DAEs, and showed that it offered 3^{rd} order for the velocity. But at each timestep, we have to solve the nonlinear system (4.16a, 4.16b), and the implementation difficulties of this operation make the Radau IIA method less attractive. In addition, if we come back to the general formulation of index 2 DAEs (3.5a, 3.5b), the presence of the algebraic variable in equation (3.5a) causes the system to be a DAE. Thus, in the case of the incompressible Navier-Stokes equations (3.1), the nonlinear convection terms in the momentum equations are not the cause of the infinite stiffness resulting from the algebraic constraint. This means that we do not have to treat these terms with an implicit Runge-Kutta method with enhanced stability properties capable of dealing with DAEs. A natural idea that comes to mind is then to use *two different schemes* to integrate (3.1): one for the DAE and stiff parts of the equations, that are completely linear, and one that is easy to apply to the nonlinear convective terms. This strategy is an illustration of a broader paradigm called *splitting methods*: these methods aim at taking advantage of the inherent structure of the governing equations, by applying integration methods to specific parts of the equations, then rolling them into a composite solver. This is opposed to using a single method to solve the

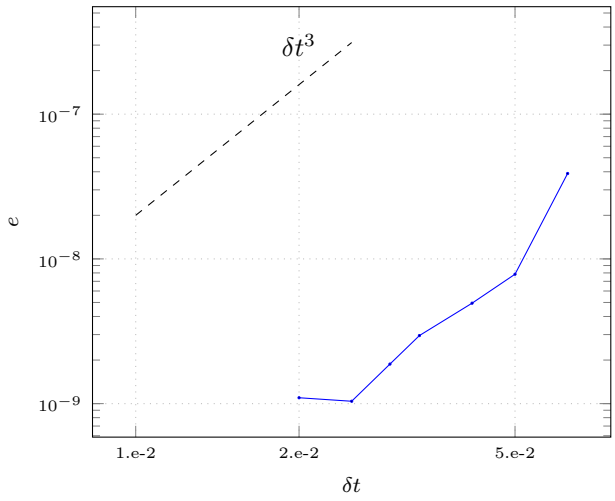


Figure 4.6: Lid-driven cavity. Temporal order of accuracy for the RadauIIA method; vertical component of the velocity

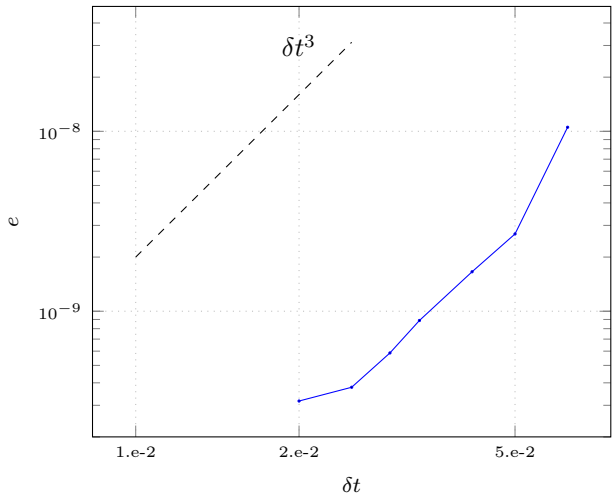


Figure 4.7: Lid-driven cavity. Temporal order of accuracy for the RadauIIA method; pressure

equations at hand: the heuristic thinking being that by applying specific solvers to separate parts of the equations, it is possible to end up with a resolution strategy with better computational efficiency than a single method applied to all parts. Time-operator splitting methods have been significantly prolific in numerical simulations of complex multi-physics real-world phenomena, and the litterature on these schemes is huge. It is not the purpose of this work to review them all, but we want to bring attention to a particular subset of these methods. It is often the case in applications that the function f in (3.3) can be naturally decomposed into η additive terms:

$$u' = \sum_{i=1}^{\eta} f^{[i]}(x, u) \tag{4.20}$$

This is the case for example in the direct numerical simulation of multi-scale combustion fronts with detailed kinetics, problem to which the splitting methods have been applied in a large variety of contexts [GPMD88, YP98, SLGS03, RP08, KNW99, AV13, NWK98b, NK05, SPN06, DB00c, BDG02, BDA+06, BDG+07]. The governing equations of the energy and species concentrations balances in gas phase combustion can be divided into advection, diffusion and reaction processes. These different processes come with a broad spectrum of time and space scales. The reactive timescales impose stringent stiffness constraints to the system, but there are highly localized in space. In general the diffusion terms introduce sitffness due to fine spatial discretization (cf. section 3.1.1), but with timescales order of magnitude bigger than the reactive phenomena. And the advective terms contribute eigenvalues that are predominantly imaginary. The idea is then to split this problem into smaller subproblems, reactive, advective and diffusive subproblems, and to use dedicated solvers for each one of them. More formally, let us consider a general species balance equation on a concentration Y yielding the following form:

$$Y' = f^{\alpha}(x, Y) + f^{\rho}(x, Y) + f^{\kappa}(x, Y) \tag{4.21}$$

where $f^{\alpha}(x, Y)$ are the advective term, $f^{\rho}(x, Y)$ the reactive term, and $f^{\kappa}(x, Y)$ the diffusive term. If we denote by Y_0 the initial value of the ODE (4.21), we can choose a specific numerical solver $\mathcal{S}_i$ for each one of the subproblem:

$$Y' = f^{i}(x, Y) \tag{4.22}$$

where $i = \alpha, \kappa, \rho$. For a given timestep Δt, we can then compute Y_1, the numerical integration of (4.21) after Δt time by the solver $\mathcal{S}_{split}$ defined as:

$$Y_1 = \mathcal{S}_{split} Y_0 = \mathcal{S}_{\alpha}^{\Delta t} \mathcal{S}_{\kappa}^{\Delta t} \mathcal{S}_{\rho}^{\Delta t} Y_0 \tag{4.23}$$

This scheme was used for example in [GPMD88]. An advantage of using such splitting schemes is the ability to use specific sub-timesteps for each supbroblem, as long as these timesteps are smaller than Δt of course. This way, we decouple the physical phenomena, and integrate each sub-part with timesteps related to its timescales. Unfortunately, the scheme (4.23) is only of order 1 in time. An improvement over this result was designed based on the seminal papers of Strang [Str63, Str68], the so-called Strang splitting scheme. An implementation of this scheme on problem (4.21) can be formulated as follows:

$$Y_1 = \mathcal{S}_{split} Y_0 = \mathcal{S}_\rho^{\Delta t/2} \mathcal{S}_\kappa^{\Delta t/2} \mathcal{S}_\alpha^{\Delta t} \mathcal{S}_\kappa^{\Delta t/2} \mathcal{S}_\rho^{\Delta t/2} Y_0 \tag{4.24}$$

This scheme is second-order in time, and was used for example in [KNW99, NWK98b, NK05]. But these methods come with splitting errors: the errors of the dedicated solvers are intertwined in the computation of Y_1. This can result for example in the fact that the order of the scheme (4.24) is reduced to 1. However, in [Dua11], it was showed that the global error in (4.24) can be decoupled from the errors of the subproblems, if we solve these latter with high-order dedicated methods. We refer to [Dua11] for a much more complete review of splitting schemes used in the context of reactive flows. The fractional-step methods described in section 3.2 to solve the incompressible Navier-Stokes equations can be interpreted as this type of splitting schemes. A general difficulty that we encounter here again is finding high order splitting methods of this flavor.

In the recent years, another paradigm has been designed to construct splitting methods with Runge-Kutta schemes, the so-called *additive Runge-Kutta (ARK) schemes*. They consist in tackling problems of the form (4.20), with partitioned methods composed of an arbitrary number of Runge-Kutta schemes. We saw at the beginning of section 4.1 that a general s-stage RK method can be completely identified by its set of real coefficients a_{ij}, b_i and c_i, with $i, j = 1, \ldots, s$. Let us now consider η s-stage RK methods, each with its own set of coefficients $a_{ij}^{[\iota]}$, $b_i^{[\iota]}$ and $c_i^{[\iota]}$, with $i, j = 1, \ldots, s$ and $\iota = 1, \ldots, \eta$. If problem (4.20) has an initial value $u(x_0) = u_0$, a general s-stage additive Runge-Kutta method to determine an approximation u_1 of $u(x_0 + \delta t)$, can be written as [KC03, AMSS97]:

$$k_i = u_0 + \delta t \sum_{\iota=1}^{\eta} \sum_{j=1}^{s} a_{ij}^{[\iota]} f^{[\iota]}\left(x_0 + c_j^{[\iota]} \delta t, k_j\right) \quad i = 1, \ldots, s \tag{4.25}$$

$$u_1 = u_0 + \delta t \sum_{\iota=1}^{\eta} \sum_{i=1}^{s} b_i^{[\iota]} f^{[\iota]}\left(x_0 + c_i^{[\iota]} \delta t, k_i\right) \tag{4.26}$$

The sound theoretical basis of Runge-Kutta methods is particularly appealing

Order	Number of coupling conditions
$p=3$	$\eta(\eta-1)(3\eta+4)/2!$
$p=4$	$\eta(\eta-1)(16\eta^2+22\eta+24)/3!$
$p=5$	$\eta(\eta-1)(125\eta^3+243\eta^2+334\eta+384)/4!$

Table 4.10: Number of coupling conditions as a function of the number of RK schemes η and the order p, for a general ARK scheme

when trying to design splitting methods, among other reasons because (i) they allow straigthforward ways to build high-order schemes, (ii) they permit an easy control of splitting errors and (iii) they offer tools for the stability analysis of additive Runge-Kutta schemes. However, the price paid is an increased complexity for the construction of efficient additive schemes. We will only illustrate this fact here, on the topic of the order conditions. Kennedy & Carpenter derived in [KC03] the order conditions for a general ARK method (4.26). It turns out that besides to the classical order conditions that each RK scheme in (4.26) must satisfy on order to be of order p, their are additional *coupling order conditions* that must be fulfilled by the coefficients $a_{ij}^{[\iota]}$, $b_i^{[\iota]}$ and $c_i^{[\iota]}$, with $i,j=1,\dots,s$ and $\iota=1,\dots,\eta$, for the ARK method to be of order p. The number of these coupling conditions grows dramatically with η and p. Table (4.10) gives the number of coupling conditions as a function of the number of RK schemes η and the order p, for a general ARK scheme [KC03]. These conditions must be met with $\eta\times s(s+1)$ Butcher coefficients. For the sake of completeness, we give here all the order conditions for an additive RK scheme with $\eta=2$ internal RK schemes, for $p=1,\dots,3$ [PR01].

First order:

$$\sum_i b_i^{[1]}=1,\quad \sum_i b_i^{[2]}=1$$

Second order:

$$\sum_i b_i^{[1]}c_i^{[1]}=1/2,\quad \sum_i b_i^{[2]}c_i^{[2]}=1/2,$$
$$\sum_i b_i^{[1]}c_i^{[2]}=1/2,\quad \sum_i b_i^{[1]}c_i^{[2]}=1/2$$

third order:

$$\sum_{ij} b_i^{[1]}a_{ij}^{[1]}c_i^{[1]}=1/6,\quad \sum_i b_i^{[1]}\left(c_i^{[1]}\right)^2=1/3,$$
$$\sum_{ij} b_i^{[2]}a_{ij}^{[2]}c_i^{[2]}=1/6,\quad \sum_i b_i^{[2]}\left(c_i^{[2]}\right)^2=1/3,$$

$$\sum_{ij} b_i^{[1]} a_{ij}^{[1]} c_i^{[2]} = 1/6, \quad \sum_{ij} b_i^{[2]} a_{ij}^{[1]} c_i^{[2]} = 1/6, \quad \sum_{ij} b_i^{[1]} a_{ij}^{[2]} c_i^{[2]} = 1/6,$$
$$\sum_{ij} b_i^{[2]} a_{ij}^{[1]} c_i^{[2]} = 1/6, \quad \sum_{ij} b_i^{[2]} a_{ij}^{[2]} c_i^{[1]} = 1/6, \quad \sum_{ij} b_i^{[2]} a_{ij}^{[1]} c_i^{[1]} = 1/6,$$

$$\sum_{i} b_i^{[1]} \left(c_i^{[2]}\right)^2 = 1/3, \quad \sum_{i} b_i^{[1]} c_i^{[1]} c_i^{[2]} = 1/3,$$
$$\sum_{i} b_i^{[2]} \left(c_i^{[1]}\right)^2 = 1/3, \quad \sum_{i} b_i^{[1]} c_i^{[2]} c_i^{[1]} = 1/3$$

We see here that up to order 3, and for ARK_η method with $\eta = 2$, if we have $b_i^{[1]} = b_i^{[2]}$ and $c_i^{[1]} = c_i^{[2]}$ for $i = 1, \ldots, s$, then *there are no coupling conditions*. This observation is actually more general and for an ARK_η, if the different internal RK schemes share the same $b_i^{[\iota]}$ and $c_i^{[\iota]}$, with $i = 1, \ldots, s$ and $\iota = 1, \ldots, \eta$, then up to and including order 3, there are no additional coupling conditions [KC03]. This property greatly simplify the construction of third-order ARK schemes. In what follows, we will restrict ourselves to the case $\eta = 2$, given the fact that we are concerned with the incompressible Navier-Stokes equations, and we saw earlier that these equations can be naturally split into two parts: a stiff linear part, and a nonstiff nonlinear part.

4.3.1 Implicit-explicit Runge-Kutta methods for Hessenberg index 2 DAEs

We are now concerned with the resolution of equation (4.20), with $\eta = 2$ parts: $f^{[1]}$ is a linear function with some degree of stiffness, and $f^{[2]}$ is a nonlinear function, that does not need special treatment due to stiffness. We want to use an ARK to solve this problem, and we consider the commbination of 2 s-stage RK methods. The method that will be used to solve the stiff part has to fullfill the different requirements established in section (4.2.1), because our goal is the resolution of the NS equations. At the very least, this method has then to be implicit. The method that will be used to solve the nonlinear part must have an easy implementation: in particular, we do not want to use a Newton solver, and this is only possible if we use an explicit method. We will resort to an implicit-explicit Runge-Kutta (IMEX-RK) scheme. Combining an implicit RK method and an explicit is really advantageous if we use a DIRK scheme for the IRK, because we decouple the computation of the different stages, and we do not have to use a nonlinear solver.

Ascher *et al.*[ARW95] were the first to propose IMEX RK methods of this type. They were trying to solve efficiently convection-diffusion equations, where the

0	0	0	...	0	0
c_2	a_{21}	γ	0	...	0
c_3	a_{31}	a_{32}	γ	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	...	$a_{s,s-1}$	γ
	b_1	b_2	...	b_{s-1}	b_s

Table 4.11: Butcher array of a general ESDIRK method

stiffness came from the linear diffusion processes. They designed a 4-stage, 3-order, L-stable and stiffly accurate ARK method, where the implicit part was a 3-stage SDIRK method, that was cast into a 4-stage DIRK by padding it with zeroes. Later Pareschi & Russo [PR01], while studying IMEX RK methods of this type, with special care for their accuracy and stability properties, showed that the 3-order ARK scheme of Ascher *et al.*could exhibit order reduction depending on the stiffness of the problem at hand. This is not surprising, given the fact that SDIRK methods have a stage order of 1, and we showed in section (4.2.1) that this parameter was crucial to obtain high-order schemes in the limit cases of Hessenberg index 2 DAEs.
We already know that we cannot obtain a 3-order scheme to solve this type of DAEs with a method exhibiting a stage order of 1, so we cannot use IMEX RK schemes which implicit part is an SDIRK. Hence the first task consists in finding DIRK methods with greater than one stage order.

4.3.2 Third-order ESDIRK schemes

The classical way to improve the stage order of SDIRK methods is to add a first explicit stage [KC16, WBCK02]. These methods were designated bythe name *ESDIRK*, which stands for Explicit Singly Diagonally Runge-Kutta. Their Butcher array thus takes the general form of table (4.11).
We want to use such a method to obtain a 3-order solution in the differential variable of a Hessenberg index 2 DAE. We already know that for an s-stage DIRK method, its order is at most $s + 1$, so we directly look for a 4-stage ESDIRK scheme, and we want it to be stiffly accurate, so that the method we are looking for takes the specific form of table (4.12).
Williams *et al.* derived in [WBCK02] the general solutions for such a method to be 3-order in the differential variable for both index 1 and index 2 DAEs,

0	0			
c_2	a_{21}	γ		
c_3	a_{31}	a_{32}	γ	
1	b_1	b_2	b_3	γ
	b_1	b_2	b_3	γ

Table 4.12: Butcher array of a 4-stage, stiffly accurate ESDIRK method

and 2-order accurate for the algebraic variable. If we denote:

$$\begin{aligned} \boldsymbol{c} &= (0, c_2, c_3, 1)^t, \quad \hat{\boldsymbol{c}} = (c_2, c_3, 1)^t \\ \boldsymbol{b} &= (b_1, b_2, b_3, \gamma)^t, \quad \hat{\boldsymbol{b}} = (b_2, b_3, \gamma)^t \\ \boldsymbol{A} &= (a_{ij})_{i,j=1,\dots,4}, \quad \hat{\boldsymbol{A}} = (a_{ij})_{i,j=2,\dots,4} \end{aligned}$$

then the order conditions our coefficients have to satisfy are [WBCK02]:

$$\boldsymbol{A}\boldsymbol{c}^{k-1} = \frac{\boldsymbol{c}^k}{k} \quad k = 1, 2 \tag{4.27a}$$

$$\boldsymbol{b}^t\boldsymbol{c}^{k-1} = \frac{1}{k} \quad k = 1, 2, 3 \tag{4.27b}$$

$$\hat{\boldsymbol{b}}^t\hat{\boldsymbol{A}}^{-1}\hat{\boldsymbol{c}}^{k-1} = 1 \quad k = 1, 2, 3 \tag{4.27c}$$

$$\hat{\boldsymbol{b}}^t\hat{\boldsymbol{A}}^{-2}\hat{\boldsymbol{c}}^{k-1} = k \quad k = 1, 2, 3 \tag{4.27d}$$

and the general solutions are given by (4.28).

$$a_{21} = \gamma, \quad c_2 = 2\gamma \tag{4.28a}$$

$$c_3 = \frac{2\gamma(\gamma - \frac{1}{4})(\gamma - 1)}{(\gamma - \frac{1}{2})^2 - \frac{1}{12}} \tag{4.28b}$$

$$a_{31} = c_3 - a_{32} - \gamma, \quad a_{32} = \frac{c_3(c_3 - 2\gamma)}{4\gamma} \tag{4.28c}$$

$$b_1 = 1 - b_2 - b_3 - \gamma \tag{4.28d}$$

0	0			
c_2	a_{21}			
c_3	a_{31}	a_{32}		
c_4	a_{41}	a_{42}	a_{43}	
	b_1	b_2	b_3	b_4

Table 4.13: Butcher array of a 4-stage, ERK method

$$b_2 = \frac{\frac{1}{3} - \gamma + \frac{1}{2}c_3 + \gamma c_3}{2\gamma(2\gamma - c_3)}, \quad b_3 = \frac{\frac{1}{3} - 2\gamma(1-\gamma)}{c_3(c_3 - 2\gamma)} \tag{4.28e}$$

where we assume that $c_3 \neq 0$, $c_3 \neq 2\gamma$ and of course $\gamma \neq 0$.
The stability function of such a method is:

$$R(z) = \frac{(-\gamma^3 + 3\gamma^2 - \frac{3}{2}\gamma + \frac{1}{6})z^3 + (3\gamma^2 - 3\gamma + \frac{1}{2})z^2 + (-3\gamma + 1)z + 1}{(1 - z\gamma)^3} \tag{4.29}$$

And we deduce from this that the method is A-stable if and only if $\gamma \in [\frac{1}{3}, \theta]$, where $\theta \simeq 1.06857902$ is the largest zero of the Laguerre polynomial $\frac{1}{24} - \frac{1}{2}x + \frac{3}{2}y^2 - y^3$ [WBCK02]. L-stability occurs at $\gamma \simeq 0.4358665215$. So we have some flexibility in the choice of the diagonal term, and we can play with this to construct high-order ARK methods for solving the incompressible Navier-Stokes equations.

4.3.3 Derivation of the explicit Runge-Kutta scheme for the convection

Our last task is to find the corresponding ERK scheme for the convective part in the NS equations. The general Butcher array of a 4-stage ERK scheme is given by table (4.13).
Hairer *et al.*derived in [HNW87] the equations that such coefficients have to be solutions to in order for the resulting ERK scheme to be of order 4. We report them here (4.30).

$$b_1 + b_2 + b_3 + b_4 = 1 \tag{4.30a}$$

$$b_2c_2 + b_3c_3 + b_4c_4 = 1/2 \tag{4.30b}$$

$$b_2c_2^2 + b_3c_3^2 + b_4c_4^2 = 1/3 \tag{4.30c}$$

$$b_2c_2^3 + b_3c_3^3 + b_4c_4^3 = 1/4 \tag{4.30d}$$

$$b_3c_3a_{32}c_2 + b_4c_4(a_{42}c_2 + a_{43}c_3) = 1/8 \tag{4.30e}$$

$$b_3a_{32} + b_4(a_{42} = b_2(1 - c_2) \tag{4.30f}$$

$$b_4a_{43} = b_3(1 - c_3) \tag{4.30g}$$

$$0 = b_4(1 - c_4) \tag{4.30h}$$

$$c_i = \sum_{j=1}^{i-1} a_{ij} \quad i = 1, \ldots, 4 \tag{4.30i}$$

We would like to use the same c_i and b_i coefficients corresponding to the ESDIRK method, in order to have zero coupling conditions for the corresponding ARK method to attain a third order scheme for ODEs. This is made possible by the following proposition:

Proposition 4.3.1. *Let $(b_i)_{i=1,\ldots,4}$, $(c_i)_{i=2,\ldots,4}$, and a_{21}, a_{31}, a_{32}, a_{41}, a_{42} and a_{43}, be the real coefficients of a general 4-stage ERK method with:*

1. *$b_4 = \gamma \neq 0$, $c_4 = 1$, c_2 and c_3 are the functions of γ given respectively by (4.28a, 4.28b), and we assume that $c_3 \neq 0$, $c_3 \neq 1$ and $c_3 \neq \gamma$.*
2. *b_1, b_2 and b_3 are the functions of γ given by (4.28d, 4.28e)*
3. $a_{43} = \frac{b_3(1-c_3)}{\gamma}$
4. $a_{32} = \frac{1}{b_3\gamma c_2(1-c_3)}\left((c_2b_2(1-c_2) - \frac{1}{8})\gamma + a_{43}c_3\gamma^2\right)$
5. $a_{42} = \frac{1}{b_3\gamma c_2(1-c_3)}\left(-b_3c_3c_2b_2(1-c_2) - a_{43}c_3b_3\gamma + \frac{1}{8}b_3\right)$
6. $a_{21} = c_2$
7. $a_{31} = c_3 - a_{32}$
8. $a_{41} = 1 - a_{43} - a_{42}$

Then this ERK method is 4^{th}-order.

Proof. The first two items mean that the b_i and c_i coefficients satisfy the order conditions (4.27). In particular, they satisfy the conditions (4.27b), which are exactly the conditions (4.30a–4.30d).
$c_4 = 1$, so that condition (4.30h) is also satisfied. The expression of a_{43} in item 3 above shows that condition (4.30g) is satisfied.
Once coefficient a_{43} is determined, the conditions (4.30e, 4.30f) are a linear system for the unknowns a_{32} and a_{42}; the determinant of this system is given by $b_3\gamma c_2(1-c_3)$ which is different from zero by item 1. This system then has a unique solution, which is given by items 4 and 5. Finally, by items 6, 7 and 8, conditions (4.30i) are also met, which concludes the proof.

□

Thus we can build a third order in the velocity implicit-explicit Runge Kutta scheme with a 4-stage ARK method. If we apply such a method to the equations (3.1), we end up with the resolution of 3 linear systems, whose matrix is the same (in the case $d = 2$ as in section (4.2.2)):

$$\begin{pmatrix} \frac{1}{\delta t}\Gamma - \gamma\nu L_1 & 0 & -\gamma D_1^t \\ 0 & \frac{1}{\delta t}\Gamma - \gamma\nu L_2 & -\gamma D_2^t \\ D_1 & D_2 & 0 \end{pmatrix} \tag{4.31}$$

Again, a saddle-point matrix.

4.3.4 Numerical simulations

After different trials and errors, we found that the ESDIRK scheme proposed by Williams *et al.* in [WBCK02], when combined to an ERK method built with the features explicited in (4.3.1), produces an ARK method that reaches 3^{rd}-order accuracy for the velocity. We denote this ESDIRK scheme $ESDIRK3(2I)4SA$, following the nomenclature of [KC16]: it is a 3^{rd}-order method in the differential variables, and at least 2^{nd}-order in the algebraic variable when applied to a index 2 Hessenberg DAE, which is stiffly accurate and A-stable, with a stage-order of 2. Its diagonal term is $\gamma = \frac{1}{2}$. Table (4.14) gives the coefficients of this method, and Table (4.15) gives the coefficients of the associated ERK method. We denote our new ARK method $ARK - ESDIRK3(2I)4SA$.
We check the order of this ARK scheme with the lid-driven cavity case described in section (4.2.3). We compute a quasi-exact solution at time $t = 5.0$ with a very small timestep (with the ARK method), we compare this solution with solutions obtained at the same final time, but with larger timesteps. All the computations are made on a uniform grid, with a grid level of 7 (128×128). Figures (4.8), (4.9) and (4.10) represent the order of accuracy of the $ARK - ESDIRK3(2I)4SA$ method for the horizontal component of the velocity, the vertical one and the pressure, respectively. We reach 3^{rd}-order accuracy for both the velocity and the pressure in this case.

0	0			
1	$\frac{1}{2}$	$\frac{1}{2}$		
$\frac{3}{2}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{1}{2}$	
1	$\frac{7}{18}$	$\frac{1}{3}$	$-\frac{2}{9}$	$\frac{1}{2}$
	$\frac{7}{18}$	$\frac{1}{3}$	$-\frac{2}{9}$	$\frac{1}{2}$

Table 4.14: Butcher array for the $ESDIRK3(2I)4SA$ method

0	0			
1	1			
$\frac{3}{2}$	$\frac{9}{8}$	$\frac{3}{8}$		
1	$\frac{11}{18}$	$\frac{3}{18}$	$\frac{2}{9}$	
	$\frac{7}{18}$	$\frac{1}{3}$	$-\frac{2}{9}$	$\frac{1}{2}$

Table 4.15: Butcher array for the $ERK - ESDIRK3(2I)4SA$ method

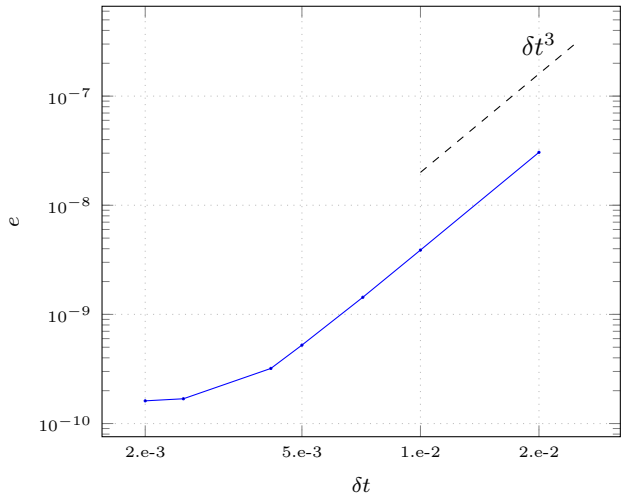


Figure 4.8: Lid-driven cavity. Temporal order of accuracy for the $ARK - ESDIRK3(2I)4SA$ method; horizontal component of the velocity

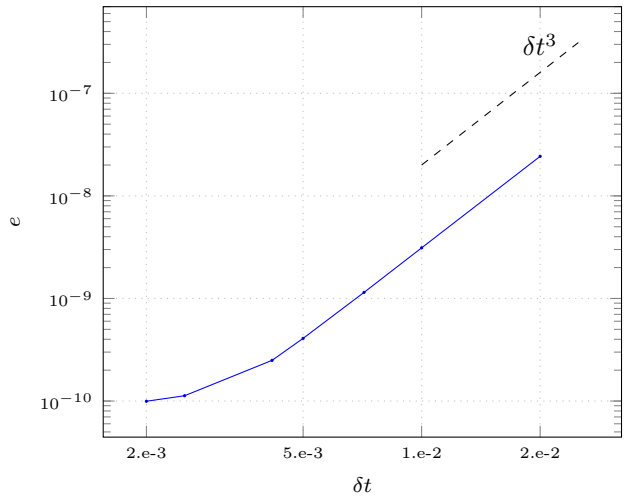


Figure 4.9: Lid-driven cavity. Temporal order of accuracy for the $ARK - ESDIRK3(2I)4SA$ method; vertical component of the velocity

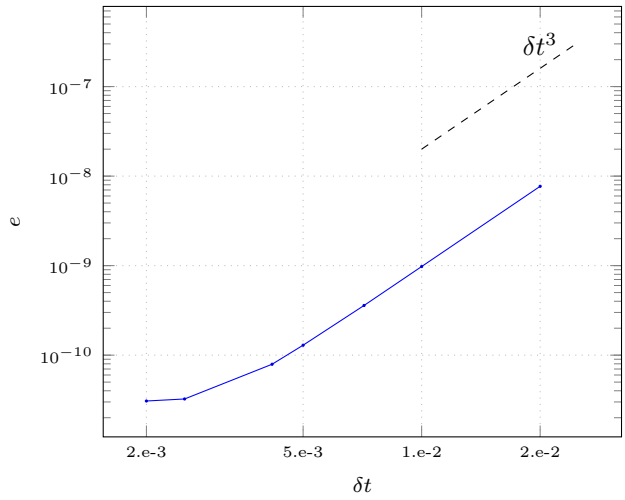


Figure 4.10: Lid-driven cavity. Temporal order of accuracy for the $ARK - ESDIRK3(2I)4SA$ method; pressure

4.4 Half-explicit Runge-Kutta methods

4.4.1 HERK schemes for Hessenberg index 2 DAEs

The last numerical method to integrate the semi-discretized Navier-Stokes equations that we experimented takes even further the idea of treating explicitly the terms in the equations that are not directly responsible for their DAE structure. Hessenberg index 2 DAE systems (3.5a, 3.5b) have a semi-explicit structure that allows to treat the differential variable in an explicit manner, and the algebraic variable in an implicit manner. In our case, it means that we treat the convection *and* the diffusion in the momentum equations in an explicit form, and only treat the pressure in an implicit form. These type of RK methods are called *half-explicit Runge-Kutta methods* [HLR89], because they rely on ERK, but with a slight modification to treat the algebraic variable implicitly. Hairer, Lubich and Roche proposed in [HLR89] the following implementation of the method to the problem (3.5a, 3.5b), to obtain u_1 and v_1 from initial consistent values u_0 and v_0:

$$U_i = u_0 + \delta t \sum_{j=1}^{i-1} a_{ij} f(U_j, V_j), \quad i = 1, \dots, s \tag{4.32a}$$

$$0 = g(U_i) \tag{4.32b}$$

$$u_1 = u_0 + \delta t \sum_{i=1}^{s} b_i f(U_i, V_i) \tag{4.32c}$$

$$0 = g(u_1) \tag{4.32d}$$

The value v_1 can be obtained from the hidden constraint (3.5c)
U_1 is u_0, and satisfies (4.32b) because we assumed that the initial values were consistent. From then we insert U_2 in (4.32b), which gives us a nonlinear equation for V_1, that has a unique solution if $a_{21} \neq 0$, and the assumption that $g_u(u) f_v(u, v)$ is nonsingular for all x is satisfied. We obtain V_1 and u_2, and we use the same procedure to compute the next stages. Existence and uniqueness of the solution are ensured more generally if $a_{i,i-1} \neq 0$ for $i = 2, \dots, s$, $b_s \neq 0$ and the usual assumption about $g_u(u) f_v(u, v)$ is satisfied [HW96].
In this way, we can apply any ERK methods to solve the NS equations, but of course we have additional order conditions to obtain high-order methods. The main result of convergence was established for example in [HLR89], and it says the following:

Theorem 4.4.1 (Convergence for the u-component)**.** *Suppose that $g_u(u) f_v(u, v)$ is nonsingular in a neighborhood of the solution $(u(x), v(x))$ of (3.5a, 3.5b), and that the initial values are consistent. If the Runge-Kutta coefficients are such*

0	0		
$\frac{1}{3}$	$\frac{1}{3}$		
$\frac{2}{3}$	0	$\frac{2}{3}$	
	$\frac{1}{4}$	0	$\frac{3}{4}$

Table 4.16: Butcher array for the Heun method

that $a_{i,i-1} \neq 0$ *for* $i = 2, \ldots, s$ *and* $b_s \neq 0$*, and the local error satisfy:*

$$\delta u = \mathcal{O}(\delta t^r), \quad P(x)\delta u = \mathcal{O}(\delta t^{r+1})$$

with $P(x)$ *given by (4.15), then the method (4.32) is convergent of order* r*, , i.e.:*

$$u_n - u(x_n) = \mathcal{O}(\delta t^r), \quad for \quad x_n = n\delta t \leq Const.$$

The only delicate part in the application of such methods to the NS equations is that we have to solve s linear systems for the pressure, with the same matrix $D_1\Gamma^{-1}D_1^t + D_2\Gamma^{-1}D_2^t$ (in the case $d = 2$). This linear system is a variant of the classical Poisson equation, and in our case it is much easier to invert this matrix than to invert the matrices of the two preceding methods. But since we treat explicitly the diffusion, we have more stringent stability constraints than for the Radau IIA and the IMEX RK schemes, and we will have to determine for each particular case study which one of these 3 methods is the more efficient in terms of global computational effort.

4.4.2 Numerical simulations

We test here the convergence properties of the 3-stage, 3^{rd}-order ERK method designed by Heun, and which coefficients are given by Table (4.16).
We check the order of the Heun method with the lid-driven cavity case described in section (4.2.3). We compute a quasi-exact solution at time $t = 5.0$ with a very small timestep (with the ARK method), we compare this solution with solutions obtained at the same final time, but with larger timesteps. All the computations are made on a uniform grid, with a grid level of 7 (128×128). Figures (4.11), (4.12) and (4.13) represent the order of accuracy of the Heun method for the horizontal component of the velocity, the vertical one and the pressure, respectively. We reach 3^{rd}-order accuracy for both the velocity and the pressure in this case.

Thus we end up with three numerical integration schemes that are 3^{rd}-order in the velocity variables when applied to the incompressible Navier-Stokes equations. They have very different implementation specificities, and depending

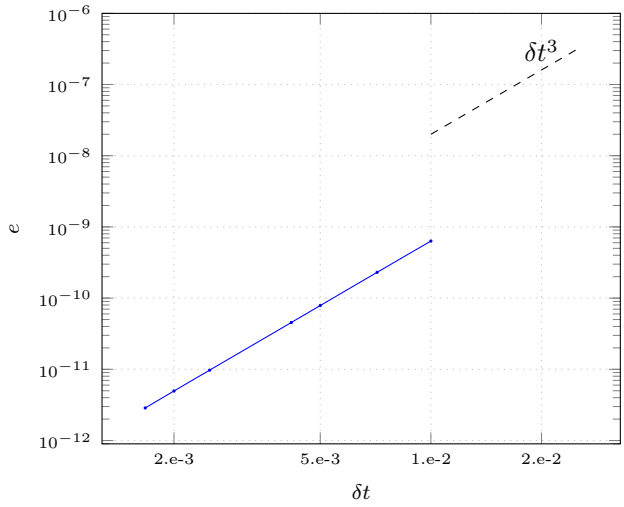


Figure 4.11: Lid-driven cavity. Temporal order of accuracy for the herk method; horizontal component of the velocity

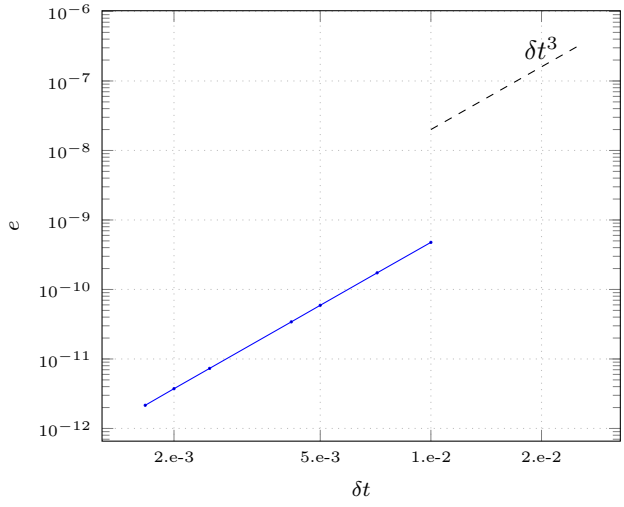


Figure 4.12: Lid-driven cavity. Temporal order of accuracy for the herk method; vertical component of the velocity

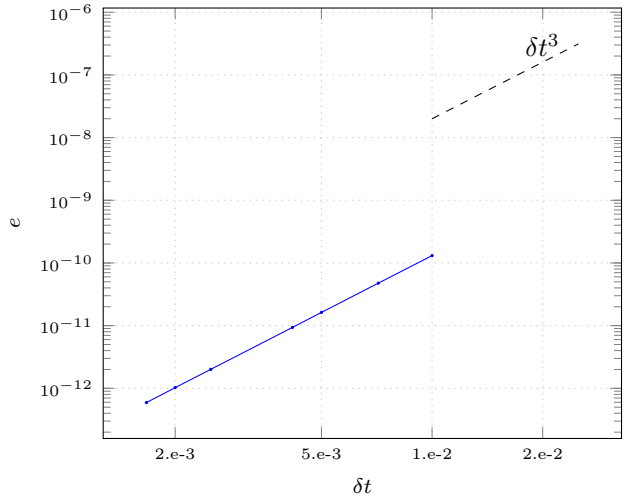


Figure 4.13: Lid-driven cavity. Temporal order of accuracy for the herk method; pressure

on the application, one could prefer a specific scheme to the others. We will give some elements regarding the comparison of these three schemes in Chapter 6.

Chapter 5

Description of the space adaptive multiresolution implementation in the mrpy code

We will describe in this chapter the practical implementation of the adaptive multiresolution strategy presented in Chapter 1. We will only mention three of the general issues we need to overcome for a proper implementation of the multiresolution analysis. First, since we eventually end up with non-uniform adaptive grids, it is more challenging to obtain the mesh connectivity, that is, to determine the neighboring cells of each cell in each direction. Then, the multiresolution analysis requires inter-level operations between meshes at different grid levels, so that our data storage solution should be able to give us access to the tree-structured data of nested grids described in section (1.3.3). Finally, we need an easy way to add and delete cells as the grid moves to adapt to the numerical solution of the PDE at hand. We chose to solve these problems by encoding the nested grids into a space-filling curve, and we store the cells data with hashtables. We implement this strategy with the `dict` data-structure object of Python. We will give more details regarding the code structure at the end of the chapter, but we start with the desciption of the multiresolution operations, assuming that the data structure gives us full access to the information needed to perform them. The algorithms described here are largely based on a tutorial that have been elaborated for a Summer School of CNRS *GDR Groupe Calcul*, on Multiresolution and Adaptive Mesh Refinement Methods, Fréjus, France (2010) [TD11a].

5.1 Multiresolution Operations

We assume that we want to perform the adaptive multiresolution strategy on a function $u(t, \boldsymbol{x}) : \mathbb{R} \times \Omega \to \mathbb{R}$, that depends on the time variable t, and is defined for $\boldsymbol{x} \in \Omega$, an open bounded domain of $\mathbb{R}^d$, where the space dimension d can take the values 1, 2 or 3. For simplicity, we will only consider embedded Cartesian grids, and we assume that Ω is either an interval in $1D$, has rectangular shape in $2D$ or parallelepipedic shape in $3D$. We define L to be the finest grid level necessary to properly approximate u by piecewise constant functions on Ω, and for grid levels $l = 0, 1, \ldots, L$, we define the partition Ω_l of Ω by 2^{dl} meshes K_γ^l of equal size, as in section (1.1.1). The meshes K_γ^l are the control volumes in the Finite-Volume terminology. The partition is designed so that the set of grids possesses the ladder property: for $l \neq L$, for each cell $K_\gamma^l \in \Omega_l$ there exists a unique set of 2^d cells $K_\mu^{l+1} \in \Omega_{l+1}$ so that:

$$\overline{K_\gamma^l} = \overline{\bigsqcup K_\mu^{l+1}}$$

We can associate to each grid Ω_l a subspace V_l of $L_2(\Omega)$ consisting of the piecewise constant functions on this partition. Then we denote by $U_l := (u_\gamma^l)$ the best approximation of u in V_l for the canonical norm of $L_2(\Omega)$:

$$u_\gamma^l := |K_\gamma^l|^{-1} \int_{K_\gamma^l} u(t, \boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{5.1}$$

where:

$$|K_\gamma^l| = \int_{K_\gamma^l} \mathrm{d}\boldsymbol{x}, \qquad \boldsymbol{x} \in \mathbb{R}^d$$

is the Lebesgue measure of K_γ^l. We saw in section (1.3.3) that we can associate to the multiresolution transform of u a (graded) tree Λ whose nodes are the cells K_γ^l necessary to obtain the multiresolution analysis of u. The root of this tree is the node at level $l = 0$, and for each node K_γ^l with $l \neq L$, its children are the nodes 2^d cells $K_\mu^{l+1} \in \Omega_{l+1}$ that form a partition of K_γ^l. We denote by $\mathcal{C}_\gamma^l$ the index set of the children-cells of K_γ^l, and we denote by I_l the index set of the nodes in Λ that belong to the level l. The leaves $L(\Lambda)$ of the tree are, at any time in the computation process, a hybrid partition of Ω that is the adaptive grid generated by the adaptive multiresolution strategy that best fits u. We denote by $U_{L(\Lambda)}$ the piecewise approximation of u on $L(\Lambda)$.

5.1.1 Projection and prediction operators

The first operations to perform the multiresolution transform are the projection and the prediction. We consider that we initially have the approximation of u on the leaves of the tree, which is the most precise approximation on our grid.

We first compute the cell-averages of u on the successive coarser grids, until we reach the root cell Ω_0. For K_γ^l a cell that is not a leaf, if we know the values u_μ^{l+1} on the cells K_μ^{l+1} for $\mu \in \mathcal{C}_\gamma^l$, we can compute u_γ^l by the following formula:

$$P_{j-1}^j : u_\gamma^l = \frac{1}{|K_\gamma^l|} \sum_{\mu \in \mathcal{C}_\gamma^l} |K_\mu^{l+1}| u_\mu^{l+1} \tag{5.2}$$

Then, for each cell K_γ^l that is not a leaf, we compute the *predicted* values $\hat{u}_\mu^{l+1}$ for $\mu \in \mathcal{C}_\gamma^l$ with the centered polynomial interpolations defined in Chapter 1, depending on their accuracy order $N = 2M+1$ that can be chosen by the user. We will recall briefly here the different formulas. The index γ of the cell K_γ^l on the grid Ω_l is an integer k if $d = 1$, a pair (j,k) of integers if $d = 2$, and a triplet (i,j,k) of integers if $d = 3$. The predited values are then:

$$\left.\begin{aligned} \hat{u}_{2k}^{l+1} &= u_{k_1}^l + \sum_{d_1=1}^{M} \xi_{d_1} \left(u_{k+d_1}^l - u_{k-d_1}^l \right), \\ \hat{u}_{2k+1}^{l+1} &= u_{j,k_1} - \sum_{d_1=1}^{M} \xi_{d_1} \left(u_{k+d_1}^l - u_{k-d_1}^l \right), \end{aligned} \right\} \tag{5.3}$$

in $1D$, by:

$$\begin{aligned} \hat{u}_{2j+p,2k+q}^{l+1} = u_{j,k}^l &+ (-1)^p Q^M(j, u_{\cdot,k}^l) + (-1)^q Q^M(k, u_{j,\cdot}^l) \\ &- (-1)^{(p+q)} Q_2^M(j,k,u_{j,k}^l), \end{aligned} \tag{5.4}$$

in $2D$, and by:

$$\begin{aligned} \hat{u}_{2i+p,2j+q,2k+r)}^{l+1} = u_{i,j,k}^l &+ (-1)^p Q^M(i, u_{\cdot,j,k}^l) + (-1)^q Q^M(j, u_{i,\cdot,k}^l) \\ &+ (-1)^r Q^M(k, u_{i,j,\cdot)}^l) \\ &- (-1)^{(p+q)} Q_2^M(i,j,u_{\cdot,\cdot,k}^l) \\ &- (-1)^{(p+r)} Q_2^M(i,k,u_{\cdot,j,\cdot}^l) \\ &- (-1)^{(q+r)} Q_2^M(j,k,u_{i,\cdot,\cdot}^l) \\ &+ (-1)^{(p+q+r)} Q_3^M(i,j,k,u_{i,j,k}^l), \end{aligned} \tag{5.5}$$

in $3D$, where the coefficients ξ are taken from the table (1.1), and the expressions Q^M, Q_2^M and Q_3^M are given respectively by the equations (1.65), (1.67)

and (1.68).
With the projection and prediction inter-level operators, we can *encode* the multiresolution transform $\mathcal{M}$ defined by (1.52), to the finest representation u on the leaves of Λ. It is a change of basis, that gives us the details vectors. Algorithm 5.1 describes how we do this transformation.

Algorithm 5.1 Encoding by multiresolution transform $\mathcal{M}$

1: **Input:** $U_{L(\Lambda)}$ given by cell-averaged values u_γ^l such that the cells K_γ^l are in $L(\Lambda)$.
2: **for** $l = L-1 \to 0$ **do**
3: **for** $\gamma \in \mathrm{I}_l$ s.t. K_γ^l is not a leaf **do**
4: Compute the cell-average u_γ^l at grid level l, from the values u_μ^{l+1} for $\mu \in \mathcal{C}_\gamma^l$ by using the projection operator P_{j-1}^j (5.2).
5: Compute the predicted values $\hat{u}_\mu^{l+1}$ for $\mu \in \mathcal{C}_\gamma^l$ by the polynomial interpolations (5.3), (5.4), or (5.5), and the corresponding details defined by (1.48): $d_\mu^{l+1} = u_\mu^{l+1} - \hat{u}_\mu^{l+1}$.
6: Save details in the array $\mathbf{D}_\gamma^l$
7: Encode the solution by replacing $(u_\mu^{l+1})_{\mu \in \mathcal{C}_\gamma^l}$ by $(u_\gamma^l, \mathbf{D}_\gamma^l)$.
8: **end for**
9: **end for**
10: **Output:** $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}_\gamma^0, \mathbf{D}_\gamma^1, \ldots, \mathbf{D}_\gamma^{L-1})$

With this new data representation on the wavelet space, the details in $\mathbf{D}_\gamma^l$ account for the local spatial smoothness in the solution. A *decoding* procedure is necessary to retrieve the representation on the physical space of the variables. The latter is done by means of the inverse multiresolution transform $\mathcal{M}^{-1}$, following the Algorithm 5.2.

Algorithm 5.2 Decoding by inverse multiresolution $\mathcal{M}^{-1}$

1: **Input:** $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}_\gamma^0, \dots, \mathbf{D}_\gamma^{L-1})$ of size $\#(L(\Lambda))$, given by the representation on the coarsest grid: u^0, and the set of detail arrays: $(\mathbf{D}_\gamma^0, \dots, \mathbf{D}_\gamma^{L-1})$.

2: **for** $l = 0 \to L-1$ **do**
3: **for** $\gamma \in \mathrm{I}_l$ s.t. K_γ^l is not a leaf **do**
4: Compute for each u_γ^l, 2^d predicted $\hat{u}_\mu^{l+1}$ such that $\mu \in \mathcal{C}_\gamma^l$, by the polynomial interpolations (5.3), (5.4), or (5.5), and the corresponding u_μ^{l+1} by (1.48): $u_\mu^{l+1} = \hat{u}_\mu^{l+1} + d_\mu^{l+1}$.
5: Save u_μ^{l+1} in the array U_{l+1}.
6: Decode the solution by replacing $(u_\gamma^l, \mathbf{D}_\gamma^l)$ by $(u_\mu^{l+1})_{\mu \in \mathcal{C}_\gamma^l}$.
7: **end for**
8: **end for**
9: **Output:** $U_{L(\Lambda)}$ of size $\#(L(\Lambda))$, given by cell-averaged values u_γ^l s.t. the cells K_γ^l are in $L(\Lambda)$

5.1.2 Thresholding and predictive refinement

Once we have encoded all the details by the multiresolution transform of u, we perform the thresholding operation defined in section (1.3.3). We first set a thresholding parameter ε, (this parameter is completely user-defined), and the goal of this operation is to build a set $\Lambda_\varepsilon \subset \Lambda$, obtained by deleting nodes in Λ according to a thresholding process as (1.70). If we denote by $\mathcal{T}_{\Lambda_\varepsilon}$ the truncation operator that corresponds to this thresholding process, we then build the approximation $\mathcal{A}_{\Lambda_\varepsilon} U_{L(\Lambda)}$ of $U_{L(\Lambda)}$, where

$$\mathcal{A}_{\Lambda_\varepsilon} := \mathcal{M}^{-1} \mathcal{T}_{\Lambda_\varepsilon} \mathcal{M}.$$

so that:

$$\|U_{L(\Lambda)} - \mathcal{A}_{\Lambda_\varepsilon} U_{L(\Lambda)}\|_{L^2} \leq C\varepsilon, \tag{5.6}$$

For $K_\gamma^l \in \Lambda$ which is not a leaf, we define the detail vector $\boldsymbol{d}_\gamma^l := (d_\mu^{l+1})_{\mu \in \mathcal{C}_\gamma^l}$, and we introduce the following discrete ℓ^2-norm:

$$\|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)} = 2^{-d/2} \left(\sum_{\mu \in \mathcal{C}_\mu^l} (d_\mu^{l+1})^2 \right)^{1/2} \tag{5.7}$$

Next, we define the level-dependent threshold values ε_l, for $0 \leq l \leq L-1$, by:

$$\varepsilon_l = 2^{\frac{d}{2}(l-L)}\varepsilon, \quad l \in [1, L], \tag{5.8}$$

and we define the truncation operator $\mathcal{T}_{\Lambda_\varepsilon}$, for a node $K_\gamma^l \in \Lambda$, by:

$$\text{for} \quad \mu \in \mathcal{C}_\gamma^l, \quad K_\mu^{l+1} \in \Lambda_\varepsilon \quad \text{if} \quad \|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)} \geq \varepsilon_l \tag{5.9}$$

It can be shown that with such a truncation operator, the approximation $\mathcal{A}_{\Lambda_\varepsilon} U_{L(\Lambda)}$ of $U_{L(\Lambda)}$ verifies (5.6) (see for example [Dua11]). Now if we draw an analogy with the AMR methods presented in section (1.1.2), we can see that we have built a *coarsen* operation. What about the *refine* operation?
We will consider that $u(t, \boldsymbol{x})$ is solution to a general Partial Differential Equation, that can be written in the following form:

$$\frac{\partial u}{\partial t} = F(u) \tag{5.10}$$

We discretise the timeline into time steps (for example with $t^n = n\,\delta t$, where δt is a fixed stepsize), and we denote by $(E_{\delta t})$ the discrete evolution operator relating the solution at two consecutive time steps: $u^{n+1} = E_{\delta t}\, u^n$. Let $\Lambda_\varepsilon^{n+1}$ be the set of nodes obtained by applying the truncation operator (5.9) to the linear approximation $U_{\Omega_L}^{n+1}$ of u^{n+1} on the finest grid. If the solution slowly evolves from one time step to another, then we can assume that $\Lambda_\varepsilon^{n+1}$ is *close* to Λ_ε^n. More specifically, we make the hypothesis that u^{n+1} might require finer meshes that u^n in some part of the domain, but only by one level. This means that we might need to refine some of the leaves in Λ, but only by one level. We will perform this operation by using one of Harten's heuristics [Har94b, Har94a, Har95]. Let K_γ^l be a leaf of Λ, so that $l \neq L$. We could decide whether this node needs refinement by comparing $\|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)}$, computed with the mean values of u^{n+1}, to ε_l. But we do not have this information, by definition. So we are going to approximate $\|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)}$ by $2^{-p}\|\boldsymbol{d}_\nu^{l-1}\|_{\ell^2(K_\nu^{l-1})}$, where K_ν^{l-1} is the parent-cell of K_γ^l, and p is a user-defined parameter, that is generally set to $p = 2M + 2$, where M is the prediction stencil used in the polynomial interpolations (5.3), (5.4), or (5.5) [TD11a, Dua11, CKMP03]. We then enlarge Λ_ε with the following criterion. For a node $K_\gamma^l \in \Lambda$, so that K_γ^l is a leaf and $l \neq L$:

$$\text{for} \quad \mu \in \mathcal{C}_\gamma^l, \quad K_\mu^{l+1} \in \Lambda_\varepsilon \quad \text{if} \quad \|\boldsymbol{d}_\nu^{l-1}\|_{\ell^2(K_\nu^{l-1})} \geq 2^{2p}\varepsilon_l \tag{5.11}$$

where K_ν^{l-1} is the parent-cell of K_γ^l. If needed, we re-create the cells K_μ^{l+1}, and we compute their values from the polynomial interpolations (5.3), (5.4), or (5.5), applied to the mean values of u^n. Algorithm (5.3) illustrates the two operations described above. We do not suppress the cells that should be discarded by (5.9)

at this point. Instead, for each cell $K_\gamma^l \in \Lambda$, we introduce a binary flag t_γ^l which indicates whether the cell K_γ^l is kept throughout the adaptive multiresolution process. Initially, $t_\gamma^l =$ `.false.`, except for $l = 0$, *i.e.*$t_\gamma^0 =$ `.true.`.

Algorithm 5.3 Thresholding and predictive refinement.

Input: $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}_\gamma^0, \ldots, \mathbf{D}_\gamma^{L-1})$ of size $\#(L(\Lambda))$, given by the representation on the coarsest grid: u^0, and the set of detail arrays: $(\mathbf{D}_\gamma^0, \ldots, \mathbf{D}_\gamma^{L-1})$.

We choose a value of the threshold parameter ε;
We initiate a binary flag, t_γ^l, that marks if the node K_γ^l belongs or not to the set Λ_ε:

- the root belongs to Λ_ε, $t^0 =$ `.true.`;
- all the other cells are set to `.false.`: $t_\gamma^l =$ `.false.`, $\forall l \in [1, L]$ and $\forall \gamma \in \mathrm{I}_l$

for $l = L - 1$ down to 0 **do**
  Evaluation of the level-dependent threshold value $\varepsilon_l = 2^{\frac{d}{2}(l-L)}\varepsilon$
  for $\gamma \in \mathrm{I}_l$ s.t. K_γ^l is not a leaf **do**
    if $\|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)} \geq \varepsilon_l$ **then**
      for $\mu \in \mathcal{C}_\gamma^l$ **do**
        $t_\mu^{l+1} =$ `.true.` $\Rightarrow K_\mu^{l+1} \in \Lambda_\varepsilon$.
      end for
    end if
    if $\|\boldsymbol{d}_\gamma^l\|_{\ell^2(K_\gamma^l)} \geq 2^{(2p)}\varepsilon_l$ **then**
      for $\mu \in \mathcal{C}_\gamma^l$ **do**
        $t_\mu^{l+1} =$ `.true.` $\Rightarrow K_\mu^{l+1} \in \Lambda_\varepsilon$.
        for $\upsilon \in \mathcal{C}_\mu^{l+1}$ **do**
          $t_\upsilon^{l+2} =$ `.true.` $\Rightarrow K_\upsilon^{l+2} \in \Lambda_\varepsilon$.
        end for
      end for
    end if
  end for
end for

5.1.3 Graduation and pruning

The two last operations are the graduation of Λ_ε and its subsequent pruning. We want to build a tree $\tilde{\Lambda}_\varepsilon$, containing Λ_ε, so that for each cell $K_\gamma^l \in \tilde{\Lambda}_\varepsilon$, the cells in its prediction stencil $R_{K_\gamma^l}$ are also in $\tilde{\Lambda}_\varepsilon$. We recall for $l \neq 0$, the prediction stencil is composed of all the cells needed to perform the polynomial interpolation in the cell K_γ^l, by formulas (5.3), (5.4), or (5.5). The algorithmic implementation is straightforward, and is given by algorithm 5.4.

Algorithm 5.4 Graduation of the tree structure

Input: $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}^0_\gamma, \ldots, \mathbf{D}^{L-1}_\gamma)$ of size $\#(L(\Lambda))$, given by the representation on the coarsest grid: u^0, and the set of detail arrays: $(\mathbf{D}^0_\gamma, \ldots, \mathbf{D}^{L-1}_\gamma)$. We assume that the binary flags t^l_γ have alerady been initialized and modified (by the thresholding and refinement processes for example).
for $l = L-1$ down to 0 **do**
  for $\gamma \in \mathrm{I}_l$ **do**
    if $t^{l+1}_\mu =$ `.true.` for any $\mu \in \mathcal{C}^l_\gamma$ **then**
      for $K^l_\nu \in R_{K^{l+1}_\mu}$ **do**
        $t^l_\nu =$ `.true.` $\Rightarrow K^l_\nu \in \tilde{\Lambda}_\varepsilon$
      end for
    end if
  end for
end for

We add that if new cells are created by the graduation, we approximate their *real* value by the polynomial interpolation, as we do for the predictive refinement. Eventually, the final step summarized in Algorithm 5.5 deletes completely all cells that are not included in the thresholded, refined and graded tree $\tilde{\Lambda}_\varepsilon$.

Algorithm 5.5 Pruning of superfluous cells.

Input: $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}^0_\gamma, \ldots, \mathbf{D}^{L-1}_\gamma)$ of size $\#(L(\Lambda))$, given by the representation on the coarsest grid: u^0, and the set of detail arrays: $(\mathbf{D}^0_\gamma, \ldots, \mathbf{D}^{L-1}_\gamma)$. We assume that the binary flags t^l_γ have alerady been initialized and modified (by the thresholding, refinement and processes for example).
for $l = L$ down to 1 **do**
  for $K^l_\gamma \in \tilde{\lambda}_\varepsilon$ **do**
    if $t^l_\gamma =$ `.false.` **then**
      Delete K^l_γ.
    end if
  end for
end for
Output: $\mathbf{M}_{L(\Lambda)} = (u^0, \mathbf{D}^0_\gamma, \mathbf{D}^1_\gamma, \ldots, \mathbf{D}^{L-1}_\gamma)$ of size $\#(L(\tilde{\Lambda}_\varepsilon$

5.2 A new multiresolution scheme for incompressible flows

The goal of this part is to present the global strategy we designed in order to couple the fully adaptive finite volume scheme [CKMP03] to a resolution of the incompressible Navier-Stokes equations. We already built the spatial discretization scheme required to this task on hybrid grids as the ones produced by the adpative multiresolution in chapter 1. We also built high-order Runge-Kutta methods able to efficiently tackle the DAE resulting from the spatial discretization of the incompressible flow equations in chapter 4. Before presenting the full solver, we mention some of the difficulties that appear when we apply the multiresolution strategy in this context, and how we overcame them in the mrpy code.

5.2.1 Data Initialization

We start by recalling the general problem setup. We consider an incompressible flow described by the velocity vector $\boldsymbol{u}(\boldsymbol{x},t) = (u_i(\boldsymbol{x},t))_{i=1,\dots,d}$ and the pressure field $p(\boldsymbol{x},t)$. The time variable t varies between 0 and T. The space variable $\boldsymbol{x}$ belongs to an open-bounded domain $\Omega \in \mathbb{R}^d$. We consider that Ω has a rectangular (*resp.* rectangular parallelepiped) domain, $\Omega =]0, b_x[\times]0, b_y[$ in $2D$ (*resp.* $\Omega =]0, b_x[\times]0, b_y[\times]0, b_z[$ in 3D, with $(b_x, b_y, b_z) \in \mathbb{R}_+^*$). We denote $\partial\Omega = \overline{\Omega}\backslash\Omega$ its boundary. We define, for $l \in \mathbb{N}^*$, the uniform discretization of Ω as follows:

$$\begin{aligned} \Omega_l = \{\,]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\\ \mid i,j \in \{0,1,\dots,2^l-1\}\} \\ K_{i,j}^l =]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\end{aligned} \tag{5.12}$$

in $2D$, and:

$$\begin{aligned} \Omega_l = \{\,]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\times]2^{-l}b_z k, 2^{-l}b_z(k+1)[\\ \mid i,j,k \in \{0,1,\dots,2^l-1\}\,\} \\ K_{i,j,k}^l =]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[\times]2^{-l}b_z k, 2^{-l}b_z(k+1)[\end{aligned} \tag{5.13}$$

in $3D$, where the K_γ^l are identical meshes of Ω_l, the uniform discretization of Ω at level l.
The flow momentum and mass balance equations read:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}^t \otimes \boldsymbol{u}) + \nabla p - \nu\, \Delta \boldsymbol{u} = \boldsymbol{f} \quad \text{in } \Omega \times]0,T[\\ \nabla \cdot \boldsymbol{u} = 0 \quad \text{in } \overline{\Omega} \times]0,T[\end{cases} \tag{5.14}$$

with a homogeneous Dirichlet boundary condition for $\boldsymbol{u}$ and the initial condition:

$$\boldsymbol{u}(\cdot, 0) = \boldsymbol{u}_{\text{ini}} \quad \text{in} \quad \overline{\Omega}$$

where ν is the cinematic viscosity of the fluid and $\boldsymbol{f}$ is the vector of the source term. We make the following assumptions:

1. T is the finite duration of the flow,
2. $\nu \in]0, +\infty[$,
3. $\boldsymbol{u}_{\text{ini}} \in L^2(\Omega)^d$
4. $\boldsymbol{f} \in L^2(\Omega \times]0, T[)^d$.

We divide the time interval into N equally spaced timesteps, $t^0 = 0, t^1 = h, t^2 = 2h, \ldots, t^n = nh, \cdots t^{N-1} = T$, where h is the temporal stepsize. The subsequent grids Ω_l are embedded, so that we can still associate a tree Λ to the union of grids. At each timestep t^n, we make sure that the information relative to the velocity and pressure variables are encoded into a graded tree. We define mean values $(u^{l,n}_{i,\gamma})_{i=1,\ldots,d}$ and $p^{l,n}_{\gamma}$, that correspond to the approximations of the velocity and pressure variables respectively, at timestep t^n, into the mesh K^l_γ. $(U^n_{i,L(\Lambda)})_{i=1,\ldots,d}$ and $P^n_{L(\Lambda)}$ are the approximations of the velocity and the pressure respectively, at timestep n, on the leaves $L(\Lambda)$, which represent the adaptive grid where we solve numerically (5.14). At the beginning of the computation, we assume that we can set a value L for most refined grid level, so that the meshes on Ω_L can capture all the spatial scales of $\boldsymbol{u}(\boldsymbol{x}, t)$ for $t \in [t^0, T]$. We discretize $\boldsymbol{u}_{\text{ini}}$ on Ω_L. However, it can occur that this initial value does not present the full range of spatial scales of the flow, and this is why we perform few iterations of the numerical solver on the grid Ω_L, before applying the multiresolution transform. In most of our cases, 5 inital solver iterations are enough.

5.2.2 Adaptive Multiresolution Algorithm

Based on the previous algorithms, the complete adaptive multiresolution scheme is implemented following the Algorithm 5.6. We only perform the multiresolution adaptation with regard to the velocity variables (we established in chapter 3 that they were the relevant variables regarding the spatial topology of the flow, while the pressure variable is essentially required to satisfy the divergence-free constraint). We perform thresholding and predictive refinement operations on the d components of the velocity, and we keep any of the nodes that are set to `.true.` by any of these components.

Algorithm 5.6 Fully adaptive multiresolution scheme.

INITIALIZATION:
Define: set of grid levels $[0, L]$, threshold value ε, dimension of the problem d, interpolation stencil M, computational domain $\Omega \subset \mathbb{R}^d$, time domain of integration $t \in [t_0, T]$, the time until which the solution is computed only on the uniform grid t_1.
for $l = 0 \to L$ **do** {Create initial set of grids}
    Create successive dyadic embedded partitions Ω^l of Ω
end for
Compute $(U^L_{i,\gamma})_{i=1,\dots,d}$ at uniform grid level L, following (5.1), from the initial continuous value $\boldsymbol{u}_{\text{ini}}$
Advance the solution from $t = t^0$ to $t = t^1$ on the uniform grid.
$t = t^0$ and $n = n_0$.
Input: $(U^{n_0}_{i,L(\Lambda)})_{i=1,\dots,d}$ of size $d \times \#(\mathrm{L}(\Lambda)) = d \times 2^{dL}$, and $P^{n_0}_{L(\Lambda)}$ of size $\#(\mathrm{L}(\Lambda)) = 2^{dL}$, given by cell values on the finest grid Ω_L, after n_0 iterations of the numerical Navier-Stokes solver
LOOP IN TIME:
while $t \leq T$ **do**
    Encode values from the leaves to the root of the tree, and compute the details, by multiresolution transform $\mathcal{M}$ with Algorithm 5.1.
    Apply the thresholding and predictive refinement algorithm 5.3 to the d components of the velocity; set to `.true.` the binary flag of a cell if at least one of components of the velocity keeps it in its set Λ_ε
    Build a graded tree $\tilde{\Lambda}_\varepsilon$ thanks to Algorithm 5.4
    Delete superfluous cells in $\tilde{\Lambda}^{n+1}_\varepsilon$ with Algorithm 5.5
    Compute the missing values in newly created cells if needed, especially for the pressure variable
    Decode values by inverse multiresolution transform $\mathcal{M}^{-1}$ with Algorithm 5.2, to obtain $(U^n_{i,L(\tilde{\Lambda}^{n+1}_\varepsilon)})_{i=1,\dots,d}$ and $P^n_{L(\tilde{\Lambda}^{n+1}_\varepsilon)}$, on the leaves of the new adaptive tree
    Time integration of the solution on the leaves of the tree: $(U^n_{i,L(\tilde{\Lambda}^{n+1}_\varepsilon)})_{i=1,\dots,d} \to (U^{n+1}_{i,L(\tilde{\Lambda}^{n+1}_\varepsilon)})_{i=1,\dots,d}$ $P^n_{L(\tilde{\Lambda}^{n+1}_\varepsilon)} \to P^{n+1}_{L(\tilde{\Lambda}^{n+1}_\varepsilon)}$, $n \to n+1$, and $t \to t+h$.
end while
$t = T$ and $n = N - 1$.
Output: $\left(U^{N-1}_{i,L(\tilde{\Lambda}^{N-1}_\varepsilon)}\right)_{i=1,\dots,d}$ and $P^{N-1}_{L(\tilde{\Lambda}^{N-1}_\varepsilon)}$

5.3 Basic Code Implementation

We wrote the mrpy code to implement the adaptive multiresolution strategy in the Python programming language. As stated at the beginning of this chapter, we essentially used two ingredients for this implementation: hashtables and space-filling curves. If we consider a set of embedded dyadic grids as the one described by (5.12, 5.13), then each node of the tree associated to this representation can be uniquely identified by its level l and its index tuple γ in the grid Ω_l. In 3D for example, the node $K^l_{i,j,k}$ can be identified by the tuple (l, i, j, k). We decided to take advantage of this fact by associating to the node $K^l_{i,j,k}$ the unique key given by the following `z-curve-index` function:

```
  cpdef int z_curve_index(int dimension, int level,
    int index_x=0, int index_y=0, int index_z=0):

    return int((((2**dimension)**(level) - 1) /
        (2**dimension - 1) +
        index_x +
        index_y * 2**level +
        index_z * 2**level * 2**level))
```

In this way, we can identify each node with a unique integer, with minimal memory cost. Then it is easy to establish the connectivity of the node $K^l_{i,j,k}$. Its neighbors are the nodes $K^l_{i\pm1,j,k}$, $K^l_{i,j\pm1,k}$ and $K^l_{i,j,k\pm1}$. We can associate a `list` of children pointers to this node, which consists of the `z-curve-index` of its children:

```
def create_children_pointers(cell, dimension, level,
    index_x=0, index_y=0, index_z=0):

    if dimension == 1:
        foo = []
        for m in range(2):
            foo.append(z_curve_index(dimension, level + 1,
                2*index_x + m))

        cell.children = foo

    elif dimension == 2:
        foo = []
        for n in range(2):
            for m in range(2):
                foo.append(z_curve_index(dimension, level + 1,
                    2*index_x + m, 2*index_y + n))

        cell.children = foo

    elif dimension == 3:
        foo = []
        for o in range(2):
```

```
            for n in range(2):
                for m in range(2):
                    foo.append(z_curve_index(dimension,
                        level + 1,
                        2*index_x + m, 2*index_y + n,
                        2*  index_z + o))

        cell.children = foo
```

And its parent node can be obtained from:

```
def create_parent_pointer(cell, dimension, level,
    index_x=0, index_y=0, index_z=0):

    if level != 0:
        index_x_parent = int(math.floor(index_x/2))
        index_y_parent = int(math.floor(index_y/2))
        index_z_parent = int(math.floor(index_z/2))

        cell.parent = z_curve_index(dimension,
            level-1, index_x_parent,
            index_y_parent, index_z_parent)
```

The node representation is formally a Python `Object` with various attributes:

`Class cell`

- `self.level (int)`: the level of the node
- `self.isleaf (bool)`: a flag which is .`true`. if the node is a leaf of the tree, and .`false`. otherwise
- `self.index-x (int)`: the node index in the x-direction in the grid Ω_l
- `self.index-y (int)`: the node index in the y-direction in the grid Ω_l
- `self.index-z (int)`: the node index in the z-direction in the grid Ω_l
- `self.dx (float)`: the node size in the x-direction
- `self.dy (float)`: the node size in the y-direction
- `self.dz (float)`: the node size in the z-direction
- `self.coord-x (float)`: the x-component of the node conter coordinate
- `self.coord-y (float)`: the y-component of the node conter coordinate
- `self.coord-z (float)`: the z-component of the node conter coordinate
- `self.parent (int)`: the z-curve-index of the node's parent
- `self.children (list of int)`: a list with the z-curve-index of the nodes's children

- `self.value (float)`: the value stored in the node, $u^l_{i,j,k}$

The second object needed is the tree representation, which is formally a Python `Object` with a `dict` storing all the nodes of the tree indexed by their z-curve-index:

`Class tree`

- `self.tree-nodes (dict of nodes)`: a hashtable sotring the nodes of the tree indexed by their z-curve-index
- `self.tree-leaves (list of int)`: a `list` storing the keys of all the leaves of the tree
- `self.dimension (int)`: the space dimension
- `self.stencil-prediction (int)`: the stencil used for the prediction operations
- `self.min-level (int)`: the minimum level of the nodes in the tree (can be different from 0)
- `self.max-level (int)`: the maximum level of the nodes in the tree

We can easily check whether a given node is in the tree: we compute its z-curve-index, and check if this key belongs to the hashtable `tree.tree-nodes`. With this structure we can perform all the algorithms introduced in section (5.1). For example, the projection from the leaves of the tree to the coarsest grid in Algorithm (5.1) can be done with the two following functions:

```
1  def compute_projection_value(tree, index):
2
3      temp = 0
4      for index_child in tree[index].children:
5          temp = temp + tree[index_child].value
6
7      tree[index].value = temp / len(tree[index].children)
```

```
1  def run_projection(*trees):
2
3      max_tree_nodes_index = max(trees[0].tree_nodes.keys())
4
5      for index in range(int(max_tree_nodes_index), -1, -1):
6          if index in trees[0].tree_nodes:
7              for tree in trees:
8                  if not tree[index].isleaf:
9                      compute_projection_value(tree, index)
```

And the computation of the details is given by:

```
def encode_details(*trees):

    def single_tree_function(tree):
        tree_indexes = tree.tree_nodes.keys()

        for index_parent in tree_indexes:
            if not tree[index_parent].isleaf:

                tree[index_parent].norm_details = 0
                foo = []
                for index_child in tree[index_parent].children:
                    temp = tree[index_child].value -
                        compute_prediction_value(tree,
                        index_parent, index_child)
                    foo.append(temp)
                    # Norm L2 of the details
                    tree[index_parent].norm_details += temp**2

                tree[index_parent].details = foo
                tree[index_parent].norm_details =
                    math.sqrt(tree[index_parent].norm_details /
                    len(tree[index_parent].children))

                tree.max_norm_details =
                    max(tree.max_norm_details,
                    tree[index_parent].norm_details)

    for tree in trees:
        single_tree_function(tree)
```

mrpy is conceived as a library that provides third-party programs with the modules necessary to manipulate the mesh associated with the adaptive multiresolution strategy.

It can be accessed at the following address: `https://github.com/marc-nguessan/mrpy`

Chapter 6

Description of our new high-order Runge-Kutta method coupled to a multiresolution strategy to solve the incompressible Navier-Stokes equations

This chapter aims at merging the theoretical and practical aspects presented in Chapters 2, 4 and 5 into a novel high-order space adaptive scheme for the numerical resolution of the incompressible Navier-Stokes equations. We begin with the description of the linear solvers that we implemented in this work for the sparse linear systems appearing in Chapter 4. We then give some details regarding the treatment of the spurious modes arising from the collocated spatial discretization designed in Chapter 2. In Chapter 4, we identified three Runge-Kutta schemes which are 3^{rd}-order in the velocity variables when applied to incompressible flows. They have very different characteristics, and we provide some insights for their comparison. Finally, we assess the ability of the new scheme to produce adaptive solutions with error control with two test cases, the lid-driven cavity and two-dimensional counter-rotating gaussian vortices.

6.1 Numerical resolution of sparse linear systems

We saw in Chapter 4 that we have to solve sparse linear systems in the implementation of the various high-order RK methods devised specifically for the incompressible Navier-Stokes equations. These systems can be very large, as

their size is directly related to the number of meshes of the adapted grid used for the numerical simulation. For the Radau IIA method (matrices of the type (4.19) if we consider the fixed-point iteration method to solve the nonlinear system (4.16)) and the additive Runge-Kutta methods (matrices of the form (4.31)), we have to deal with a saddle-point problem (4.18), and for the half-explicit RK methods (4.32), the matrix is a variant of the Poisson problem. Solving large linear systems is one of the most difficult tasks in scientific computation, it represents a research topic by itself. Saddle-point problems are especially difficult to solve, due to their indefiniteness and the poor properties of their set of eigenvalues. We will give a brief introduction to these arduous subjects, but only insofar as it serves the purpose of exposing the algorithms and technical solutions implemented in the `mrpy` code to solve the linear systems mentioned above. This is by no means supposed to be a state-of-the-art review of the matter, that would largely exceed the scope of this monograph. The interested reader is referred to the excellent book of Y. Saad on iterative solvers [Saa03] and the review of numerical solutions of saddle-point problems of Benzi, Golub and Liesen [BGL05], and the references therein.

Let us consider the problem of solving the following linear system in the variable x:

$$A\,x = b \tag{6.1}$$

where A is a real square matrix of size $n \times n$ (with n a positive integer), and $b \in \mathbb{R}^n$ a known vector forming the right-hand side of the equation. Basically, there are two types of numerical solvers for this problem, *direct solvers* and *iterative solvers*. Direct solvers rely generally on the LU decomposition of matrix A, or its Cholesky decomposition LDL^t if A is symmetric. The term "direct" refers to the fact that once the decomposition has been computed, x is obtained by an application of forward and backward triangular sweeps to b. On the other hand, iterative solvers generally start with an approximate solution x_0, and pass from one iterate x_k to x_{k+1} by modifying one or a few compenents of x_k at a time. Each iteration involves a few matrix-vector multiplications, and/or vector-vector additions or multiplications. The iterations go on until convergence is reached, where convergence is generally measured relatively to the norm of the residual vector. In our case, we use a relative tolerance argument, and we stop the iterations when we arrive at a value x^* so that:

$$\frac{\|A\,x^* - b\|}{\|b\|} \leq \epsilon$$

Where ϵ is a user-defined parameter, and the norm is the L_2-norm. Direct solvers will typically perform $\mathcal{O}(n^3)$ operations to build the appropriate decomposition of matrix A, whereas iterative solvers perform $\mathcal{O}(n^2)$ operations

at each iteration. Hence, if we have a small number of iterations, iterative solvers can be much more competitive than their direct counterparts. Whereas for a given nonsingular matrix, we are guaranteed that direct solvers can find the solution, the convergence of iterative solvers is rarely guaranteed for all matrices, and is a very problem-dependent issue. But the efficiency and robustness of iterative methods can be largely improved by the use of *preconditioners*. Preconditioning means transforming the original linear system (6.1) into another one with the same solution, hoping that the new system will be much more easier to solve with an iterative solver. Generally speaking, an iterative method is ill-advised unless we know of a very good preconditioning matrix M for our specific problem. Even though some techniques have been developed over the years to find generic good preconditioners, this task remains more of an art than an exact science, and a lot of time should be spent trying to determine a good preconditioning matrix. We quote here Y. Saad [Saa03] regarding the task of finding a good preconditioning matrix M:

> The first step in preconditioning is to find a preconditioning matrix M. The matrix M can be defined in many different ways but it must satisfy a few minimal requirements. From a practical point of view, the most requirement for M is that it is inexpensive to solve linear systems $M\mathrm{x} = \mathrm{b}$. This is because the preconditioned algorithms will all require a linear system solution with the matrix M at each step. Also M should be close to A in some sense and it should clearly be nonsingular.

In any case, the actual performance of a numerical method to solve linear systems depends heavily upon the code implementation of the algorithm, and the specific data structures used to store the matrices and vectors. We chose to use the library **PETSc** (on the recommendation of L. Gouarin) developed at the Mathematics and Computer Science Division of the Argonne National Laboratory [BAA+19b, BAA+19a, BGMS97]. The Portable Extensible Toolkit for Scientific Computation is a sophisticated set of software tools designed for the scalable (parallel) solution of (large-scale) scientific applications modeled by Partial Differential Equations. In particular, it contains dedicated data structures and routines for the storage of matrices (in various formats, among them dense storage and compressed sparse row storage, both sequential and parallel versions), and the resolution of (very large and sparse) linear systems. Iterative solvers based on Krylov subspaces [Saa03] are the heart of PETSc, and their most basic solver is an implementation of the famous GMRES method [Saa03] combined with a preconditioner. Although the library was originally written in C, a Python wrapper, `petsc4py`, has been developed by Lisandro Dalcin, and this is the software that we use in `mrpy`. PETSc also contains a basic implementation of the direct solver technique based on the LU decomposition (that runs only in sequential; for a parallel implementation of direct solvers, it

is possible to interface PETSc with MUMPS [ADKL01, AGLP06].

We developed dedicated preconditioned iterative algorithms to solve our saddle-point problems, relying on a mix of iterative and direct building blocks solvers provided by the PETSc library. We precise here that our goal was simply to find an algorithm to solve our linear systems in a reasonable amount of time, and not to develop state-of-the-art solvers for these problems. We do not pretend that our solvers have optimal convergence rates for example, or that they converge for all the types of matrices that can arise from our resolution of the incompressible Navier-Stokes equations coupled to adaptive multiresolution. But for all of our application test cases, we converged over tens of iterations, which is a quite good performance in this context. Given the fact that the resolution of linear systems are (by far) the most time-consuming task when simulating incompressible flows, we believe that there is still big room for improvement of the techniques exposed below.
We used preconditioned versions of the Uzawa algorithm [UAH58]. We encountered two types of "sub"-matrices to be inverted:

- matrices of the type $D+\Upsilon$ where D is (obviously) a diagonal matrix, and Υ is a small perturbation matrix (in a topological sense) relatively to D. For this case we used the LU decomposition of PETSc, the heuristic reasoning behind this choice being that such matrices are almost diagonal, so direct solvers should work quickly on them (we compared the execution time of the LU decomposition with that of various preconditioned GMRES techniques, and the former was most of the time faster)
- Poisson-like matrices. For this case we used the basic restarted GMRES method of PETSc, with an ILU(0) preconditioner. It should probably be better to use a multigrid preconditioner, but the ILU(0) preconditioner gave satisifying results for all our test cases

The Poisson-like matrices coming from the half-explicit Runge-Kutta methods were also dealt with the restarted GMRES preconditioned by an ILU(0) method. We refer to the PETSc documentation for details regarding the implementation of these various techniques.
In what follows, we will first give a brief presentation of the original Uzawa method, and recall some of its convergence properties. Then we will present our precondtioned versions.

6.1.1 The Uzawa algorithm for saddle-point problems

Let us consider the following saddle-point problem:

$$\begin{pmatrix} A & B \\ B^t & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \tag{6.2}$$

where A is a symmetric definite-positive matrix, and B is a full rank, which represents the most basic case of saddle-point problems [BGL05]. In this case the system (6.2) does have a solution. Algorithm 6.1 is the basic implementation of the Uzawa iterative solver.

Algorithm 6.1 The Uzawa algorithm

1: **Input:** Initial approximate values x_0 and y_0.
2: **for** $k = 0 \rightarrow$ convergence: **do**
3: $\quad x_{k+1} = A^{-1}(b - By_k)$
4: $\quad y_{k+1} = y_k + \omega(B^t x_{k+1} - c)$
5: **end for**
6: **Output:** The (approximate) solutions x^* and y^*.

ω is a relaxation parameter that is user-defined. It turns out that a simple condition on the value ensures the convergence of Algorithm 6.1 to the true solutions of (6.2):

Theorem 6.1.1. *Let A be a Symmetric Positive Definite matrix and B a matrix of full rank. Then $S = B^t A^{-1} B$ is also Symmetric Positive Definite, and Uzawa's Algorithm 6.1 converges, if and only if:*

$$0 < \omega < \frac{2}{\lambda_{\max}(S)} \tag{6.3}$$

In addition, the optimal convergence parameter ω is given by:

$$\omega_{opt} = \frac{2}{\lambda_{\min}(S) + \lambda_{\max}(S)}$$

$\lambda_{\min}(S)$ and $\lambda_{\max}(S)$ denote respectively the minimum and maximum real eigenvalues of matrix S. A proof of this result can be found in [Saa03]. Matrix S is called *the Schur complement* of the saddle-point matrix (6.2); it plays an important role in the study of the mathematical properties of saddle-point matrices [BGL05]. In the Uzawa's algorithm's for instance, if we substitute x_{k+1} from line 3 into line 4, it can be shown that we are in fact applying the Richardson iterative solver to the following system [Saa03]:

$$B^t A^{-1} B y = B^t A^{-1} b - c \tag{6.4}$$

In most cases, we do not have access to the matrix A^{-1}, but Algorithm 6.1 needs not computing it explicitly to solve S. It also shows us than we can speed up this algorithm by choosing a preconditioner for S, as long as we can compute easily an approximate of A^{-1}, and this exactly what we are going to do for our saddle-point problems.

6.1.2 A new preconditioned Uzawa algorithm for the saddle-point problem arising from ARK methods for the incompressible Navier-Stokes equations

If we apply the implicit-explicit Runge-Kutta methods designed in section (4.3.1) to solve the semi-discretized incompressible Navier-Stokes equations obtained from the spatial discretization of the Chapter 2, we have to solve for each implicit stage i a linear system whose matrix has the form $E = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}$, where:

- B is block-diagonal matrix consisting of d (the space dimension) diagonal blocks B_j, with $B_j = \frac{1}{\delta t}\Gamma - \gamma\nu L_j$
- $M = \begin{pmatrix} M_1 \\ \vdots \\ M_d \end{pmatrix}$, with $M_j = -\gamma D_j^t$
- $N = \begin{pmatrix} N_1 & \cdots & N_d \end{pmatrix}$, with $N_j = D_j$

The matrices Γ, D_j and L_j come from the finite volume scheme developed in Chapter 2, and we refer the reader to the beginning of Chapter 3 for their definition. We denote by $(g_j^i)_{j=1,\ldots,d}$ the intermediate velocity components that we have to compute at implicit stage i, and by q^i the intermediate pressure. We also denote by $(b_j^i)_{j=1,\ldots,d}$ the right-hand side of this system for the momentum equations (that depends on the source-terms S_i in (3.1) and the convective terms of the stage i of the ERK method), and by c^i the right-hand side for the divergence equation.
We solve for matrix E with a preconditioned version of Algorithm 6.1. We choose as initial approximate solutions the values from the preceding stage $(g_j^{i-1})_{j=1,\ldots,d}$ and q^{i-1}. The first step of each iteration consists in computing B^{-1}. Since this is a block diagonal matrix, this task reduces to computing the independent sub-matrices B_j^{-1} (with the LU decomposition). It is actually faster to solve for the d independent linear systems B_j than to solve for the bigger matrix B. We remark that since Γ is a diagonal matrix (the mass matrix), and that the timestep δt is user-defined, we can always choose it so as to make sure that the matrices B_j are symmetric positive definite matrices (thanks to the Gershgorin theorem). The second step of the iteration k of our Uzawa algorithm consists of updating q_{k+1}^i. To this end, we first compute δq by solving the following system:

$$N\tilde{B}^{-1}M\delta q = \sum_{j=1}^{d} N_j g_{j,k+1}^i - c^i$$

where $g^i_{j,k+1}$ are the updated components of the velocity computed from the first step of the iteration. $N\tilde{B}^{-1}M$ is our preconditioning matrix. We approximate B^{-1} by the bock-diagonal matrix $\tilde{B}^{-1}$ consisting of d diagonal blocks $\tilde{B}_j$, with $\tilde{B}_j = \delta t \Gamma^{-1}$. this matrix is easy to assemble, because Γ^{-1} is the inverse of the mass matrix, which is a diagonal matrix. The heuristic reasoning behind this choice is the following: for δt small enough, the B_j matrices are of the type $D + \Upsilon$, with D a diagonal matrix, and Υ a small perturbation in comparison. Hence the B_j^{-1} matrices get closer and closer to the matrices $\tilde{B}_j$ as δt tends to zero. $N\tilde{B}^{-1}M$ is a Poisson-like matrix, and we invert it with the GMRES method of PETSc. Then we update q^i_{k+1}:

$$q^i_{k+1} = q^i_k + \omega \delta q$$

where ω is the relaxation parameter. All this procedure results in Algorithm 6.2.

Algorithm 6.2 The Uzawa algorithm for the stage i of the ARK method

1: **Input:** Initial approximate values $(g^i_{j,0} = g^{i-1}_j)_{j=1,\dots,d}$ and $q^i_0 = q^{i-1}$.
2: **for** $k = 0 \rightarrow$ convergence: **do**
3: **for** $j = 1 \rightarrow d$: **do**
4: $g^i_{j,k+1} = B_j^{-1}(b^i - M_i q^i_k)$
5: **end for**
6: $\delta q = (N\tilde{B}^{-1}M)^{-1}(\sum_{j=1}^{d} N_j g^i_{j,k+1} - c^i)$
7: $q^i_{k+1} = q^i_k + \omega \delta q$
8: **end for**
9: **Output:** The (approximate) solutions $(g^i_j)_{j=1,\dots,d}$ and q^i.

6.1.3 A new preconditioned Uzawa algorithm for the saddle-point problem arising from the Radau IIA method for the incompressible Navier-Stokes equations

We use the fixed-point Picard iteration exposed in section 4.2.2 to solve the nonlinear system (4.16) that comes from the application of the Radau IIA method (4.5) to the semi-discretized Navier-Stokes equations. To perform the iteration i of this nonlinear solver, we have to solve a linear system whose matrix has the form $E = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}$, where:

- B is block-diagonal matrix consisting of d (the space dimension) diagonal bloks B_i, with $B_i = \begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{11}\nu L_i & -a_{12}\nu L_i \\ -a_{21}\nu L_i & \frac{1}{\delta t}\Gamma - a_{22}\nu L_i \end{pmatrix}$

- $M = \begin{pmatrix} M_1 \\ \vdots \\ M_d \end{pmatrix}$, with $M_i = \begin{pmatrix} -a_{11}D_i^t & -a_{12}D_i^t \\ -a_{21}D_i^t & -a_{22}D_i^t \end{pmatrix}$
- $N = \begin{pmatrix} N_1 & \cdots & N_d \end{pmatrix}$, with $N_i = \begin{pmatrix} D_i & 0 \\ 0 & D_i \end{pmatrix}$

The matrices Γ, D_j and L_j come from the finite volume scheme developed in Chapter 2, and we refer the reader to the beginning of Chapter 3 for their definition. We denote by $(\boldsymbol{g}_j^i = (g_j^{1,i}, g_j^{2,i}))_{j=1,\ldots,d}$ the intermediate velocity components that we have to compute at iteration i, that comprises the variables for the two stages of the Radau method, and by $\boldsymbol{q}^i = (q^{1,i}, q^{2,i})$ the intermediate pressure, that comprises the variables for the two stages of the Radau method. We also denote by $(\boldsymbol{b}_j^i = (b_j^{1,i}, b_j^{2,i}))_{j=1,\ldots,d}$ the right-hand side of this system for the momentum equations (that depends on the source-terms S_i in (3.1) and the convective terms of the iteration i of the Picard method), and by $\boldsymbol{c}^i = (c^{1,i}, c^{2,i})$ the right-hand side for the divergence equation.
We solve for matrix E with a preconditioned version of Algorithm 6.1. We choose as initial approximate solutions the values from the preceding Picard iteration $(\boldsymbol{g}_j^{i-1})_{j=1,\ldots,d}$ and $\boldsymbol{q}^{i-1}$. The first step of each iteration consists in computing B^{-1}. Since this is a block diagonal matrix, this task reduces to computing the independent sub-matrices B_j^{-1} (with the LU decomposition). It is actually faster to solve for the d independent linear systems B_j than to solve for the bigger matrix B. The second step of the iteration k of our Uzawa algorithm consists of updating $\boldsymbol{q}_{k+1}^i$. To this end, we first compute $(\delta q^1, \delta q^2)$ by solving the following system:

$$\delta t a_{11}\Big(\sum_{j=1}^{d} D_j \Gamma^{-1}(-D_j^t)\Big)\delta q^1 = \sum_{j=1}^{d} D_j g_{j,k+1}^{1,i} - c^{1,i}$$

$$\delta t a_{22}\Big(\sum_{j=1}^{d} D_j \Gamma^{-1}(-D_j^t)\Big)\delta q^2 = \sum_{j=1}^{d} D_j g_{j,k+1}^{2,i} - c^{2,i}$$

where $\boldsymbol{g}_{j,k+1}^i$ are the updated components of the velocity computed from the first step of the (Uzawa) iteration. Then we update $\boldsymbol{q}_{k+1}^i$:

$$q_{k+1}^{1,i} = q_k^{1,i} + \omega \delta q^1$$

$$q_{k+1}^{2,i} = q_k^{2,i} + \omega \delta q^2$$

where ω is the relaxation parameter. All this procedure results in Algorithm 6.3.

1: **Input:** Initial approximate values $(\boldsymbol{g}_{j,0}^i = \boldsymbol{g}_j^{i-1})_{j=1,\ldots,d}$ and $\boldsymbol{q}_0^i = \boldsymbol{q}^{i-1}$.
2: **for** $k = 0 \rightarrow$ convergence: **do**

Algorithm 6.3 The Uzawa algorithm for the iteration i of the fixed-point Picard iteration for the Radau IIA method

for $j = 1 \to d$: **do**
  $\boldsymbol{g}^i_{j,k+1} = B_j^{-1}(b^i - M_i \boldsymbol{q}^i_k)$
end for
$\delta q^1 = [\delta t a_{11}(\sum_{j=1}^d D_j \Gamma^{-1}(-D_j^t))]^{-1}(\sum_{j=1}^d D_j g^{1,i}_{j,k+1} - c^{1,i})$
$\delta q^2 = [\delta t a_{22}(\sum_{j=1}^d D_j \Gamma^{-1}(-D_j^t))]^{-1}(\sum_{j=1}^d D_j g^{2,i}_{j,k+1} - c^{2,i})$
$q^{1,i}_{k+1} = q^{1,i}_k + \omega \delta q^1$
$q^{2,i}_{k+1} = q^{2,i}_k + \omega \delta q^2$
end for
Output: The (approximate) solutions $(\boldsymbol{g}^i_j)_{j=1,\dots,d}$ and $\boldsymbol{q}^i$.

6.2 Treatment of the spurious pressure and velocity modes

As we explained at the beginning of Chapter 2, the spurious pressure and velocity modes are artificial kernel modes of the discrete divergence and gradient operators. Given the fact that our spatial discretization scheme is not staggered, we are not ensured that such modes cannot appear in our numerical simulations. We thus have to design a special treatment for these parasite values, and we found a solution for the Radau IIA method and the ARK methods, that we present below.

The spurious modes are due to the linear part of equation (2.1) [GS00], so that we only have to consider the Stokes equation for their treatment. If we do not take into account the convective terms in equations (3.1), the implementation of the Radau IIA method or the ARK methods amounts to the resolution of the linear systems that we explicited in section 6.1. These matrices can have artificial kernel modes that can pollute the approximate velocity and pressure fields computed at each time step. Our treatment for both methods is the same: we will show that depending on the timestep δt, the particular choice of the discrete operators precludes the apparition of spurious modes for the velocity. We simply do not care about the spurious pressure modes, as long as they cannot affect the accuracy of the approximate velocities. Indeed, it is the flow velocity that is detrimental for the species transport in reacting flows.

6.2.1 Treatment of the spurious modes for the ARK methods

If we apply the ESDIRK methods designed in section (4.3.2) to solve the semi-discretized incompressible Stokes equations obtained from the spatial discretization of the Chapter 2, we have to solve for each implicit stage i a linear system whose matrix has the form $E = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}$, where:

- B is block-diagonal matrix consisting of d (the space dimension) diagonal blocks B_j, with $B_j = \frac{1}{\delta t}\Gamma - \gamma\nu L_j$
- $M = \begin{pmatrix} M_1 \\ \vdots \\ M_d \end{pmatrix}$, with $M_j = -\gamma D_j^t$
- $N = \begin{pmatrix} N_1 & \cdots & N_d \end{pmatrix}$, with $N_j = D_j$

The matrices Γ, D_j and L_j come from the finite volume scheme developed in Chapter 2, and we refer the reader to the beginning of Chapter 3 for their definition. We denote by $(g_j^i)_{j=1,\dots,d}$ the intermediate velocity components that we have to compute at implicit stage i, and by q^i the intermediate pressure. The spurious modes that will affect the solution of our problem are the vectors of the form $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ which are in the null-space of E: $E\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = 0$. If $\boldsymbol{v} \neq 0$, it will pollute the quantities $(g_j^i)_{j=1,\dots,d}$ causing the apparition of spurious modes in the computed solution; and of course the same is true with q regarding q^i. The matrix B depends on δt the timestep. We will characterise the null-space of matrix E for some values of δt, and we start with the following lemma:

Lemma 6.2.1. *If a matrix B is as defined below, then there exists $h_0 > 0$ so that for each $0 < \delta t \leq h_0$ and for any real vector $\boldsymbol{z}$ the two following propositions are equivalent:*

1. $\boldsymbol{z}^t B \boldsymbol{z} = 0$
2. $\boldsymbol{z} = 0$

Proof. (2) $\implies$ (1) is obvious for any $\delta t \in \mathbb{R}_+^*$. We then turn to (1) $\implies$ (2). $\boldsymbol{z}^t B \boldsymbol{z} = \sum_{i=1}^d \boldsymbol{z^i}^t B_i \boldsymbol{z^i}$, each L_i is a symmetric real matrix and Γ is the mass matrix, so that it is diagonal and invertible, and all its diagonal terms are non-negative. The set of invertible matrices being open, for δt small enough, each matrix B_i is symmetric, definite and positive. This implies that for each $i = 1, \dots, d$, $\boldsymbol{z^i}^t B_i \boldsymbol{z^i} \geq 0$, with $\boldsymbol{z^i}^t B_i \boldsymbol{z^i} = 0 \implies \boldsymbol{z^i} = 0$, which concludes the

proof.

□

We then have the following result:

Theorem 6.2.1. *There exists $h_0 > 0$ so that for each $0 < \delta t \leq h_0$, if $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ is a null vector of the matrix E, then $\boldsymbol{v} = 0$.*

Proof. We have that $B\boldsymbol{v} + Mq = 0$. If we multiply this vector by $\boldsymbol{v}^t$, we have $\boldsymbol{v}^t B\boldsymbol{v} + \boldsymbol{v}^t Mq = 0$. Or $\boldsymbol{v}^t Mq = q^t(M^t\boldsymbol{v})$, and if we write $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^d \end{pmatrix}$, we have that $M^t\boldsymbol{v} = -\gamma(\sum_{i=1}^d D_i\boldsymbol{v}^i)$ because $N\boldsymbol{v} = 0$. Hence $\boldsymbol{v}^t B\boldsymbol{v} = 0$.
We see that if we choose a real h_0 that satisfies the conditions of the lemma, then necessarily, $\boldsymbol{v} = 0$, which concludes the proof of the theorem.

□

We see that we can always choose δt small enough to preclude the apparition of spurious modes for the velocity field, the one that is detrimental for the species transport in a combustion simulation.

6.2.2 Treatment of the spurious modes for the Radau method

If we use the Radau IIA method (4.5) to the semi-discretized Stokes equations, at each timestep we have to solve a linear system whose matrix has the form $E = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}$, where:

- B is block-diagonal matrix consisting of d (the space dimension) diagonal bloks B_i, with $B_i = \begin{pmatrix} \frac{1}{\delta t}\Gamma - a_{11}\nu L_i & -a_{12}\nu L_i \\ -a_{21}\nu L_i & \frac{1}{\delta t}\Gamma - a_{22}\nu L_i \end{pmatrix}$
- $M = \begin{pmatrix} M_1 \\ \vdots \\ M_d \end{pmatrix}$, with $M_i = \begin{pmatrix} -a_{11}D_i^t & -a_{12}D_i^t \\ -a_{21}D_i^t & -a_{22}D_i^t \end{pmatrix}$
- $N = \begin{pmatrix} N_1 & \cdots & N_d \end{pmatrix}$, with $N_i = \begin{pmatrix} D_i & 0 \\ 0 & D_i \end{pmatrix}$

The matrices Γ, D_j and L_j come from the finite volume scheme developed in Chapter 2, and we refer the reader to the beginning of Chapter 3 for their definition. We denote by $(\boldsymbol{g}_j = (g_j^1, g_j^2))_{j=1,\ldots,d}$ the intermediate velocity components that we have to compute, that comprises the variables for the two stages of the

Radau method, and by $\boldsymbol{k} = (k^1, k^2)$ the intermediate pressure, that comprises the variables for the two stages of the Radau method.
The spurious modes that will affect the solution of our problem are the vectors of the form $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ which are in the null-space of E: $E \begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = 0$. If $\boldsymbol{v} \neq 0$, it will pollute the quantities g_j^1, g_j^2, causing the apparition of spurious modes in the computed solution; and of course the same is true with q regarding k^1, k^2. But as we wrote earlier, we only need the velocity for the species transport equations, not the pressure, so we are just going to ensure that there is no spurious mode for the velocity. There might be spurious mode in the pressure, but if we make sure that they do not affect the velocity, we are ensured of the precision of our velocity computation.
We remark here that by construction, $-\nu L_i$ is a symmetric and diagonally dominant matrix, so that it is diagonalizable (in $\mathbb{R}$) and all its eigenvalues are positive. The matrix B depends on δt the timestep, and we use the following result:

Theorem 6.2.2. *There exists $h_0 > 0$ so that for each $0 < \delta t \leq h_0$, if $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ is a null vector of the matrix E, then $\boldsymbol{v} = 0$.*

We will use the following lemma:

Lemma 6.2.2. *If a matrix B is as defined above, then there exists $h_0 > 0$ so that for each $0 < \delta t \leq h_0$ and for any real vector $\boldsymbol{z}$ the two following propositions are equivalent:*

1. $\boldsymbol{z}^t B \boldsymbol{z} = 0$
2. $\boldsymbol{z} = 0$

Proof. (2) $\implies$ (1) is obvious for any $h \in \mathbb{R}_+^*$. We then turn to (1) $\implies$ (2). For $\delta t \to 0$, B converges to a diagonal matrix where each block is the mass matrix multiplied by the inverse of the time step. Hence, in the limit, all its eigenvalues are positive. By the continuous dependency of the eigenvalues on the matrix coefficients, there is some $h_0 > 0$ such that all eigenvalues of B, for $\delta t \leq h_0$, are positive too. In that case $\boldsymbol{z}^t B \boldsymbol{z} = 0$ implies that $\boldsymbol{z} = 0$, which concludes the proof of the lemma.

□

Proof. We have that $B\boldsymbol{v} + Mq = 0$. If we multiply this vector by $\boldsymbol{v}^t$, we have $\boldsymbol{v}^t B \boldsymbol{v} + \boldsymbol{v}^t M q = 0$. Or $\boldsymbol{v}^t M q = q^t (M^t \boldsymbol{v})$, and if we write $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^d \end{pmatrix}$, we have

that $M^t\boldsymbol{v} = \begin{pmatrix} -a_{11}(\sum_{i=1}^{d} D_i\boldsymbol{v}^i) - a_{21}(\sum_{i=1}^{d} D_i\boldsymbol{v}^i) \\ -a_{12}(\sum_{i=1}^{d} D_i\boldsymbol{v}^i) - a_{22}(\sum_{i=1}^{d} D_i\boldsymbol{v}^i) \end{pmatrix} = 0$, because $N\boldsymbol{v} = 0$.
Hence $\boldsymbol{v}^t B\boldsymbol{v} = 0$.
We see that if we choose a real h_0 that satisfies the conditions of the lemma, then necessarily, $\boldsymbol{v} = 0$, which concludes the proof of the theorem.

□

We see that we can always choose δt small enough to preclude the apparition of spurious modes for the velocity field, the one that is detrimental for the transport of scalars.

6.3 Comparison of the 3^{rd}-order Runge-Kutta schemes

In chapter 4 we identified three 3^{rd}-order Runge-Kutta methods to solve the incompressible Navier-Stokes equations: the RadauIIA method in section (4.1.1), the half-explicit implementation of the explicit 3-stage Heun method in section (4.4.2), and the new additive Runge-Kutta method that we designed, $ARK-ESDIRK3(2I)4SA$ in section (4.3.4). We showcased the performance of these three methods on the classical two-dimensional lid-driven cavity. But these three schemes have very different properties, and it could be difficult from a theoretical point of view to decide which one is the best. In our opinion, the best suited scheme will be application-dependent, resulting from an arbitrage between precision and time-to-solution. In this section, we want to give some insights for the comparison of these schemes. For the sake of simplicity, we will denote the Radau IIA method *radau*, the half-explicit 3-stage Heun method *herk*, and the additive Runge-Kutta method *ark*.

We consider the two dimensional lid-driven cavity. We consider the problem set described at the beginning of section (4.2.3). We perform computations on a uniform grid at $l_{max} = 6$, from $t = 0$ until $t = 5$, with different timesteps values, and at two different Reynolds numbers, $Re = 1000$ and $Re = 100$.
For each one of the three RK schemes, we compute the error between the solution obtained with various large timesteps, and the solution obtained with the half-explicit Heun method with a very small timestep of $\delta t = 10^{-4}$. We also save the computational time-to-solution for each timestep.
Since the results of the schemes are compared to the same quasi-exact solution, a first element of comparison is to consider which scheme, given a specific timestep, reaches the smallest error. Figures (6.1) and (6.2) give us a first insight. They represent the error in the first component of the velocity, for $Re = 1000$ and $Re = 100$, respectively. For $Re = 1000$, we tested the three schemes with the same set of timestep values, $\delta t = 6.25 \times 10^{-2}$, 5×10^{-2}, $4.16 \times$

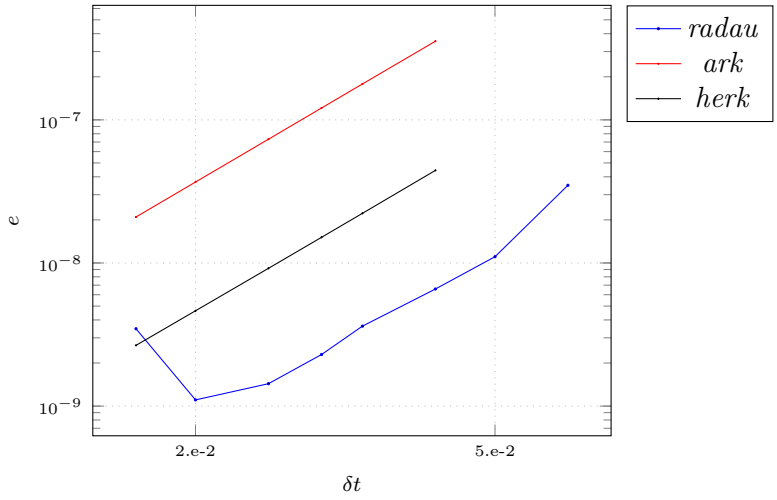


Figure 6.1: Lid-driven cavity, $Re = 1000$. Temporal order of accuracy comparisons of *radau*, *herk* and *ark*; horizontal component of the velocity

10^{-2}, 3.33×10^{-2}, 2.94×10^{-2}, 2.5×10^{-2}, 2×10^{-2}, 1.67×10^{-2}. For $Re = 1000$, it is clear that the best scheme, at a given timestep, is *radau*. It is also the most stable, the methods *ark* and *herk* were instable for the two largest timesteps, $\delta t = 6.25 \times 10^{-2}$ and $\delta t = 5 \times 10^{-2}$. *herk* gives a more precise solution than *ark* at a given timestep. But *ark* is more stable than *herk* with regard to the diffusive terms in the momentum equations. Indeed, when we reduce the Reynolds number, the diffusive terms have more influence than the convective terms, and since the former are *stiffer* than the latter, schemes better suited to tackle sitffness in the incompressible flow equations will tend to perform better. At $Re = 100$ in this example, *radau* remains stable over the same timestep range of the $Re = 1000$ test case; *ark* is instable for the three largest timesteps $\delta t = 6.25 \times 10^{-2}$, $\delta t = 5 \times 10^{-2}$ and 4.16×10^{-2}, but converges for the remaining values; but *herk* was completely instable for the whole range of initial timesteps. The first value for a stable computation of *herk* at $Re = 100$ was $\delta t = 7.14 \times 10^{-3}$, which is almost an order of magnitude smaller than the timesteps at which the other two methods are stable.
Another element of comparison is the time-to-solution to reach a given error threshold with respect to the quasi-exact solution. In figures (6.3) and (6.4), we draw a scatter plot of the time-to-solution versus the error for *ark* and *radau*, at $Re = 1000$ and $Re = 100$, respectively. We see that even though *radau* is more stable and more precise than *ark*, the time-to-solution of *ark* is almost two order of magnitude smaller than the time-to-solution of *radau*.

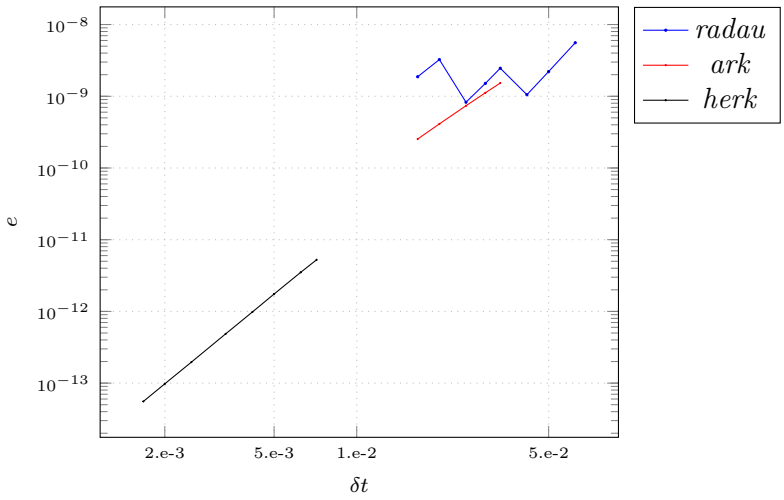


Figure 6.2: Lid-driven cavity, $Re = 100$. Temporal order of accuracy comparisons of *radau*, *herk* and *ark*; horizontal component of the velocity

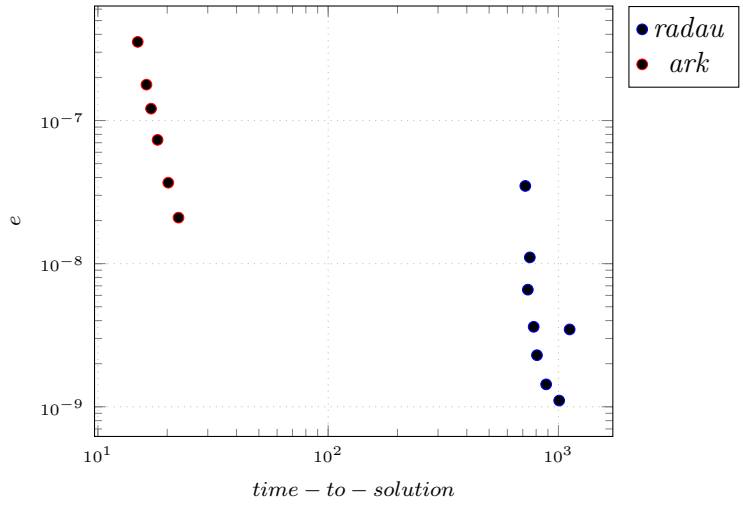


Figure 6.3: Lid-driven cavity, $Re = 1000$. computational time vs error scatter plot of *radau* and *ark*; horizontal component of the velocity

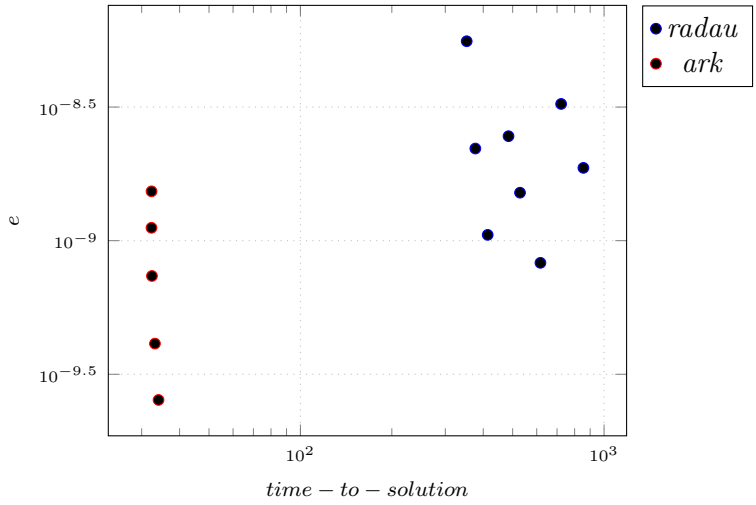


Figure 6.4: Lid-driven cavity, $Re = 100$. computational time vs error scatter plot of *radau* and *ark*; horizontal component of the velocity

6.4 Numerical assessment of the new adaptive strategy

The goal of this section is to check the performance of the new fully adaptive multiresolution scheme exposed in section 5.2, Algorithm (5.6). We will show (i) that the scheme allows effectively to obtain a dynamic grid adaptation as the flow evolves in time, and (ii) that we are able to control the adaptive error with regard to a computation on a uniform grid. All the computations in this section rely on the additive Runge-Kutta scheme $ARK - ESDIRK3(2I)4SA$ presented in section 4.3.4. We present two test cases, the lid-driven cavity, and the simulation of two-dimensional counter-rotating vortices.

6.4.1 Two-dimensional lid-driven cavity

The first test case is again the lid-driven cavity case, already presented in section 4.2.3, that we recall here for the sake of completeness. We discretize equations (2.1) on a uniform or adaptive grid with the spatial scheme (2.5), in an open bounded domain Ω in $2D$, where $\Omega =]-0.5, 0.5[\times]-0.5, 0.5[$. The fluid is initially at rest within Ω ($\boldsymbol{u}_{ini}(\boldsymbol{x}, 0) = \mathbf{0}$ for $\boldsymbol{x} \in \Omega$), and we impose Dirichlet boundary conditions on the four edges of $\partial\Omega$: the velocity is set to zero on all but one of these boundaries, in our case the top one at $y = 0.5$, where the tangential velocity is set to $u_1(x, 0.5, t) = -1$. The Reynolds number is $Re = 1000$.
We fix the thresholding parameter at $\varepsilon = 5 \times 10^{-3}$, the level max $l_{max} = 8$ for

ε	7.10^{-2}	3.10^{-2}	1.10^{-2}	7.10^{-3}	3.10^{-3}	1.10^{-3}	7.10^{-4}	3.10^{-4}
τ	94.27%	92.11%	87.39%	84.47%	77.68%	63.30%	50.43%	35.40%
Speedup	$\times$11.95	$\times$8.96	$\times$6.27	$\times$5.49	$\times$3.72	$\times$1.99	$\times$1.52	$\times$0.97

Table 6.1: Lid-driven cavity. Comparison between the uniform grid and multiresolution adaptive grid computing times for different thresholding parameters. Uniform grid with $l^{max} = 8$. For a given value ε of the thresholding parameter, the uniform grid computation is *speedup* times longer than the computation with adaptive multiresolution. τ is the compression rate.

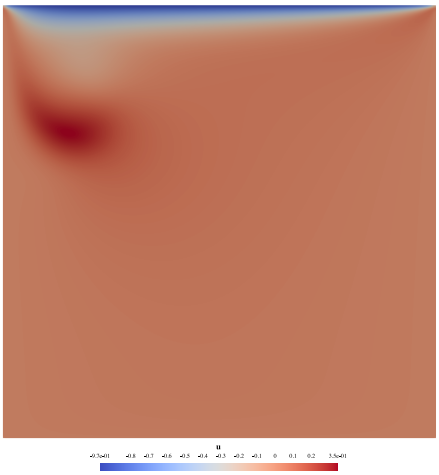
the finest grid, the timestep $h = 6.7\times10^{-3}$ and compute from $t = 0$ until $t = 16$.

Figures (6.5), (6.6), (6.7) show the comparison between the uniform computation and the adapted computation, for the horizontal and vertical components of the velocity, and the velocity norm, respectively. There is a very good match between both computations, and the adaptive scheme produces good qualitative results. Figures (6.8) show the evolution of the adaptive grid, and we can verify that the level of refinement is directly related to the variation in the velocity norm, as expected.
Next, we check the ability of the code to control the multiresolution error. We fix the maximum grid level $l_{max} = 8$ for the finest grid, the timestep $h = 5\times10^{-3}$ and compute from $t = 0$ until $t = 5$ for different values of the thresholding parameter. At the end of the computation, we compute the L^2 norm of the error between the adapted grid solution, and the solution computed with the same features but on a uniform grid. Figures (6.9) and (6.10) show the evolution of the error versus the thresholding parameter for the horizontal and vertical components of the velocity, respectively. We see that we effectively control the adaptive multiresolution error, that scales as a multiple of the thresholding parameter.

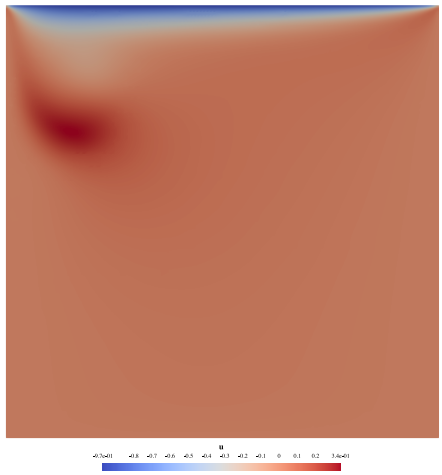
Finally, we assess the computational gain of the adaptive scheme.
The reduction in the number of mesh cells can be measured with the compression rate τ (1.83) introduced in section (1.3) and that we recall here:

$$\tau = 1 - \frac{\#(L(\Lambda))}{2^{2\times l_{max}}}$$

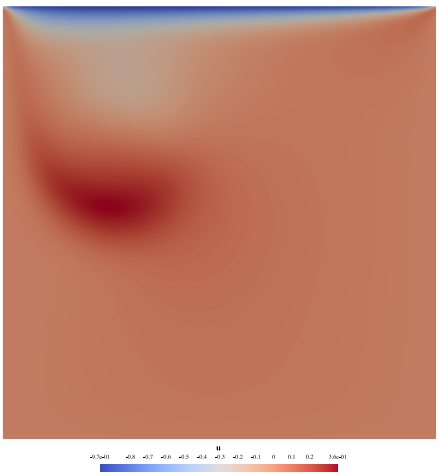
Since $l_{max} = 8$, we have 65536 mesh cells on the uniform finest grid.
Table (6.1) shows some impressive results in the acceleration of the multiresolution computations. For $\varepsilon = 3 \times 10^{-2}$, the error with regard to the uniform computation is around 10^{-5}, and the adaptive scheme is almost 9 times faster than the uniform grid computation. We precise that the code used here is a research/development software, which primary purpose was not to obtain high-



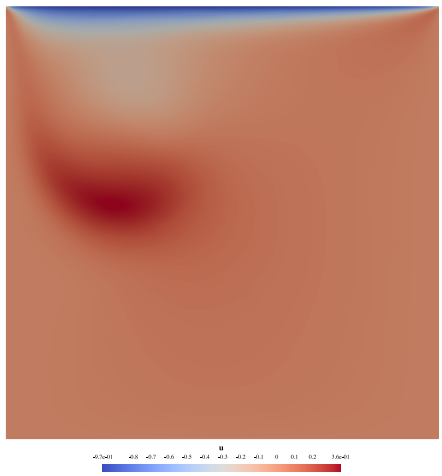
(a) $t = 2$



(b) $t = 2$

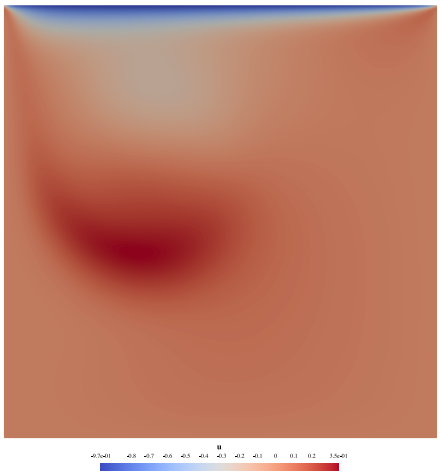


(c) $t = 4$

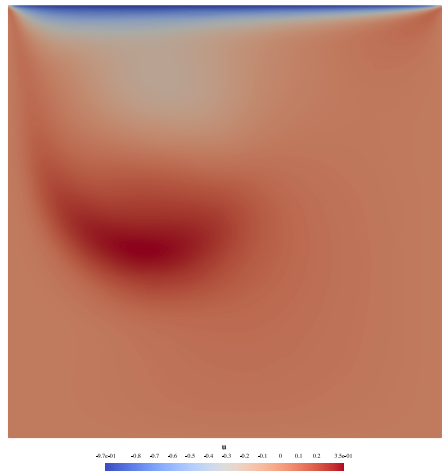


(d) $t = 4$

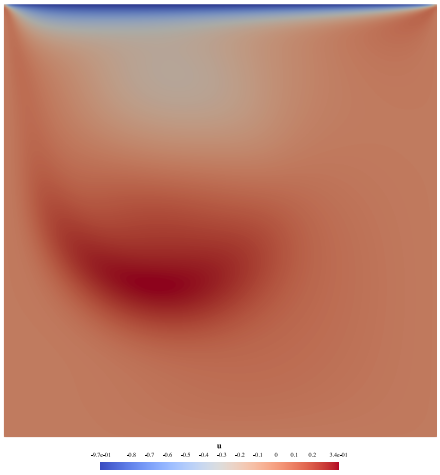
Figure 6.5: Lid-driven cavity. Evolution of the horizontal velocity component for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



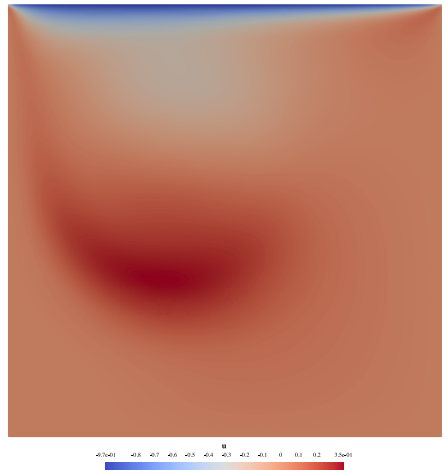
(e) $t = 6$



(f) $t = 6$

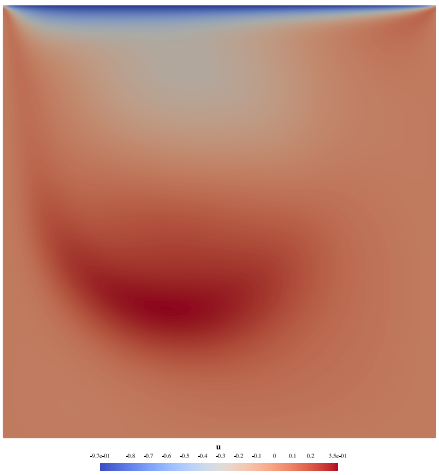


(g) $t = 8$

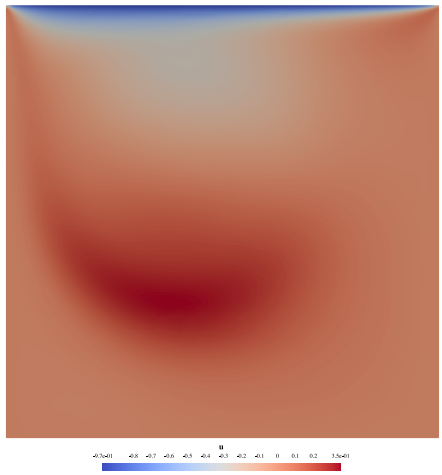


(h) $t = 8$

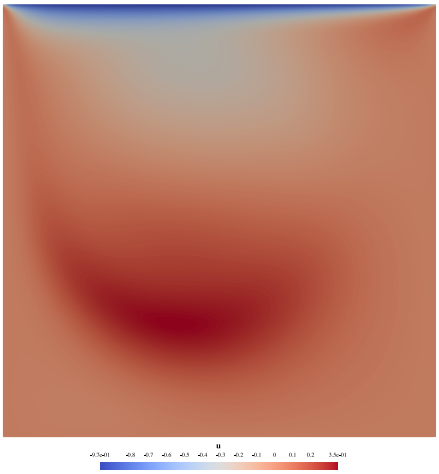
Figure 6.5: Lid-driven cavity. Evolution of the horizontal velocity component for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



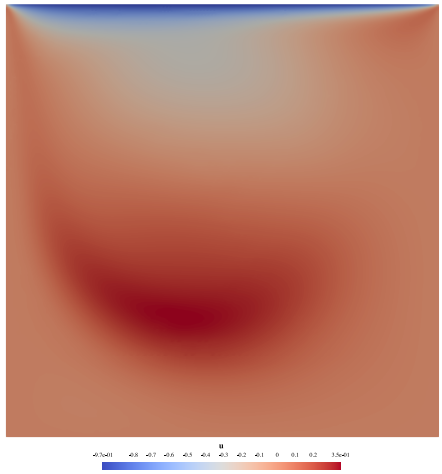
(i) $t = 10$



(j) $t = 10$

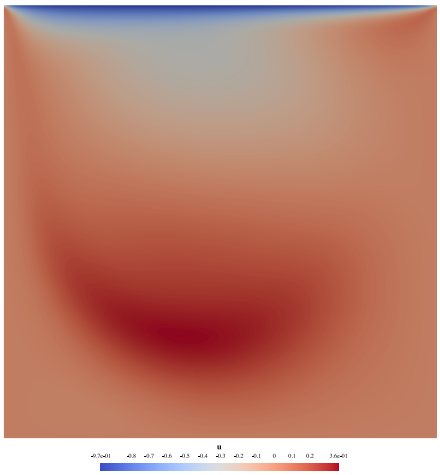


(k) $t = 12$

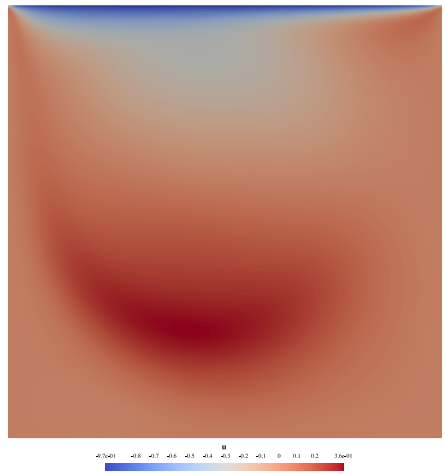


(l) $t = 12$

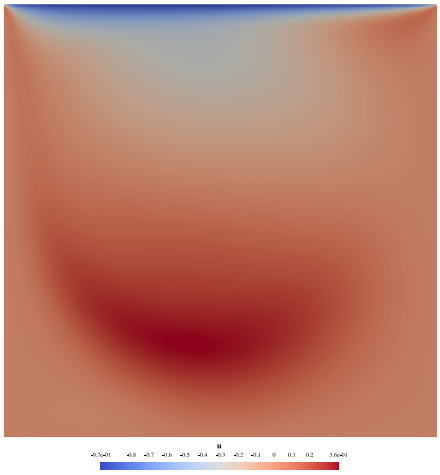
Figure 6.5: lid-driven cavity. evolution of the horizontal velocity component for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



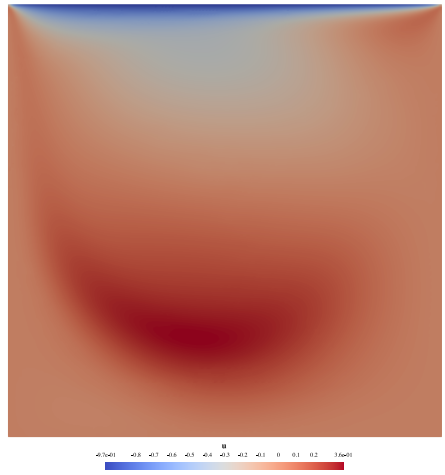
(m) $t = 14$



(n) $t = 14$

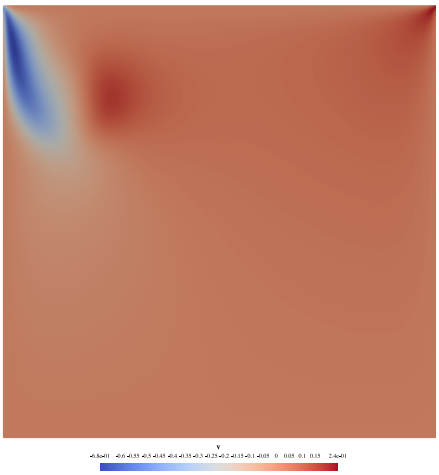


(o) $t = 16$

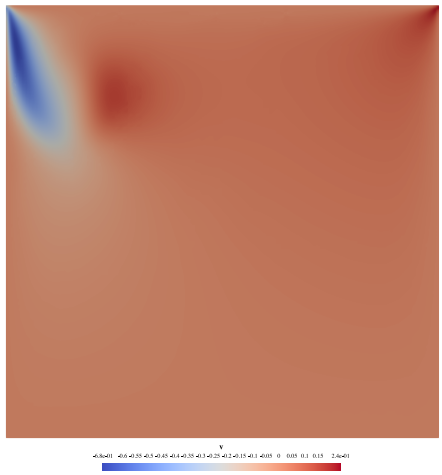


(p) $t = 16$

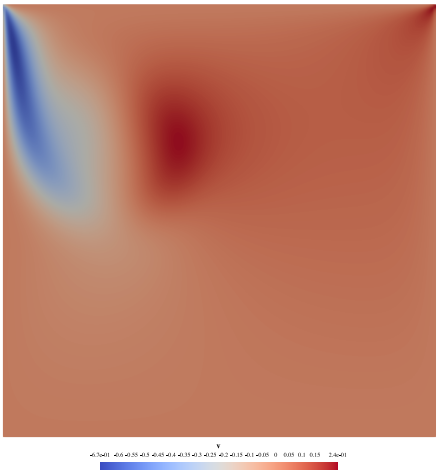
Figure 6.5: lid-driven cavity. evolution of the horizontal velocity component for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



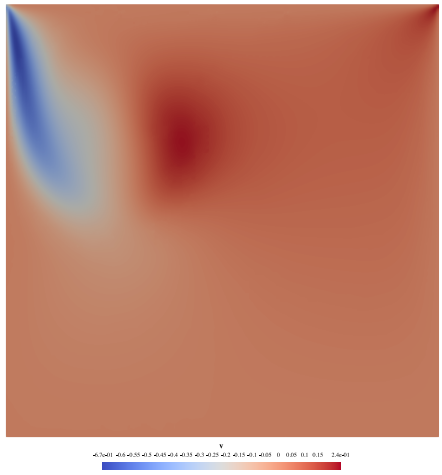
(a) $t = 2$



(b) $t = 2$

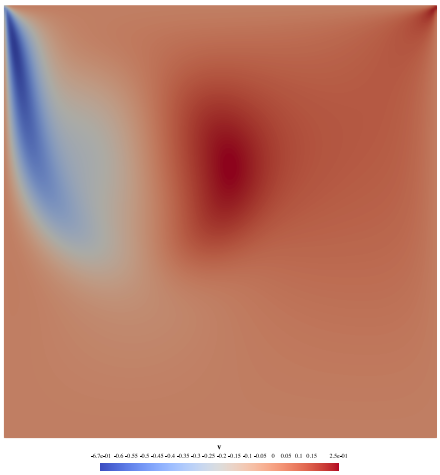


(c) $t = 4$

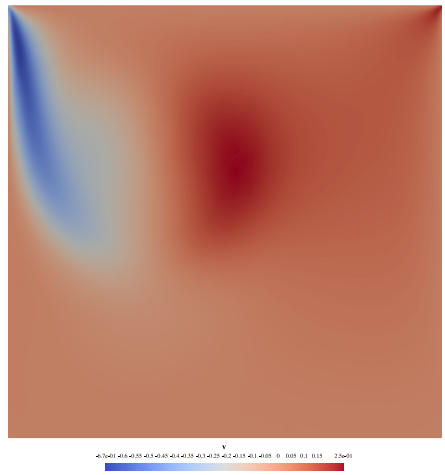


(d) $t = 4$

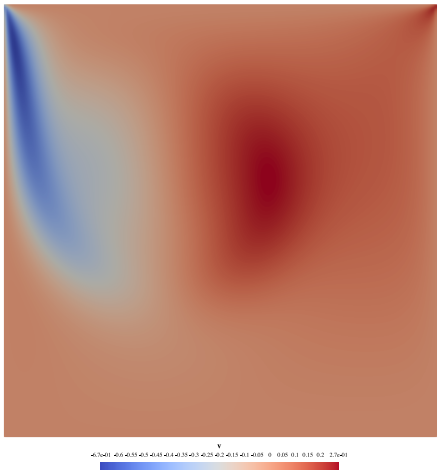
Figure 6.6: Lid-driven cavity. Evolution of the vertical velocity component for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



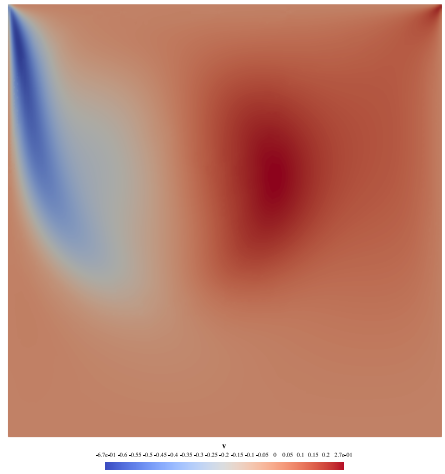
(e) $t = 6$



(f) $t = 6$

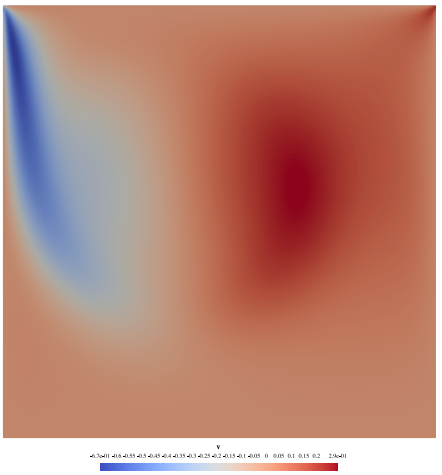


(g) $t = 8$

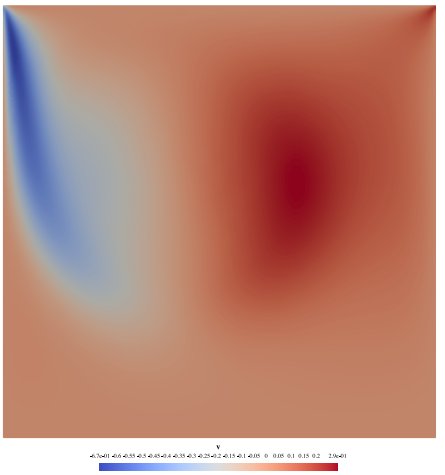


(h) $t = 8$

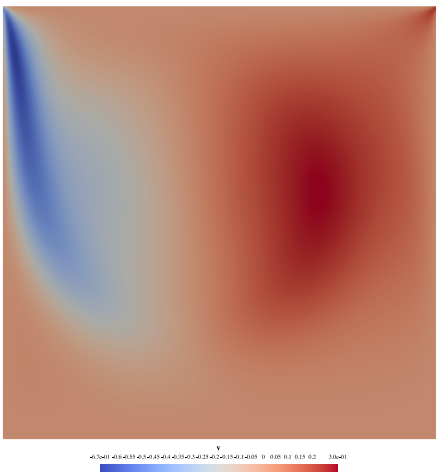
Figure 6.6: Lid-driven cavity. Evolution of the vertical velocity component for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



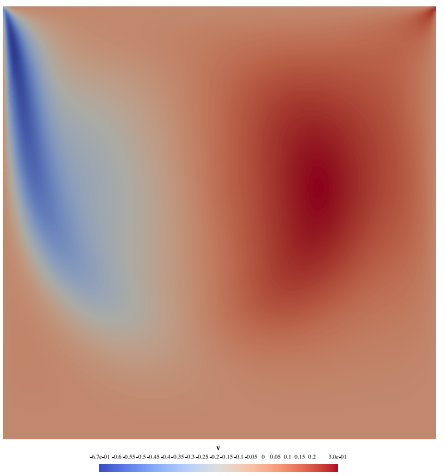
(i) $t = 10$



(j) $t = 10$

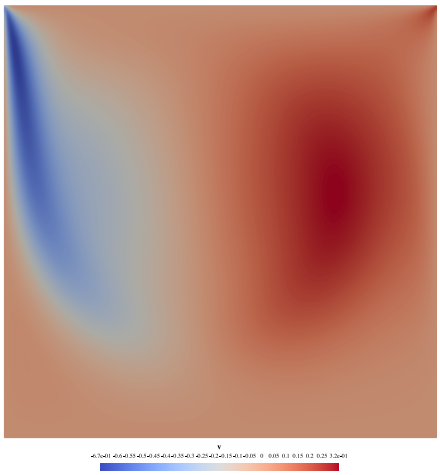


(k) $t = 12$

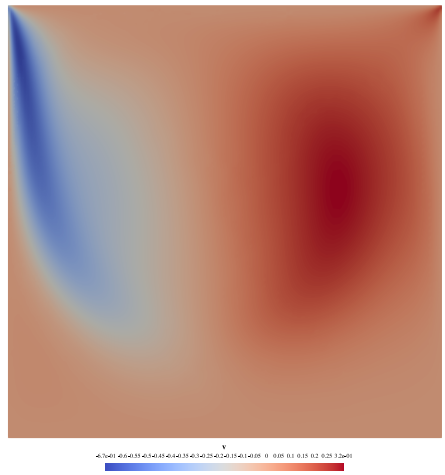


(l) $t = 12$

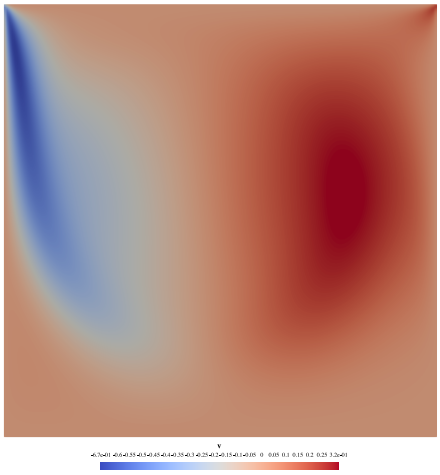
Figure 6.6: lid-driven cavity. evolution of the vertical velocity component for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



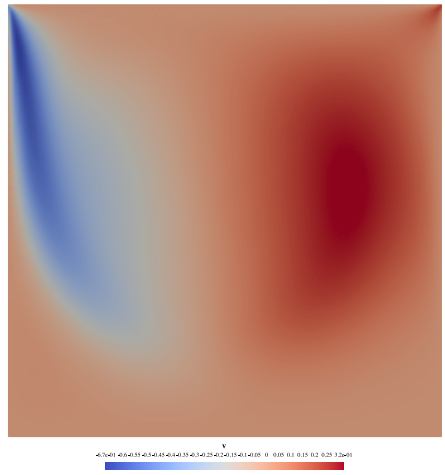
(m) $t = 14$



(n) $t = 14$

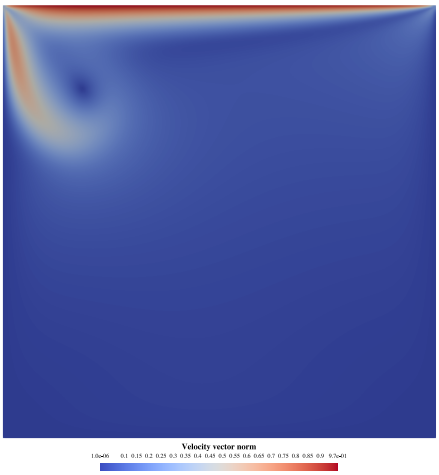


(o) $t = 16$

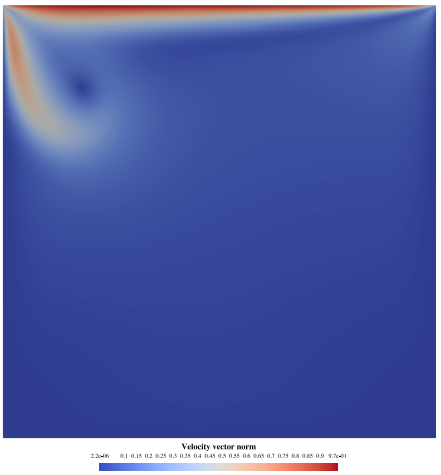


(p) $t = 16$

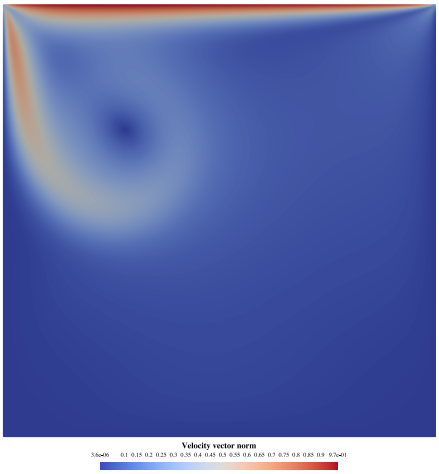
Figure 6.6: lid-driven cavity. evolution of the vertical velocity component for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



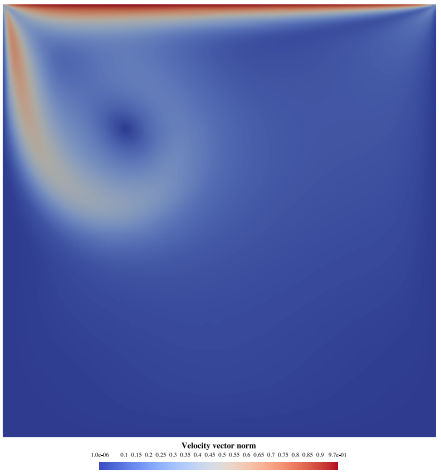
(a) $t = 2$



(b) $t = 2$

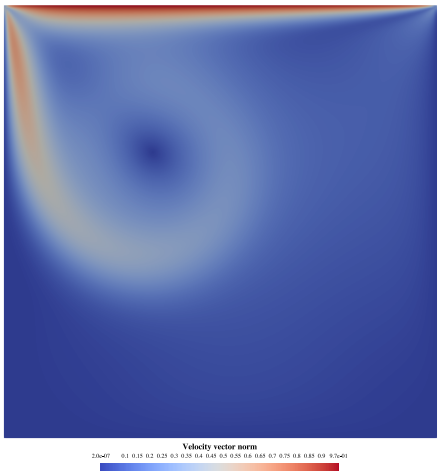


(c) $t = 4$

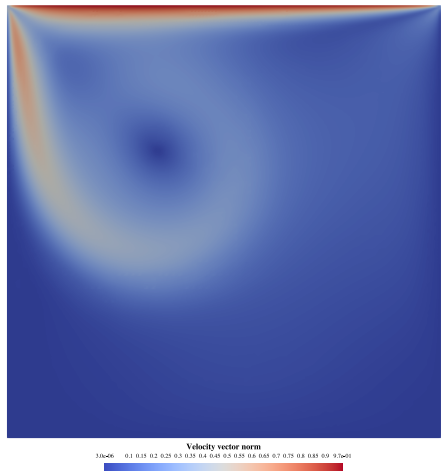


(d) $t = 4$

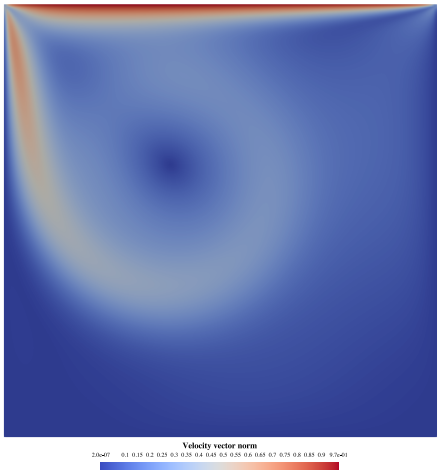
Figure 6.7: Lid-driven cavity. Evolution of the velocity norm for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



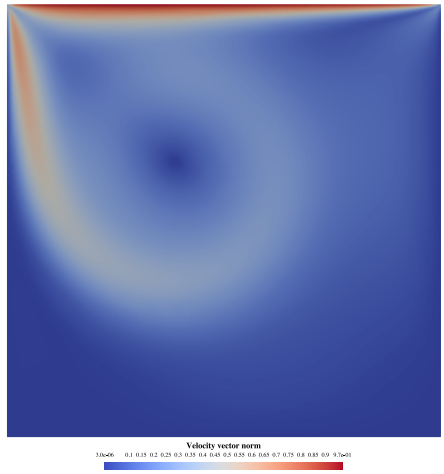
(e) $t = 6$



(f) $t = 6$

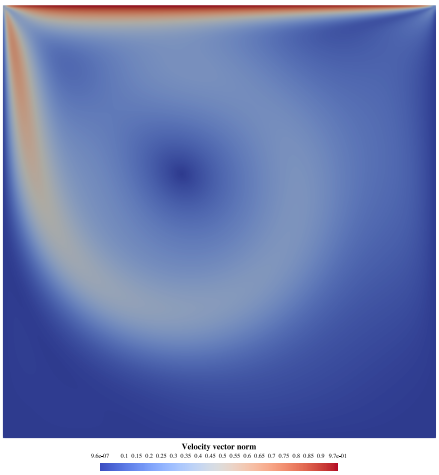


(g) $t = 8$

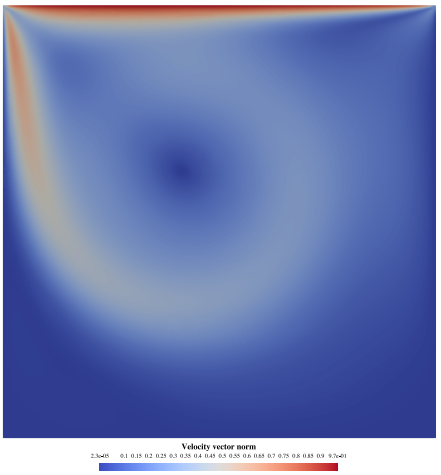


(h) $t = 8$

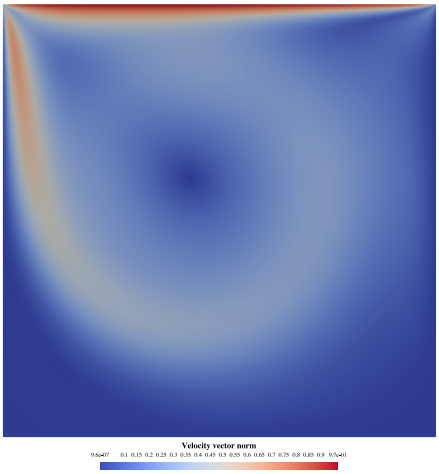
Figure 6.7: Lid-driven cavity. Evolution of the velocity norm for $Re = 1000$. Comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



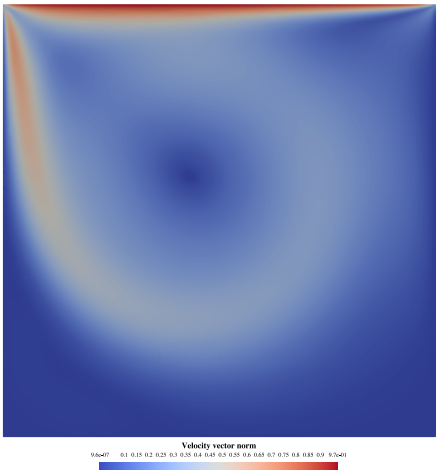
(i) $t = 10$



(j) $t = 10$

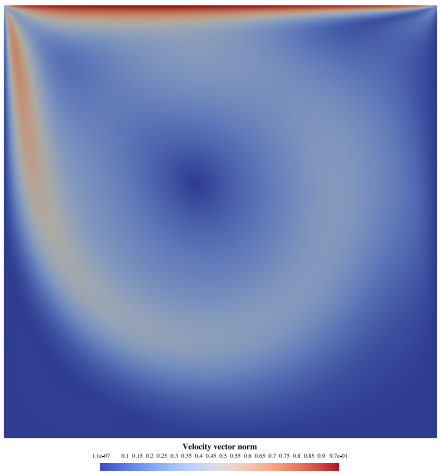


(k) $t = 12$

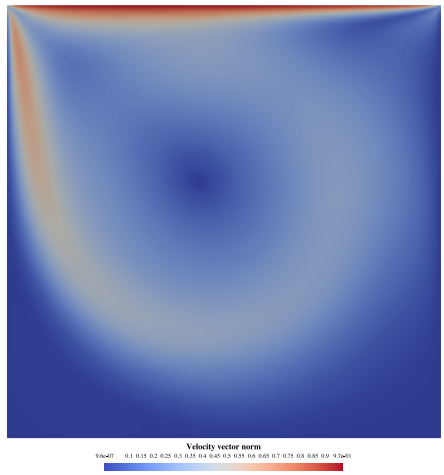


(l) $t = 12$

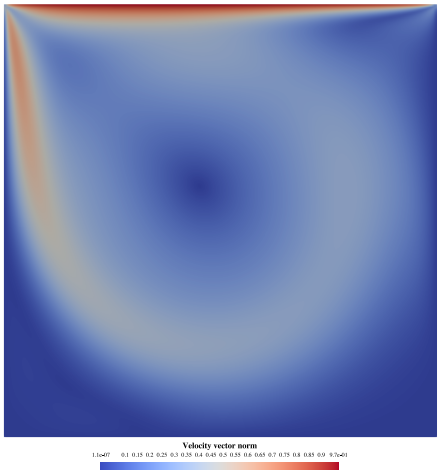
Figure 6.7: lid-driven cavity. evolution of the velocity norm for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



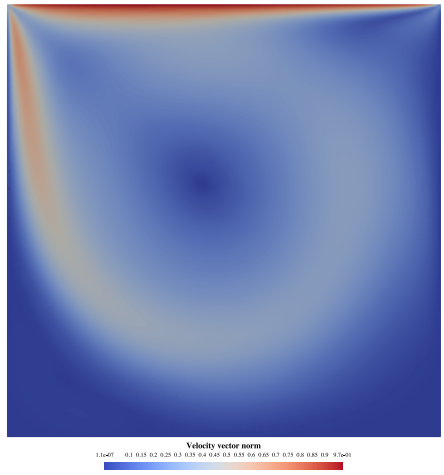
(m) $t = 14$



(n) $t = 14$

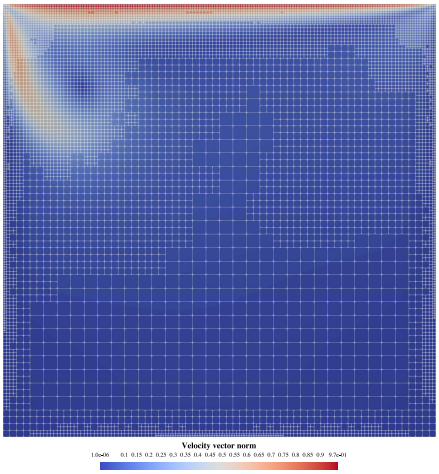


(o) $t = 16$

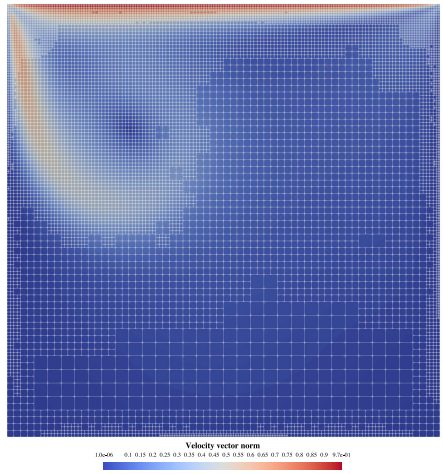


(p) $t = 16$

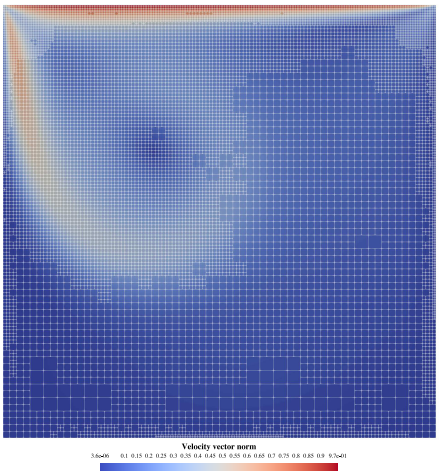
Figure 6.7: lid-driven cavity. evolution of the velocity norm for $re = 1000$. comparisons between the uniform grid computation (left) and the adapted grid computation (right). $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



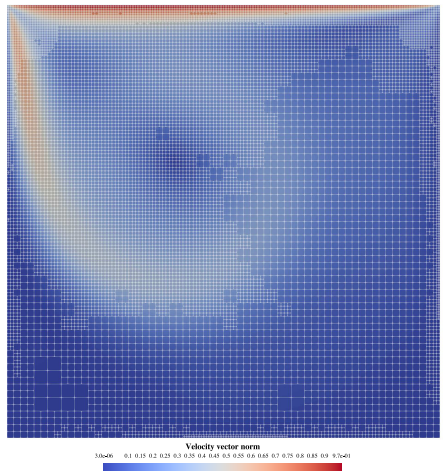
(a) $t = 2$



(b) $t = 4$

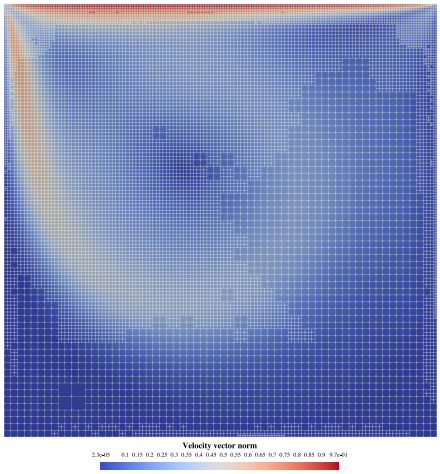


(c) $t = 6$

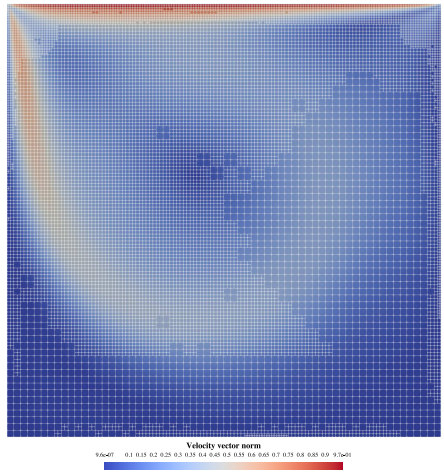


(d) $t = 8$

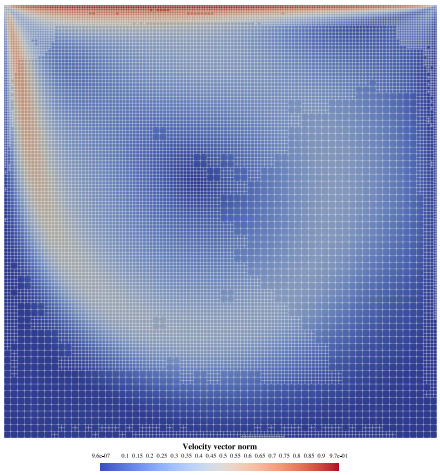
Figure 6.8: Lid-driven cavity. Evolution of the velocity norm for $Re = 1000$. Adapted grid. $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$



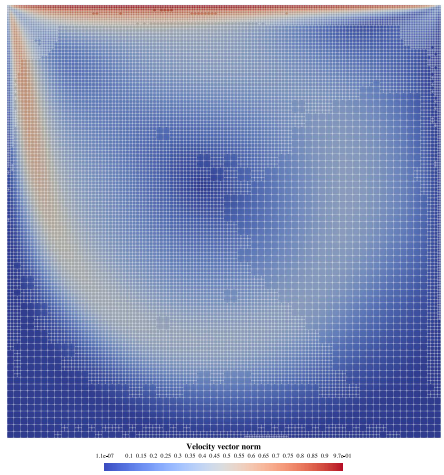
(e) $t = 10$



(f) $t = 12$



(g) $t = 14$



(h) $t = 16$

Figure 6.8: Lid-driven cavity. Evolution of the velocity norm for $Re = 1000$. Adapted grid. $l_{max} = 8$, $\varepsilon = 5 \times 10^{-3}$

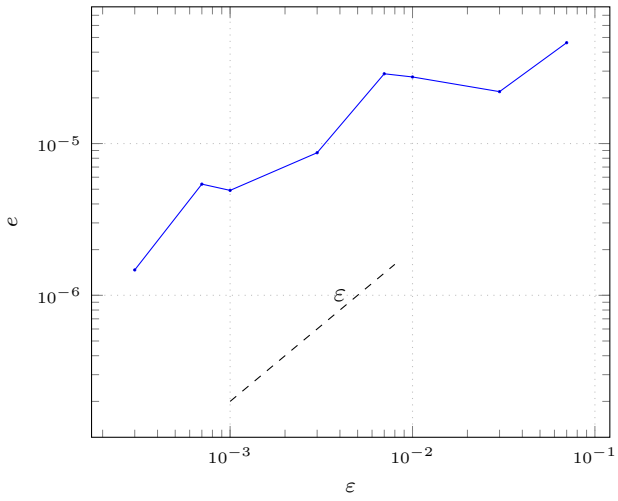


Figure 6.9: Lid-driven cavity. L^2 norm of the multiresolution error versus the threshold parameter ε for the horizontal component of the velocity

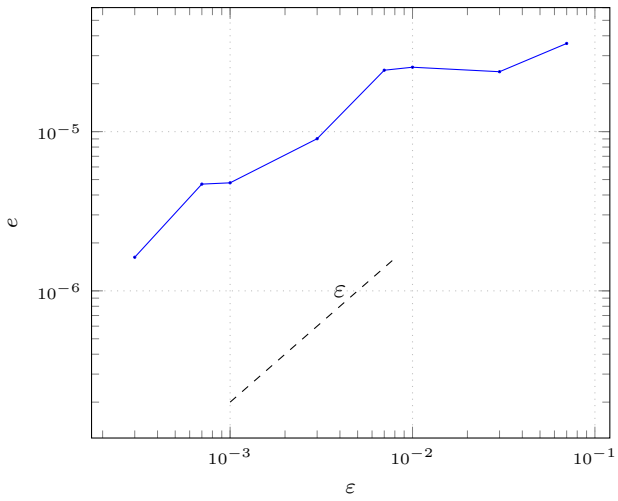


Figure 6.10: Lid-driven cavity. L^2 norm of the multiresolution error versus the threshold parameter ε for the vertical component of the velocity

performance computing, but rather to showcase the ability of the adaptive multiresolution method to obtain correct and precise simulation results. Both computational settings, the uniform grid and the adaptive multiresolution all share the same routines for the linear space operations (matrix–vector multiplication, vector–vector addition, vector–vector inner product, linear solver): we use the optimized PETSc libraries [BAA+19b, BAA+19a, BGMS97] which are written in C. But the data structures and routines needed for the adaptive multiresolution implementation in the multiresolution simulations are written in Python, for sequential computations only. What is more, we built a specific code for the uniform grid computation, that does not suffer at all from the overhead of the multiresolution. What is more, this code runs entirely on PETSc, and is fairly optimized compared to the adaptive multiresolution code that relies heavily on Python. This can be observed with the last thresholding value, where even though we obtain a 35% compression rate, the adapted computation takes more time than the uniform one. We can expect an even greater speedup with a high performance computing implementation of the multiresolution strategy.

6.4.2 Two-dimensional counter-rotating gaussian vortices

The second test case is the numerical simulation of two-dimensional counter-rotating gausian vortices. We discretize equations (2.1) on a uniform or adaptive grid with the spatial scheme (2.5), in an open bounded domain Ω in $2D$, where $\Omega =]-5,5[\times]-5,10[$. The initial configuration of the flow consists of two vortices (ω_1, ω_2) distributed on the horizontal axis with their centers located respectively at $(-0.5, 0)$ and at $(0.5, 0)$. They have the following shapes:

$$\omega_1 = \frac{\varphi}{\pi r_0^2} \exp\left(-\frac{(x-x_1)^2+y^2}{r_0^2}\right)$$
$$\omega_2 = \frac{\varphi}{\pi r_0^2} \exp\left(-\frac{(x-x_2)^2+y^2}{r_0^2}\right)$$

where the circulation φ is set at $\varphi = 10$ and $r_0 = 0.35$. We use free-flow boundary conditions, and set $\nu = 10^{-2}$.
We fix the thresholding parameter at $\varepsilon = 1 \times 10^{-3}$, the level max $l_{max} = 8$ for the finest grid, the timestep $h = 5\times10^{-3}$ and compute from $t = 0$ until $t = 5$.

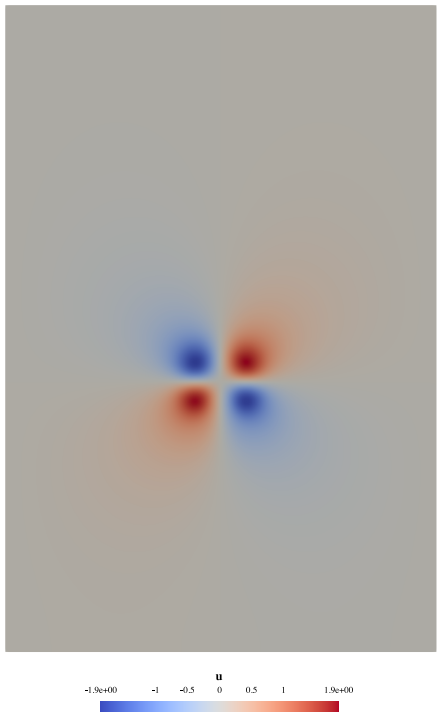
Figures (6.11), (6.12), (6.13) show the comparison between the uniform computation and the adapted computation, for the horizontal and vertical components of the velocity, and the velocity norm, respectively. There is a very good match between both computations, and the adaptive scheme produces good qualitative results. Figures (6.14) show the evolution of the adaptive grid, and we

ε	3.10^{-2}	1.10^{-2}	7.10^{-3}	3.10^{-3}	1.10^{-3}	7.10^{-4}	3.10^{-4}	1.10^{-4}
τ	94.71%	90.60%	89.0%	83.09%	73.74%	69.89%	60.37%	44.05%
Speedup	×11.94	×7.41	×6.51	×4.44	×2.80	×2.40	×1.71	×1.22

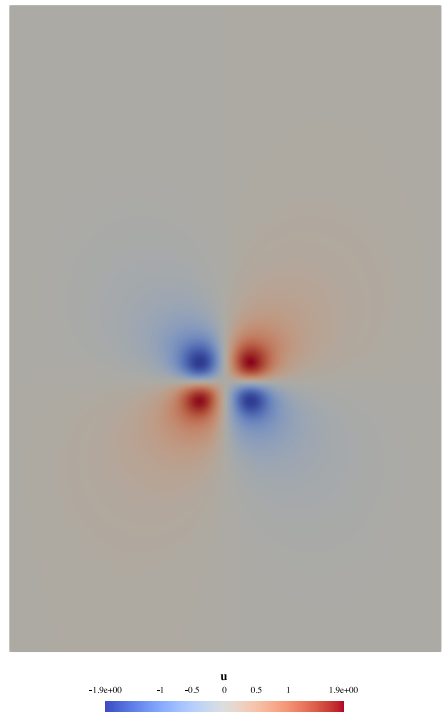
Table 6.2: Counter-rotating gaussian vortices. Comparison between the uniform grid and multiresolution adaptive grid computing times for different thresholding parameters. Uniform grid with $l^{max} = 8$. For a given value ε of the thresholding parameter, the uniform grid computation is *speedup* times longer than the computation with adaptive multiresolution. τ is the compression rate.

can verify that the level of refinement is directly related to the variation in the velocity norm, as expected.
Next, we check the ability of the code to control the multiresolution error. We fix the maximum grid level $l_{max} = 8$ for the finest grid, the timestep $h = 2{\times}10^{-3}$ and compute from $t = 0$ until $t = 2$ for different value of the thresholding parameter. At the end of the computation, we compute the L^2 norm of the error between the adapted grid solution, and the solution computed with the same features but on a uniform grid. Figures (6.15) and (6.16) show the evolution of the error versus the thresholding parameter for the horizontal and vertical components of the velocity, respectively. We see that we effectively control the adaptive multiresolution error, that scales as a multiple of the thresholding parameter.

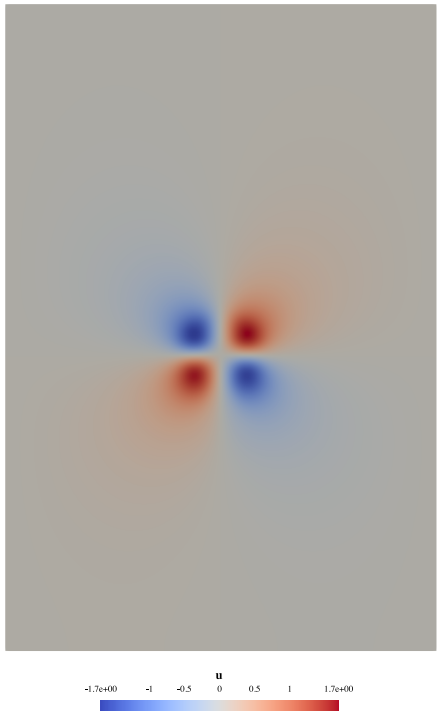
Finally, we assess the computational gain of the adaptive scheme.
Since $l_{max} = 8$, we have 65536 mesh cells on the uniform finest grid.
Table (6.2) shows some impressive results in the acceleration of the multiresolution computations, similar to the results obtained with the lid-driven cavity test.



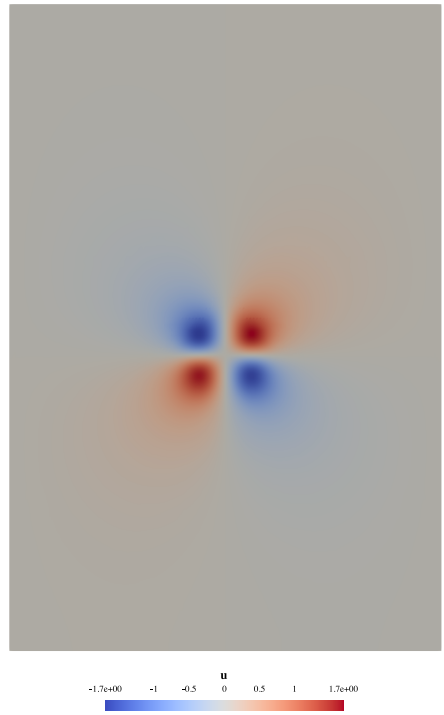
(a) $t = 1$



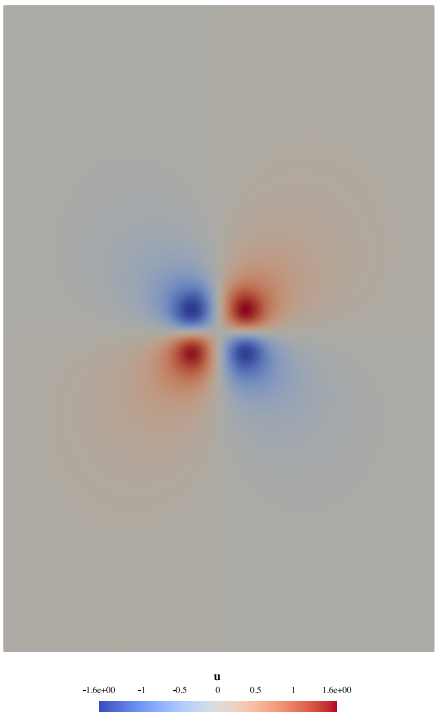
(b) $t = 1$



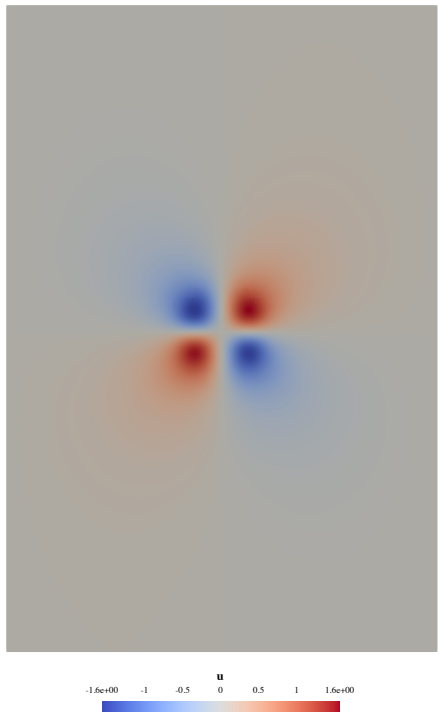
(c) $t = 1.5$



(d) $t = 1.5$

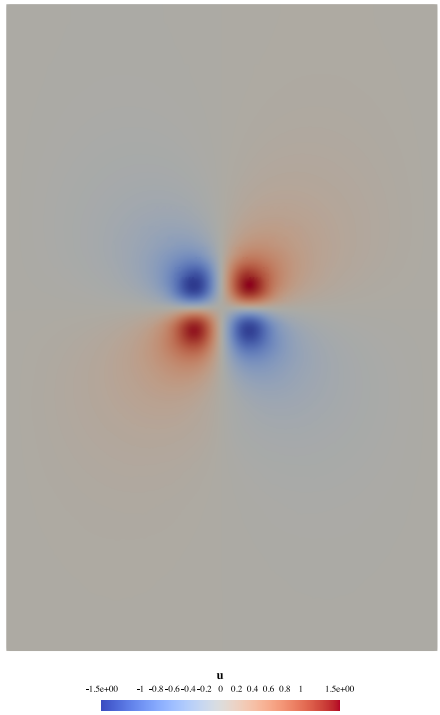


(e) $t = 2$

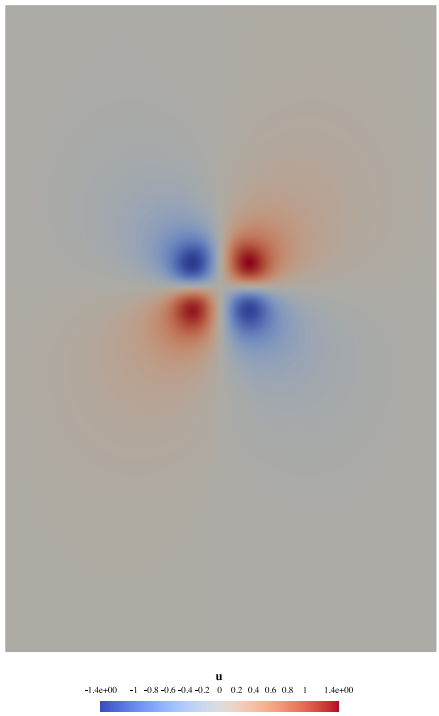


(f) $t = 2$

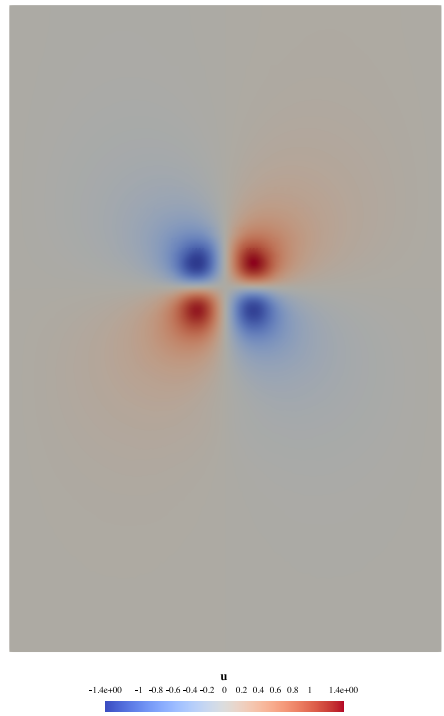
(g) $t = 2.5$



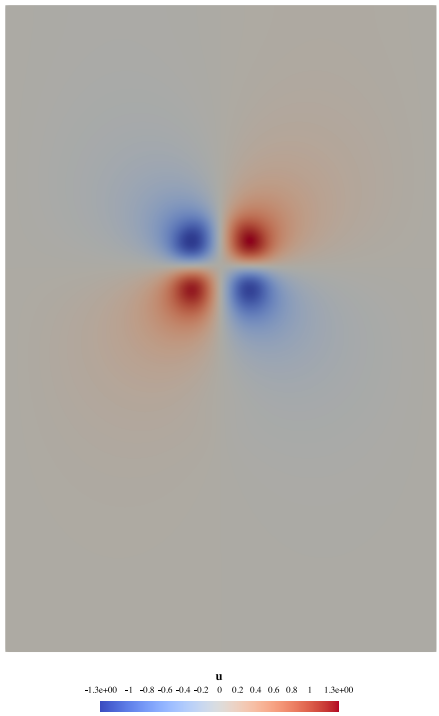
(h) $t = 2.5$



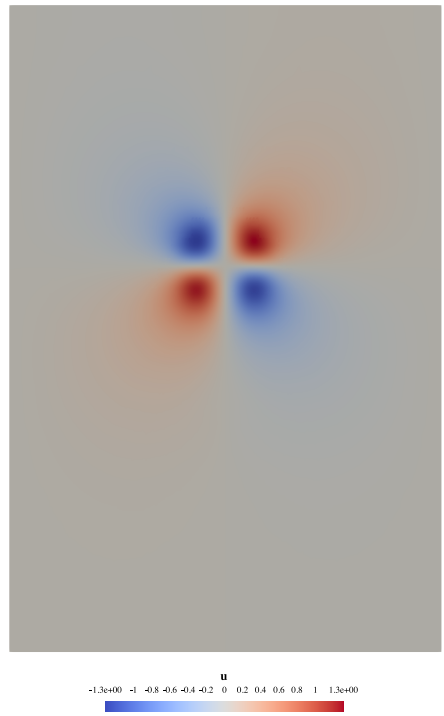
(i) $t = 3$



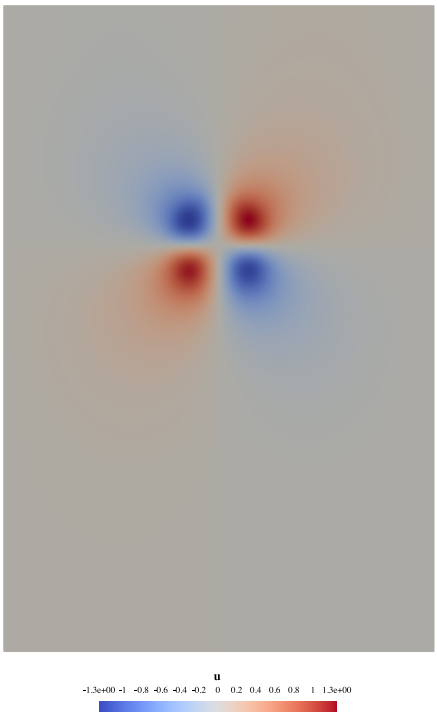
(j) $t = 3$



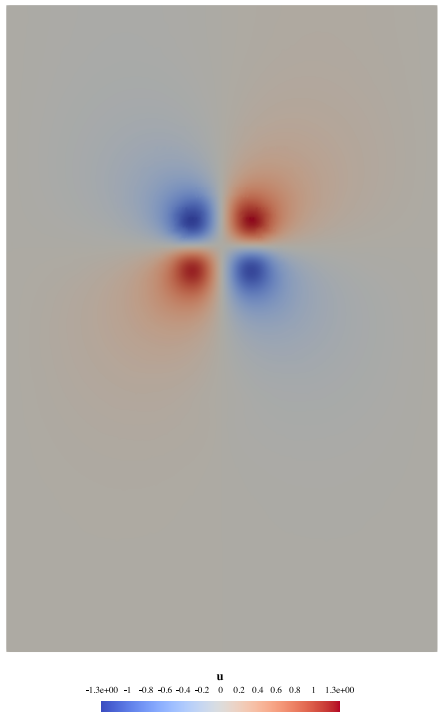
(k) $t = 3.5$



(l) $t = 3.5$

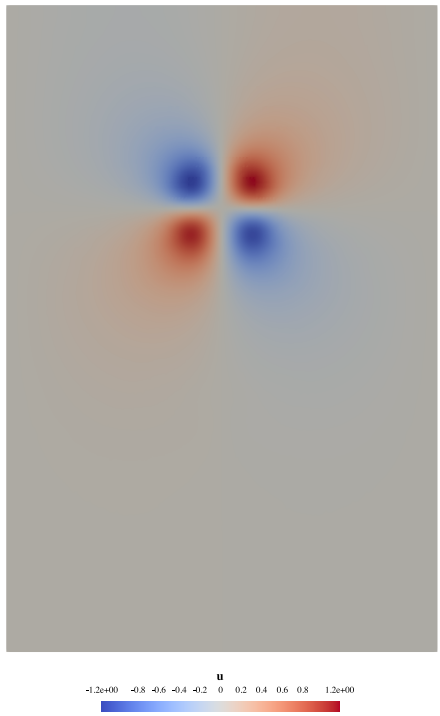


(m) $t = 4$

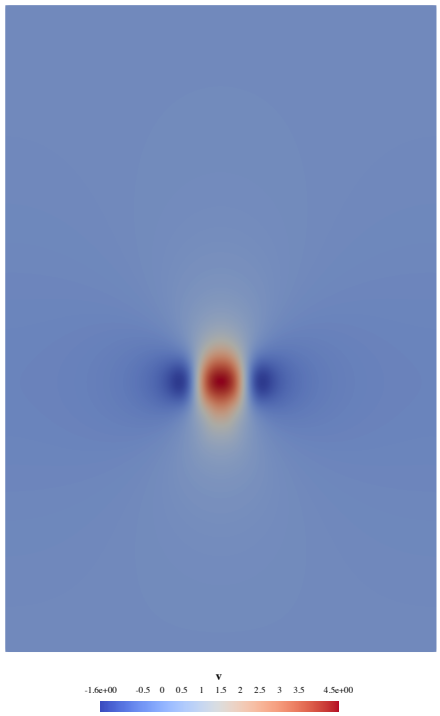


(n) $t = 4$

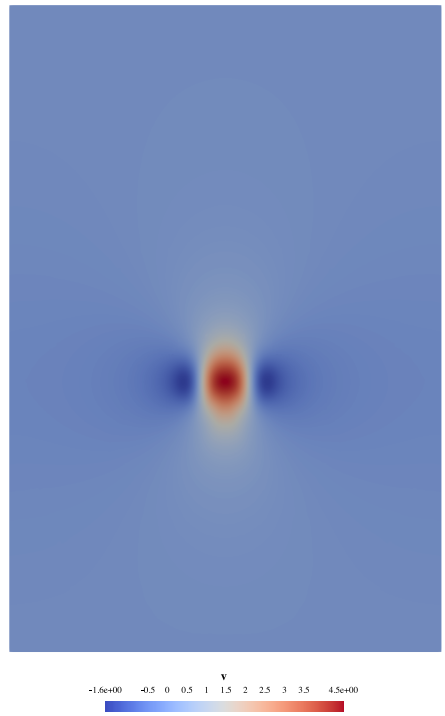
(o) $t = 5$



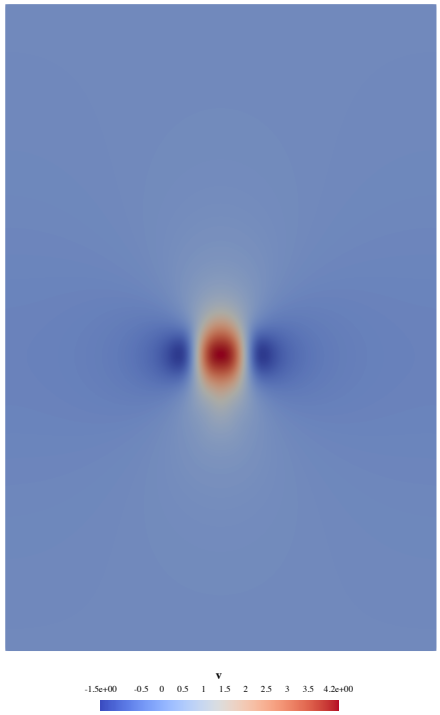
(p) $t = 5$



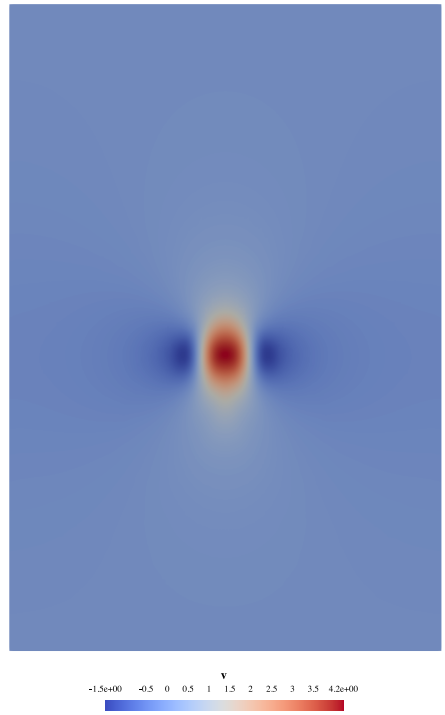
(a) $t = 1$



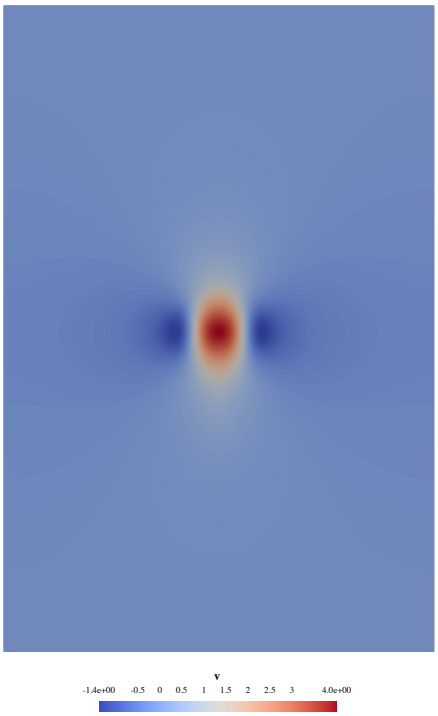
(b) $t = 1$



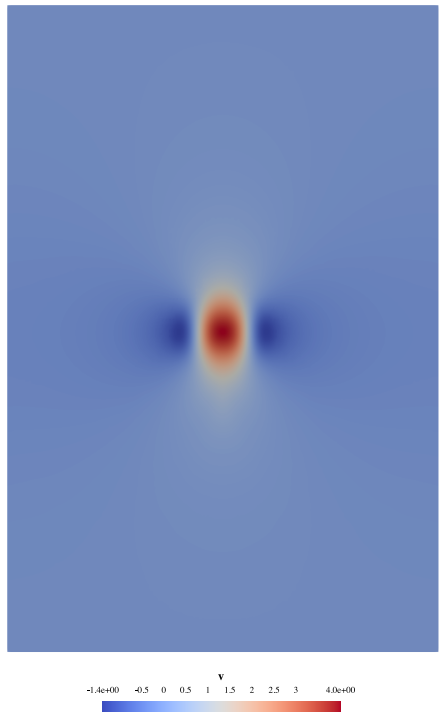
(c) $t = 1.5$



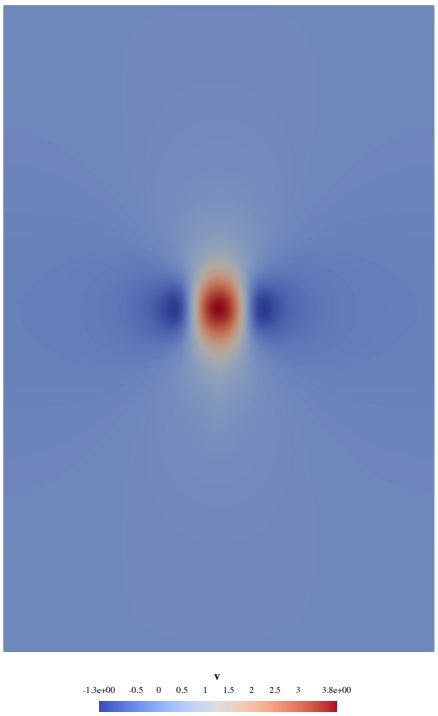
(d) $t = 1.5$



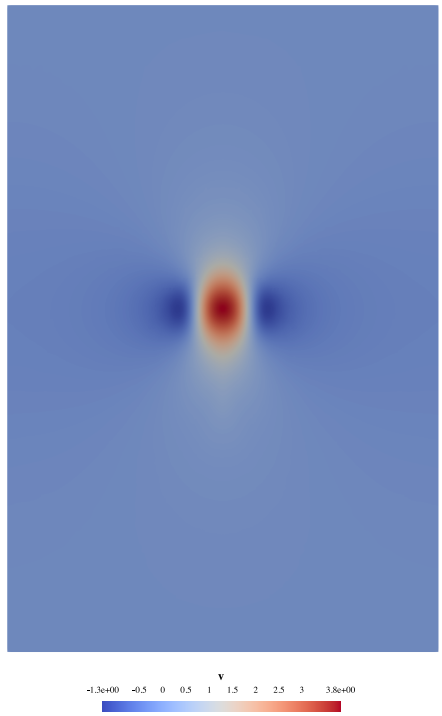
(e) $t = 2$



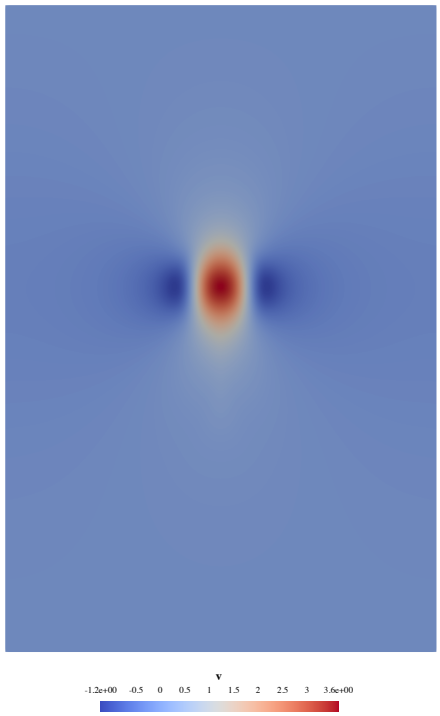
(f) $t = 2$



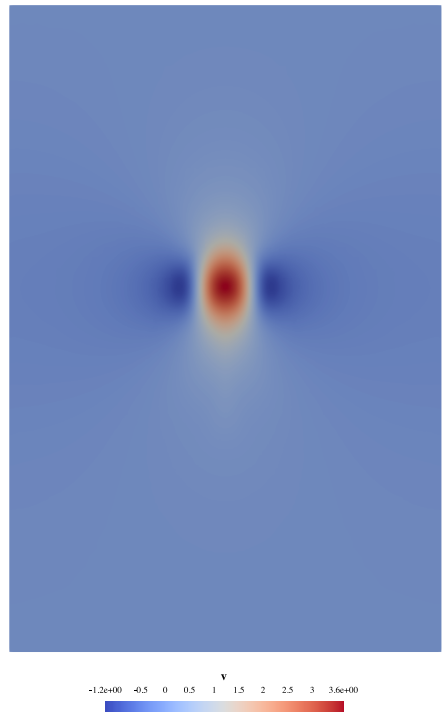
(g) $t = 2.5$



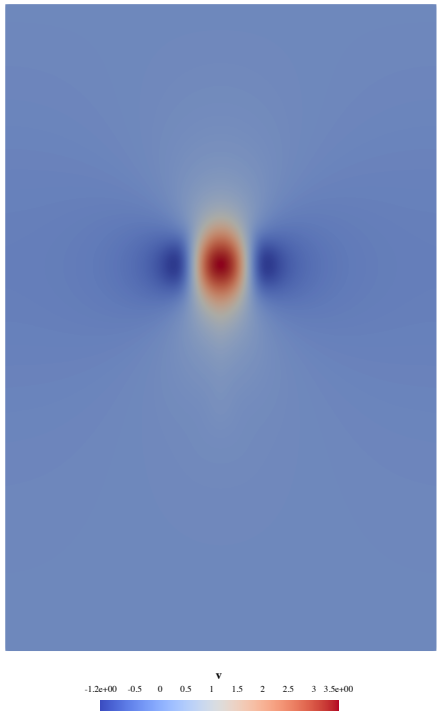
(h) $t = 2.5$



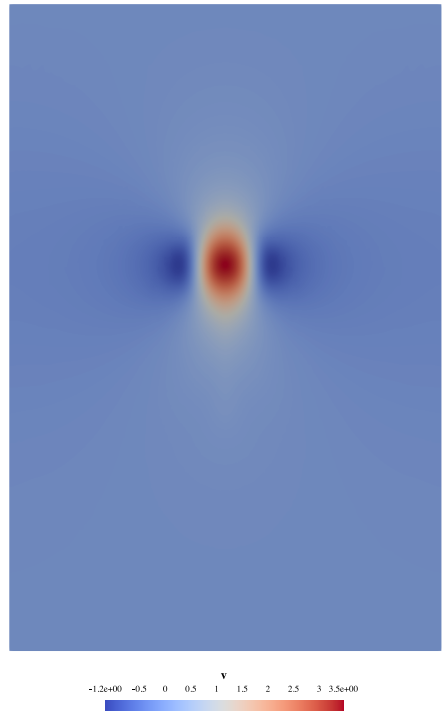
(i) $t = 3$



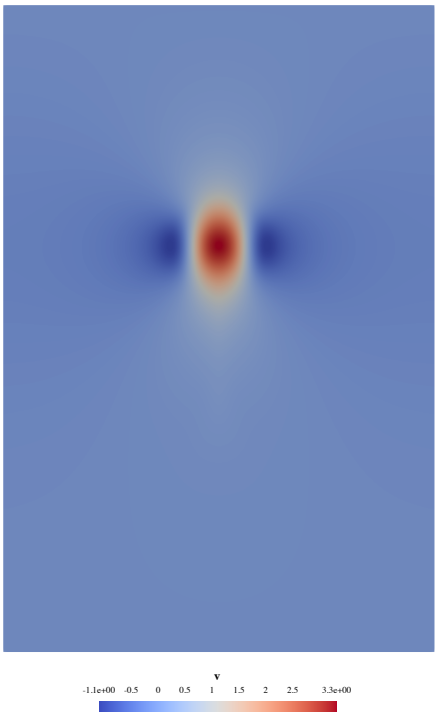
(j) $t = 3$



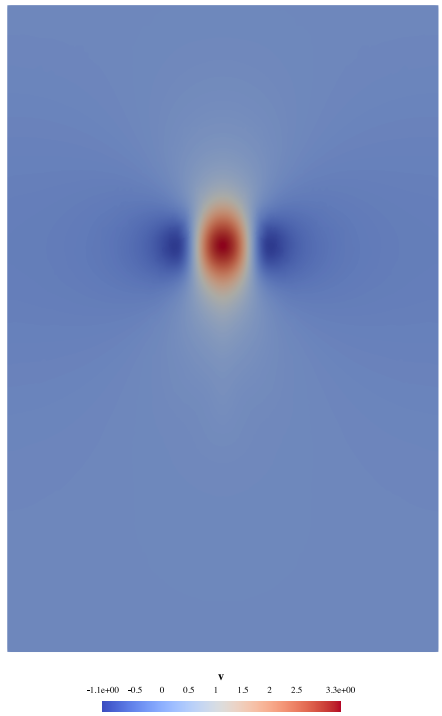
(k) $t = 3.5$



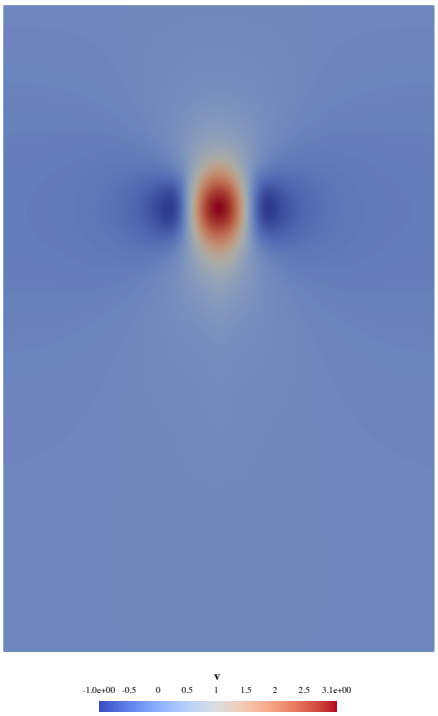
(l) $t = 3.5$



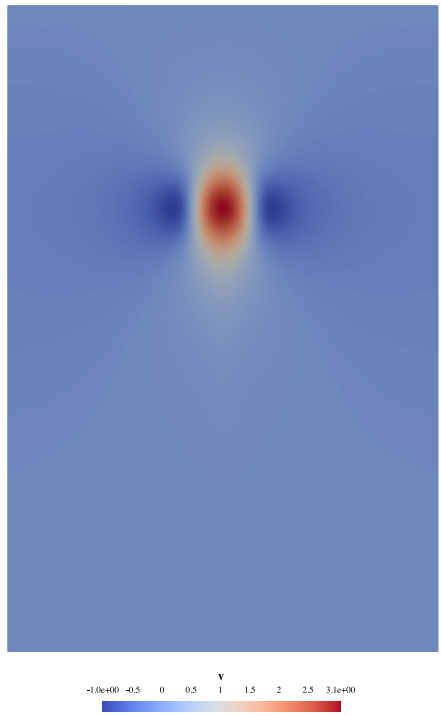
(m) $t = 4$



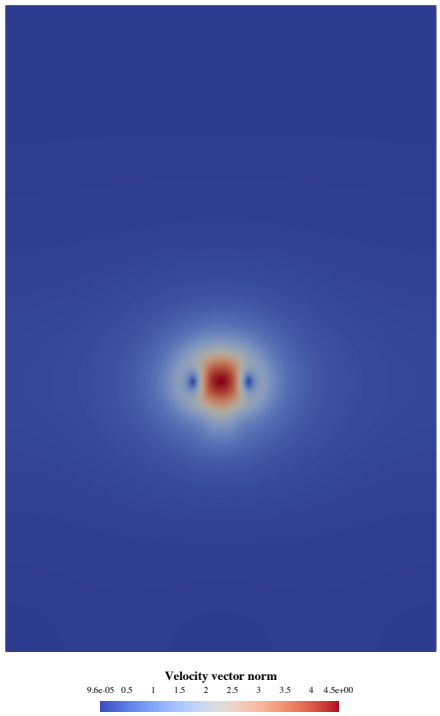
(n) $t = 4$



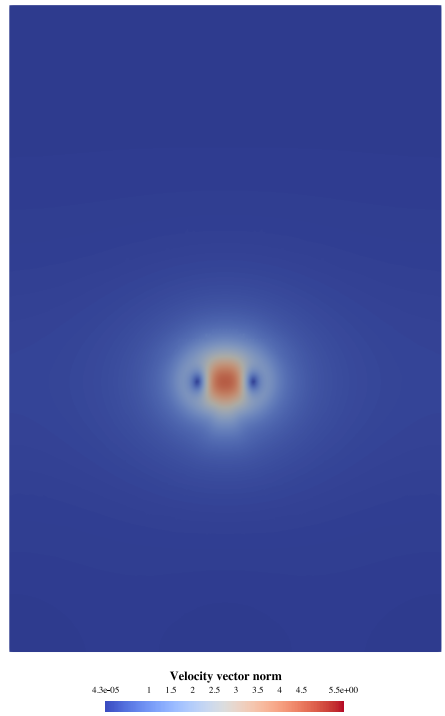
(o) $t = 5$



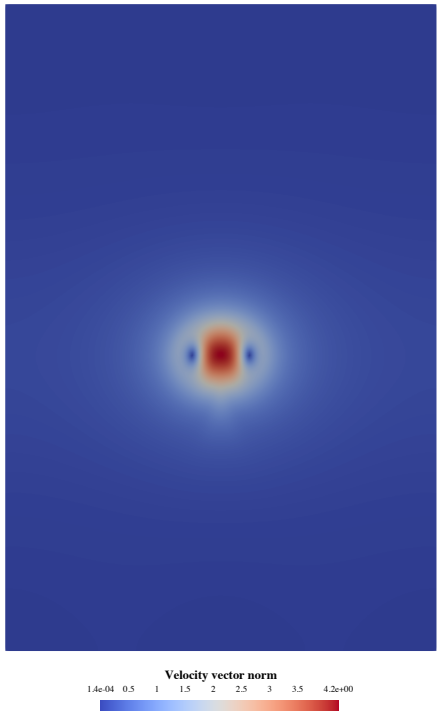
(p) $t = 5$



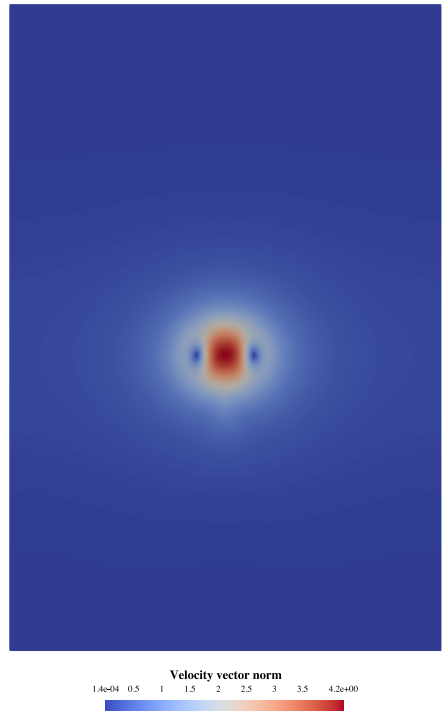
(a) $t = 1$



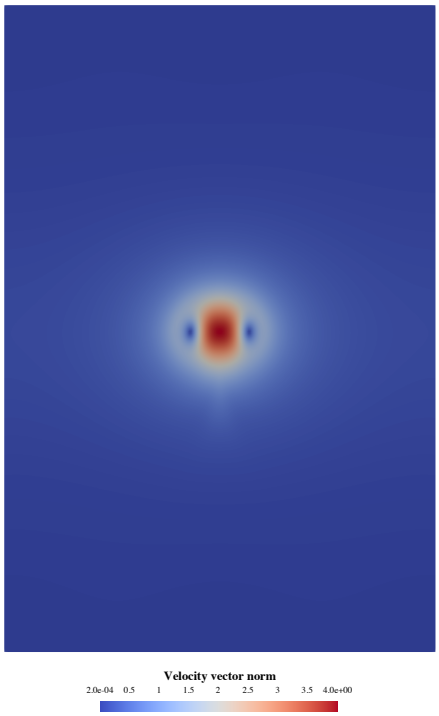
(b) $t = 1$



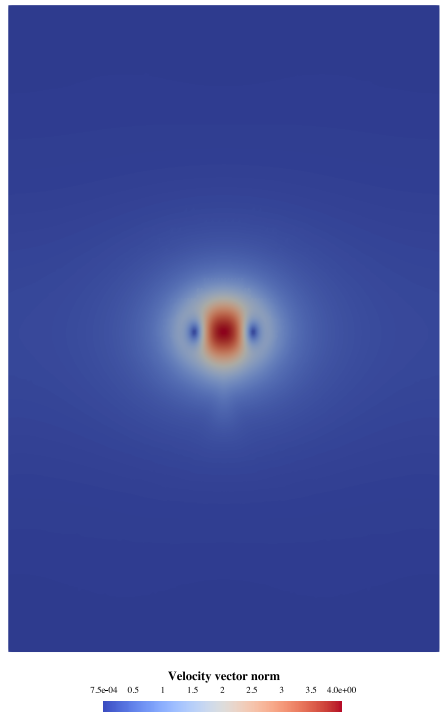
(c) $t = 1.5$



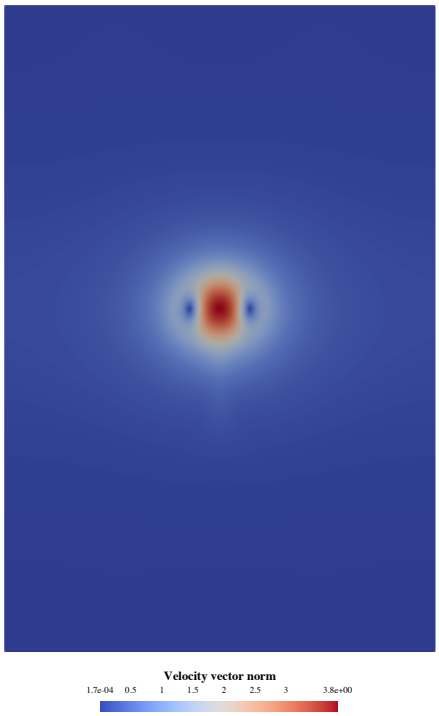
(d) $t = 1.5$



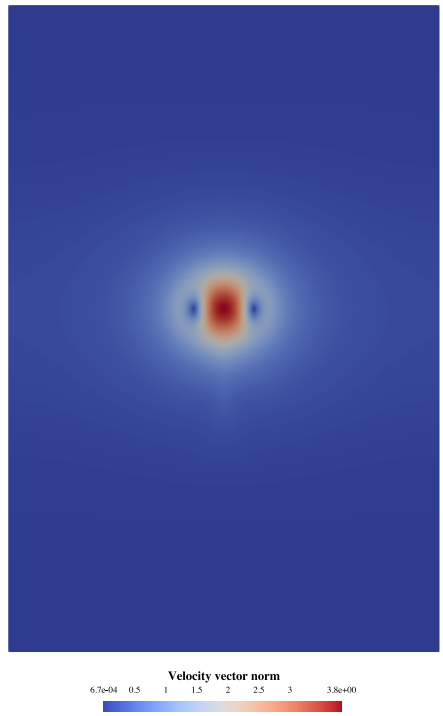
(e) $t = 2$



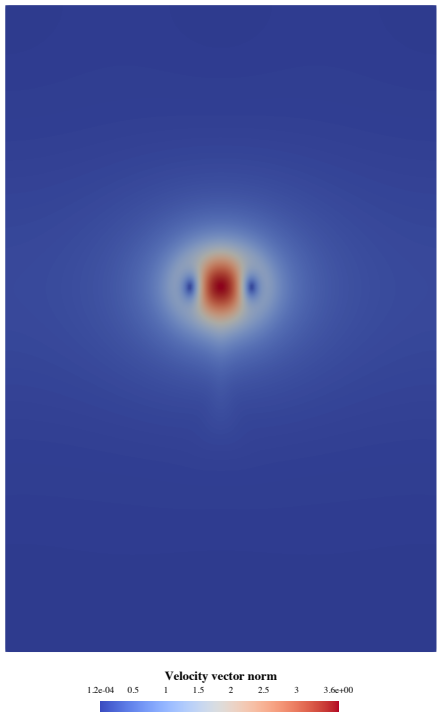
(f) $t = 2$



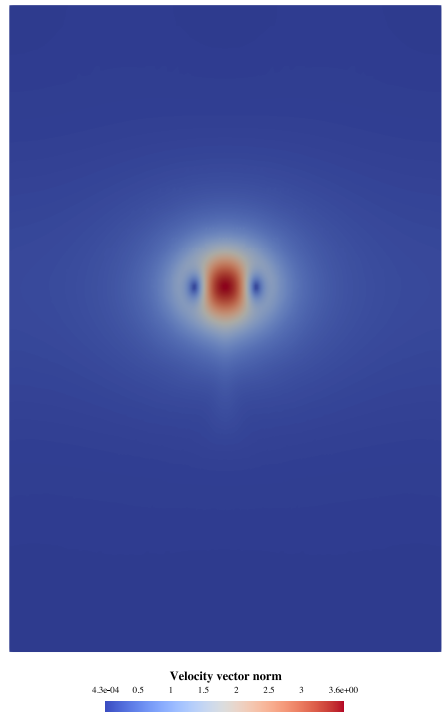
(g) $t = 2.5$



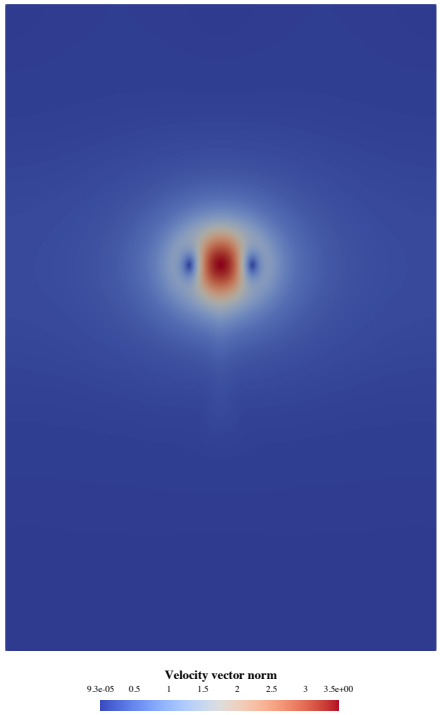
(h) $t = 2.5$



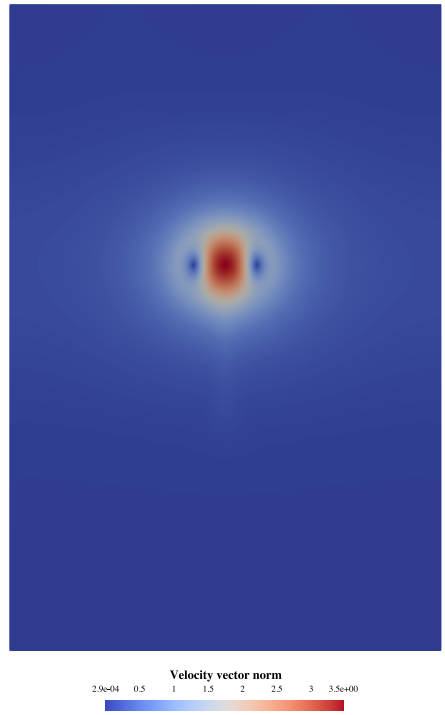
(i) $t = 3$



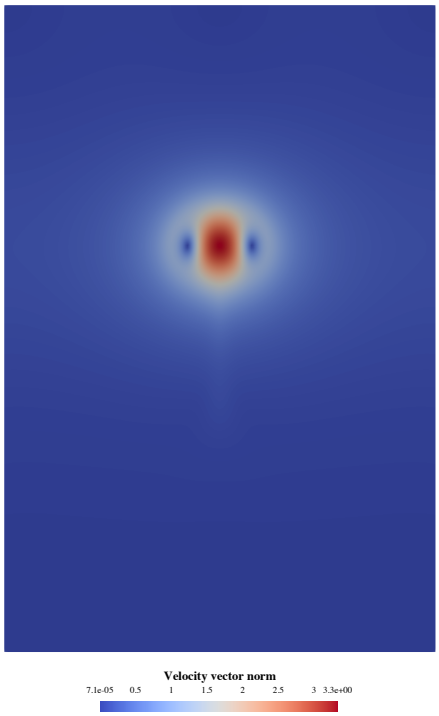
(j) $t = 3$



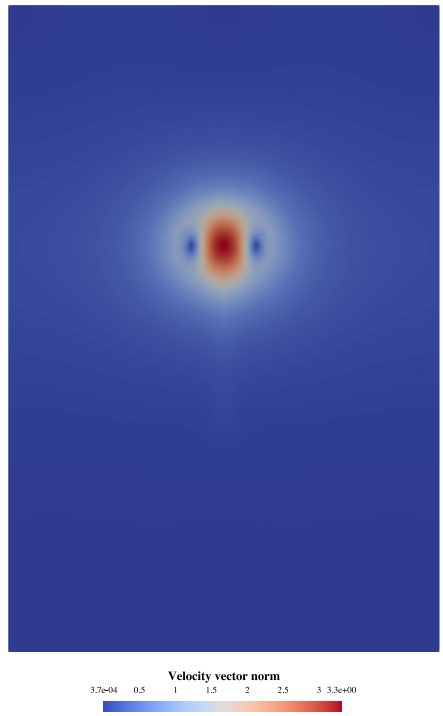
(k) $t = 3.5$



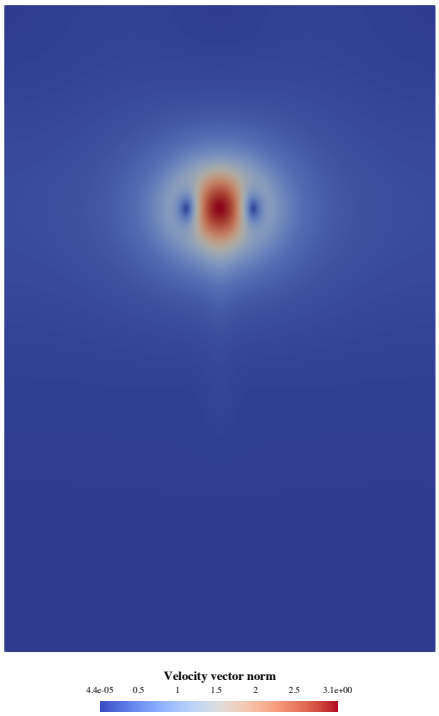
(l) $t = 3.5$



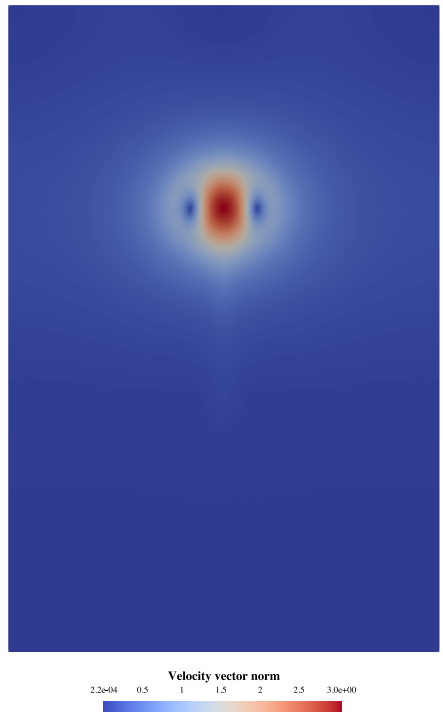
(m) $t = 4$



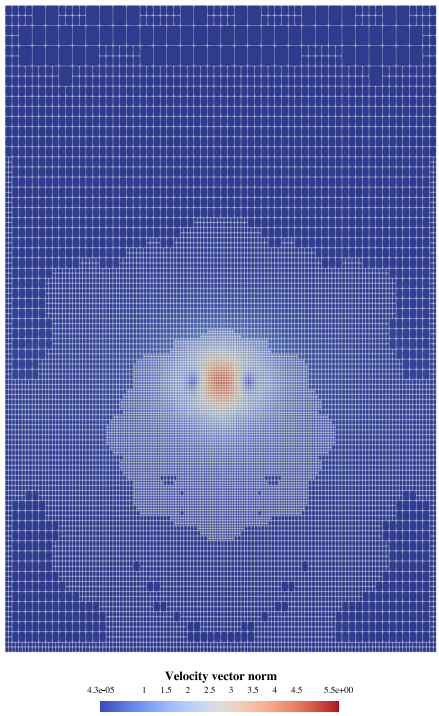
(n) $t = 4$



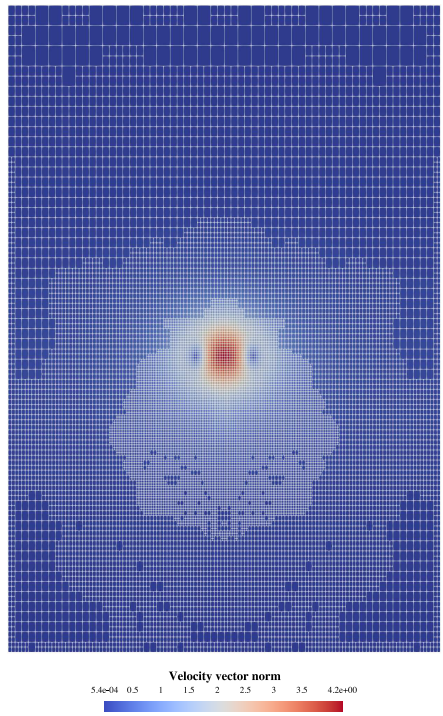
(o) $t = 5$



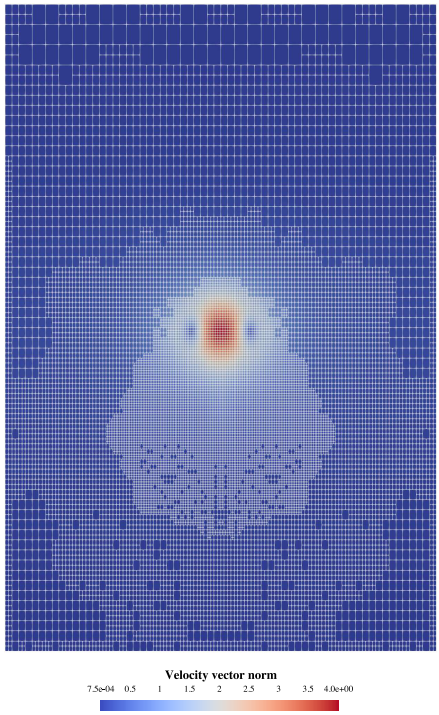
(p) $t = 5$



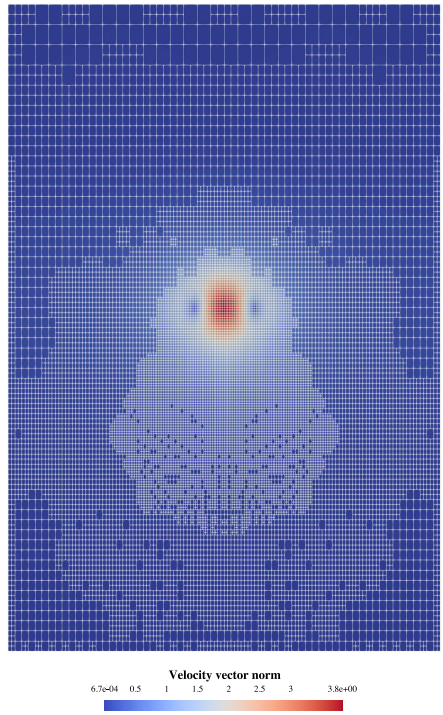
(a) $t = 1$



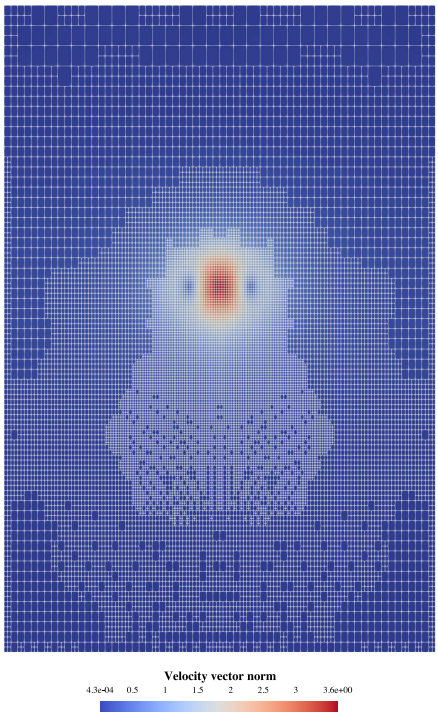
(b) $t = 1.5$



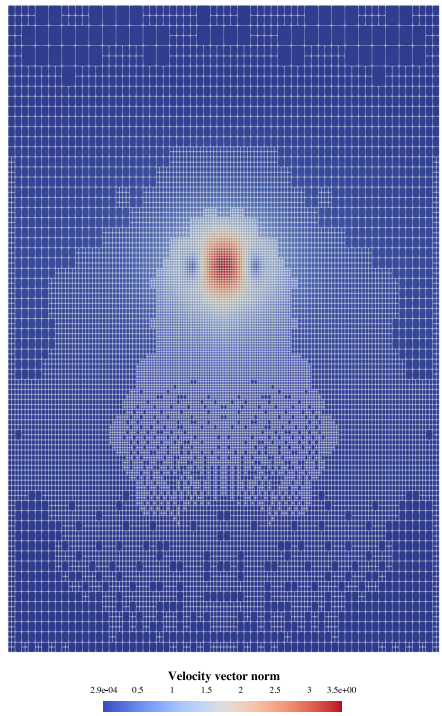
(c) $t = 2$



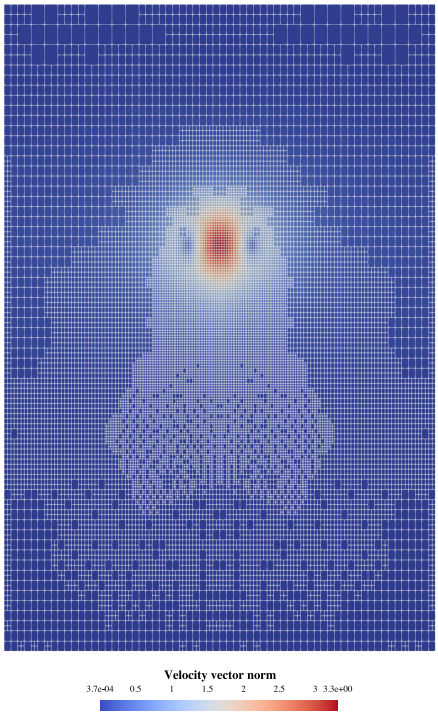
(d) $t = 2.5$



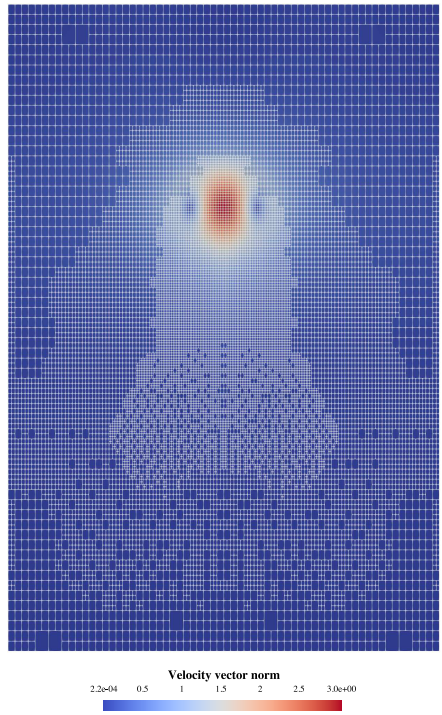
(e) $t = 3$



(f) $t = 3.5$



(g) $t = 4$



(h) $t = 5$

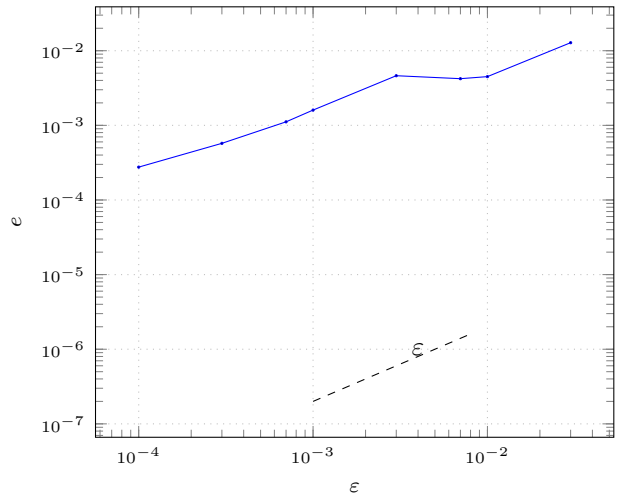


Figure 6.15: Counter-rotating gaussian vortices. L^2 norm of the multiresolution error versus the threshold parameter ε for the horizontal component of the velocity

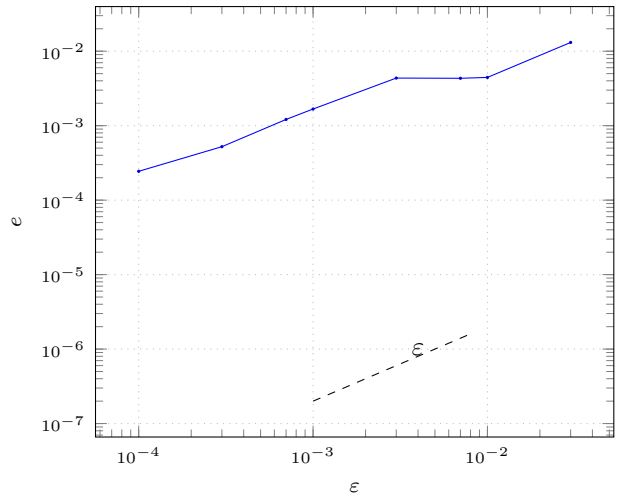


Figure 6.16: Counter-rotating gaussian vortices. L^2 norm of the multiresolution error versus the threshold parameter ε for the vertical component of the velocity

Chapter 7

Scalar transport application: transport of a passive scalar on dual grids

This chapter focuses on an application of the strategy developed throuhgout this work to the transport of a passive scalar by an incompressible flow. It is a phenomenon that constantly occurs in reactive flows, although generally coupled to many other physical processes. But we believe that this simplification is a good starting point, when trying to develop a new strategy to efficiently solve reactive flows, for two reasons mainly: (i) in many fluid flow simulations, the phenomena of interest are the transport and diffusion of scalar properties, for example the species concentration, that play a little role in the flow conditions, and (ii) passive scalar transport is the first step when considering the interaction of a flow field with various other components. In the context of multi-scale reaction fronts, both the flow and the scalars can be characterized by an inhomogeneous spatial distribution, but at different scales. More specifically, depending on the Schmidt number, the characteristic spatial scales of the flow can be much larger than the scales of the transported species, and the latter require then a finer resolution grid to be accurately described compared to the former. Clearly, we should use an adpative grid technique to simulate such phenomena, but given the fact that resolving an incompressible flow is more time consuming than solving the advection-diffusion equations of the scalar transport, we add a useless computational overhead if we simulate both processes on the finest grid related to the species. Also, we wish to be able to control the spatial error related to the grid adaptation. The multiresolution strategy offers us an efficient way to do just that, and since we are now able to use this technique for the incompressible Navier-Stokes equations and the advection-diffusion ones, we can apply it to solve these two set of equations on *different* grids. We called this technique **multihierarchy multiresolution**

adaptive strategy (MMR), and the study of its efficiency has motivated an article recently published in Journal of Computational and Applied Mathematics [NMST19]. We will reproduce this article in what follows in its integral version. It is a good application framework for most of the discoveries made during this investigation.

HIGH ORDER TIME INTEGRATION AND MESH ADAPTATION WITH ERROR CONTROL FOR INCOMPRESSIBLE NAVIER-STOKES AND SCALAR TRANSPORT RESOLUTION ON DUAL GRIDS

Marc-Arthur N'Guessan, Marc Massot, Laurent Séries, and Christian Tenaud

Abstract. Relying on a building block developed by the authors in order to resolve the incompressible Navier-Stokes equation with high order implicit time stepping and dynamic mesh adaptation based on multiresolution analysis with collocated variables, the present contribution investigates the ability to extend such a strategy for scalar transport at relatively large Schmidt numbers using a finer level of refinement compared to the resolution of the hydrodynamic variables, while preserving space adaptation with error control. This building block is a key part of a strategy to construct a low-Mach number code based on a splitting strategy for combustion applications, where several spatial scales are into play. The computational efficiency and accuracy of the proposed strategy is assessed on a well-chosen three-vortex simulation.

Keywords. *Incompressible Navier-Stokes; high order implicit Runge Kutta; multiresolution analysis; dynamic mesh adaptation; scalar transport; dual grid with error control*

Mathematics Subject Classification. 65M08, 65M50, 76D05, 80A25, 80A32.

Journal of Computational and Applied Mathematics (2019) 112542
DOI: 10.1016/j.cam.2019.112542
Published online October 31, 2019.

7.1 Introduction

Numerical simulations of chemically reacting flows place a considerable strain on computational resources, because of the large spectrum of characteristic spatial and temporal scales involved in these phenomena. Furthermore, the Direct Numerical Simulation (DNS) of low-Mach combustion requires important computational resources, partly due to the highly refined meshes necessary to accurately describe the reactive fronts, but also due to the various numerical schemes, which involve costly linear algebra for Poisson solvers or implicit schemes [BS99]. One way to reduce the computational effort that has been investigated over the years is the use of Adaptive Mesh Refinement (AMR) [BO84] to spatially adapt the grid in the reactive fronts, thus reducing the number of unknowns [DB00a, ABC+98, PHB+98, SRN10].

Whereas such techniques have led to very interesting developments, one of the difficulties of AMR is the heuristics used in order to refine the mesh, which lead to a high compression level but hardly provide any error estimate. Our contribution also focuses on spatial mesh adaptation in order to reduce the memory of such simulation, but rather involves multiresolution (MR) analysis in order to obtain high compression [Pos01, CKMP03], with error control in space and time when coupled to an adaptive splitting technique [DDD+11]. In [CKMP03], Cohen *et al.* developed the algorithms to efficiently perform adaptive MR for systems of conservative laws in a finite volume context. They were implemented for example in [DDT+13] for the numerical simulations of premixed and diffusion flames, where the advection-diffusion-reaction problem was spatially discretized on a Cartesian grid with MR adaptation, whereas the flow field was provided analytically at each time step; a similar strategy was investigated in [RS05] for another application. Using the proposed splitting strategy implies that the missing building block was a solver for the hydrodynamics. We started with the incompressible Navier-Stokes equations and the use of multiresolution and finite volume on adapted grids and tree-data structures made the classical approach on staggered grids [HW65] or the resolution of the resulting differential algebraic equation (DAE) by a fractional-step method [Cho68, Tem69, KM85] rather impossible or low order. In [NSTM18] we developed a high-order time integration based on the Radau IIA Runge-Kutta method, and a finite-volume method coupled to adaptive multiresolution to solve incompressible flows on collocated grids.

One of the particularities of low-Mach combustion is the fact that the flow and the transported species involve different spatial and temporal scales. The ones describing the flow require more computational effort because they involve the numerical resolution of linear systems [HW65, GS00, RC83, Tem69, KM85, RP13]. But the characteristic spatial scales of the flow are often larger than the scales of the species, and one may exploit this fact by resolving these two sets of equations on different grids. This idea was used in [SRN10], where a full low-Mach combustion solver was designed, with the flow being solved on a uniform coarse grid, while the advection-reaction-diffusion of scalars were solved on a finer grid using AMR. See also [Sch03, Sch04] for an application of this technique for phase-field simulations. Our aim is to design a numerical strategy along the same lines, which adapts the mesh at different levels for the hydrodynamics variables and for the species equations with finer discretization, while sticking to error control based on MR on this dual grid.

Low-Mach number combustion is our goal, but in order to introduce the fundamentals of the approach, we rather focus on a simpler problem representative of the difficulties we will encounter and tackle the problem of a scalar transport at various Schmidt numbers by a flow field, which is a solution of the incompressible Navier-Stokes equation, where we introduce a numerical strategy relying

on a dual grid for both fields with error control based on MR. To this end, we design a $2D$ configuration inspired by the canonical interaction of vortex pair with the mixing layer of a passive reactant [ABC+98]. The strategy is assessed in this academic configuration in terms of accuracy and efficiency and we show that the proposed strategy allows to obtain large gains in terms of computational cost and memory trace, without tempering on the accuracy of the global solution even at relatively large Schmidt numbers.
The outline of the paper is the following. In section 7.2 the governing equations for an incompressible flow and the advection-diffusion of a passive scalar are presented, as well as the details of the mixing layer and vortex interaction. Then, in section 7.3 we expose our numerical strategy, namely the adaptive multiresolution algorithm, our spatial discretization strategy for both the flow and the scalar and finally the temporal discretization retained here. We assess the efficiency of this strategy to properly tackle the transported scalar problem at hand in section 7.4, and finally conclusions are drawn in section 7.5.

7.2 Governing equations

We consider the transport of a passive scalar $s(x, y, t)$ in a rectangular domain denoted Ω, of characteristic length L_0, by an incompressible fluid flow which is fully described by two variables, the velocity vector $\boldsymbol{u} = (u_i(x, y, z, t))_{i=1,2}$ and the pressure field $p(x, y, z, t)$. The time variable t varies between 0 and T. The flow momentum and mass balance equations read:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}^t \otimes \boldsymbol{u}) + \nabla p - \nu \, \Delta \boldsymbol{u} = \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases} \tag{7.1}$$

and are coupled to a transport equation for the scalar s given by:

$$\frac{\partial s}{\partial t} + \nabla \cdot (s\boldsymbol{u}) - \kappa \, \Delta s = \mathrm{f} \tag{7.2}$$

where ν is the kinematic viscosity of the fluid, κ is the diffusivity of the scalar, and $\boldsymbol{f}$ and f are source terms. Initial and boundary conditions are added to the system of equations and will be presented in the numerical results section. In order to mimic the situation in combustion applications where the characteristic length related to the flame inner layer is lower than the scales that have to be resolved in order to properly capture the velocity field [DB00a, SRN10], we will consider several Péclet number in our study. The Péclet number is the product of the Reynolds number and of the Schmidt number. Given a vortical structure of size L, with a vorticity amplitude of φ, the Péclet number reads $\varphi \, L^2/\kappa$ and represents the ratio between the convection and diffusion processes; it can be interpreted as a ratio of eddy turn over time compared to the diffu-

sion time over the size of the vortex. In order to resolve cases, where the Péclet number is relatively large, adapting the grid for both hydrodynamic field and scalar field at the same time will lead to much too fine a mesh for the hydrodynamic solver in terms of memory trace and then computational cost for a given accuracy. The idea is then to use a dual grid and couple MR on this dual grid in order to save memory and time, while preserving error control.

7.3 Numerical strategy

We want to tackle problems where the flow and transported scalars have an inhomogeneous spatial distributions, with localized scalar fronts moving across the domain. We exploit this specificity by dynamically adapting the computational grid thanks to the multiresolution analysis [CKMP03, Pos01], in order to refine the mesh in regions where steep gradients occur and coarsen it elsewhere. By studying the physical characteristics of the problem at hand, we start with a uniform mesh refined enough to capture the smallest length scales. We then apply MR algorithms, at each timestep, to restrain the use of the finest meshes only where the variables present steep gradients, while employing coarser meshes elsewhere. Two attributes of the multiresolution analysis make it perfectly suited for adaptive grid refinement in numerical resolution of PDEs [Pos01]:

1. the adaptation process is based on local regularity indicators of the variable we are approximating. It is thus inherently more accurate than ad-hoc criteria used in AMR

2. we have the ability to monitor the error of the multiresolution process. This adaptation strategy comes with error-tracking capabilities allowing us to control the information loss that data compression necessarily entails

We do not need the same mesh resolution for the flow and the scalar transport; it suggests the use of a *multihierarchy* spatial discretization strategy [SRN10, Sch03, Sch04]. We will use one grid for the velocity and pressure fields, another one for the transported scalars, and will perform adaptive MR separately but in a consistent manner on *both* grids. This means that at any timestep we have a hybrid grid for the flow, that is hopefully coarser than the scalars' hybrid grid. Here we only consider the transport of a passive scalar, so we only need to project the velocity field on the scalars' grid when needed: this is done with the inter-level operations of the MR procedure (the prediction and projection operators, which are described in the next section). This will introduce some errors compared to a computation of both the flow and the scalar performed in a single grid, especially regarding the divergence-free constraint on the velocity, nevertheless this error is controlled thanks to the MR algorithm. We will denote

by (MMR) this new multihierarchy multiresolution adaptive strategy.
We discretize our PDEs in space via a finite volume method. The classical approach for finite volume schemes to approximate incompressible flows, is to use staggered grids for the velocity and pressure, to avoid spurious velocity and pressure modes [HW65]. However, the inter-level operations between embedded grids necessary for the MR algorithms on the one-hand, and the non-uniform character of the adaptive computational grid on the other hand, hinder the use of the staggered-grid layout in our case. We resort then to a *collocated* arrangement [CET06], and we have to deal with the spurious modes. Moreover, our adaptive grid is *non-conforming* ([CET06, CEH09, EGH10], see figure 7.1). This problem was also taken into account in the finite volume scheme that we designed to couple multiresolution to an incompressible flow solver. This scheme is easily usable to discretize the scalar transport, too.
The spatial discretization yields a Differential Algebraic Equation (DAE) for the velocity and pressure variables, and an Ordinary Differential Equation (ODE) for the scalar, since we only consider here the advection-diffusion of the scalar by the flow. Since the grid can change at each timestep, we prefer to integrate these equations in time by one-step methods. The integration of the DAE needs special care due to its stiffness [HW96, San13, AP98]; in addition the satisfaction of the divergence-free constraint cannot be achieved with an explicit method. We chose the fully implicit two stages Radau IIA method [HW96]. It is *stiffly accurate* [PR74], and does not suffer from order reduction when applied to DAEs [HW96]. It is 3^{rd} order for the velocity, and 2^{nd} order for the pressure. To integrate the scalar we use the classical explicit four stages Runge-Kutta method (RK4). We use the velocity at time n to advance the scalar at time $n+1$, and then advance the velocity.
In what follows, we will first give a short presentation of the adaptive multiresolution algorithms used in a finite volume context; more details can be found in [CKMP03, Pos01, TD11b]. We then describe briefly our spatial and temporal schemes.

7.3.1 Adaptive multiresolution strategy

We consider a variable u defined on a computational domain $\Omega =]0, b_x[\times]0, b_y[$, with $(b_x, b_y) \in \mathbb{R}_+^*$. We choose the maximum grid level $l^{\max} \in \mathbb{N}^*$ so that the computational mesh of size $2^{-l^{\max}} b_x \times 2^{-l^{\max}} b_y$ is fine enough to properly capture all the spatial scales of u in Ω. Let Ω_l be a set of nested dyadic Cartesian grids, indexed by their *refinement level* $l = 0, 1, 2, \ldots, l^{\max}$ so that for each l we have:

$$\Omega_l = \{\,]2^{-l} b_x i, 2^{-l} b_x (i+1)[\times]2^{-l} b_y j, 2^{-l} b_y (j+1)[\mid i, j \in \{0, 1, \ldots, 2^l - 1\} \,\}$$

We define:

$$K_{i,j}^l =]2^{-l}b_x i, 2^{-l}b_x(i+1)[\times]2^{-l}b_y j, 2^{-l}b_y(j+1)[$$
$$S_l = \{0, 1, \ldots, 2^l - 1\} \times \{0, 1, \ldots, 2^l - 1\}$$

The following then holds:

- Ω_l is the disjoint union of cells $K_{i,j}^l$, with $(i,j) \in S_l$ where S_l is the index set of the meshes of Ω_l
- $\overline{\Omega_l} = \overline{\bigsqcup_{(i,j)\in S_l} K_{(i,j)}^l} = \overline{\Omega}$
- if $l < l^{\max}$, for any cell $K_{i,j}^l \in \Omega_l$, there exists a unique set of 4 cells K_μ^{l+1} with $\mu \in S_{l+1}$ so that $\overline{K_{i,j}^l}$ is the union of the cells $\overline{K_\mu^{l+1}}$: the cells $K_{2i,2j}^{l+1}$, $K_{2i+1,2j}^{l+1}$, $K_{2i,2j+1}^{l+1}$ and $K_{2i+1,2j+1}^{l+1}$. We denote this set $\mathcal{C}_{i,j}^l$

There is a *natural* tree structure associated with such a set of embedded dyadic grids [CKMP03]. The root of the tree is the coarsest cell K^0, and for any cell K_γ^l with $l < l^{\max}$, we say that the cells $K_\mu^{l+1} \in \mathcal{C}_\gamma^l$ are the *children* of K_γ^l, and (reciprocally) that K_γ^l is the *parent* of the cells in $\mathcal{C}_\gamma^l$. *Leaves* of the tree are cells with no child. By definition, the initial set of leaves is formed by the cells at the most refined grid level $l^{\max}$. Here we have *quadtrees* in $2D$. Given $\varepsilon \in \mathbb{R}_+^*$, we build a multiresolution representation of u with the following steps:

1. Initialization
We start by computing a discrete representation $U_{l^{\max}} = (u_\gamma)_{\gamma \in S^{l^{\max}}}$ of u on $\Omega_{l^{\max}}$, where each u_γ is the average of u over the mesh $K_\gamma^{l^{\max}}$.

2. Projection
For $l \in \{l^{\max} - 1, l^{\max} - 2, \ldots, 1, 0\}$, we derive the approximation U_l on the grid Ω_l by *a projection* [TD11b] of the finer approximation on Ω_{l+1}. For each $\gamma \in S^l$, we have: $u_\gamma = \frac{1}{4}\sum_{\mu \in \mathcal{C}_\gamma^l} u_\mu$, *i.e.* u_γ is the average of the 4 values of the children meshes of K_γ^l.

3. Details computation
For each $K_{i,j}^l$, with $l \in \{0, 1, \ldots, l^{\max} - 1\}$ and $(i,j) \in S^l$, we derive a local regularity indicator of the variable u. For any vector of values $V = (v_k)$, where the k belongs to a finite set of indexes, let Q^s be a polynomial interpolation defined as:

$$Q^s(k, V) = \sum_{q=1}^{s} \xi_q (v_{k+q} - v_{k-q})$$

with $s \in \mathbb{N}$, and the ξ_q are the coefficients of centered linear polynomial interpolations of order $2s+1$ [BH96]. For each child $K^{l+1}_{2i+p,2j+q}$ (with $(p,q) \in \{0,1\}\times\{0,1\}$ depending on the child), we compute an approximate value (a.k.a. *a prediction*) $\hat{u}^{l+1}_{2i+p,2j+q}$ of $u^{l+1}_{2i+p,2j+q}$ by polynomial interpolation of the values on the grid Ω_l [TD11b]:

$$\hat{u}^{l+1}_{2i+p,2j+q} = u^l_{i,j} + (-1)^p Q^s(i, u^l_{.,j}) + (-1)^q Q^s(i, u^l_{i,.}) + (-1)^{(p+q)} Q^s_2(i,j;U^l) \tag{7.3}$$

where Q^s_2 reads:

$$Q^s_2(i,j;U^l) = \sum_{a=1}^{s} \xi_a \sum_{b=1}^{s} \xi_b (u^l_{i+a,j+b} - u^l_{i-a,j+b} - u^l_{i+a,j-b} + u^l_{i-a,j-b})$$

The local regularity indicator (a.k.a. *the detail*) is then defined as: $d^l_{i,j} = \sqrt{\sum_{\mu \in \mathcal{C}^l_{i,j}} (u^{l+1}_\mu - \hat{u}^{l+1}_\mu)^2}$

4. Thresholding
For each l from $l^{\max}-1$ down to 0, we associate a flag `keep-children` to every mesh $K^l_{i,j}$, that is initially set to `false`. Then if $\frac{d^l_{i,j}}{\max(d^l_{i,j})} \geq 2^{l-l^{\max}}\varepsilon$ we set the flag to `true`, otherwise we keep it to `false`. The maximum is taken over the set of all details of meshes belonging to the tree.

5. Grading
For each $K^l_{i,j}$, with $l \in \{0,1,\ldots,l^{\max}-1\}$ and $(i,j) \in S^l$, we denote by $R^l_{i,j}$ the indexes of the nodes needed for the computation 7.3. Then for l from $l^{\max}-1$ down to 0, if the flag `keep-children` of $K^l_{i,j}$ is set to `true`, for each cell K^l_γ with $\gamma \in R^l_{i,j}$, we set the `keep-children` of its parent to `true`.

6. Pruning
For each $K^l_{i,j}$, with $l \in \{0,1,\ldots,l^{\max}-1\}$ and $(i,j) \in S^l$, if its flag `keep-children` is set to `false`, then we *discard* its children from the tree structure. Let Λ be the set of indexes (l,γ), so that the cell K^l_γ belongs to the tree structure, let $\mathcal{M}$ be its set of leaves and $L(\Lambda)$ the indexes corresponding to these leaves . Λ and $\mathcal{M}$ evolve after the preceding pruning procedure, in such a way that the cells belonging to $\mathcal{M}$ are still a disjoint partition of Ω. If we denote: $\mathcal{M}_\Lambda U = (u^l_\gamma)_{(l,\gamma)\in L(\Lambda)}$, then $\mathcal{M}_\Lambda U$ is a *multiresolution approximation* of u, a new discrete and hybrid (because the leaves in $\mathcal{M}$ may not have the same size anymore) representation of this variable on the computational domain Ω.

The multiresolution analysis [Pos01] ensures that there exist a constant C in-

dependent of ε so that: $\|U_{l^{\max}} - \mathcal{M}_\Lambda U\| \leq C\varepsilon$, and so we have a control of the precision of our hybrid approximation with regard to the most refined uniform representation (see for example [Dua11]). We proceed with some remarks about this adaptation strategy:

1. If the interpolation stencil s in step 3 is so that $s \geq 1$, then the Grading step ensures that the level of two adjacent cells in $\mathcal{M}$ can differ by at most one unit (if K_γ^l and $K_\mu^{l'}$ are adjacent cells, then $l' \in \{l-1, l, l+1\}$). In this study, except stated otherwise, s will be set to 1 (figure 7.1 gives an example of a graded mesh discretization in this case)

2. We can combine steps 2 to 6 with a PDE numerical solver S to perform dynamic grid adaptation in the following way. Suppose that we start with an initial condition on u discretized over the most refined uniform grid. We apply the preceding adaptation strategy and obtain new sets Λ^0 and $\mathcal{M}^0$, and a multiresolution approximation $\mathcal{M}_\Lambda U^0$, that we will simply denote U^0. We then apply S to U^0 to obtain $\tilde{U^1}$ on $\mathcal{M}^0$, that we use to compute new projection values (step 2) and new details (step 3) for the nodes in Λ^0 that are not leaves. We modify step 4, and add another procedure: suppose that mesh K_γ^l is a leaf in Λ^0, that $l < l^{\max}$, and that d_μ^{l-1} is the detail of its parent cell (computed from $\tilde{U^1}$). If $\frac{d_\mu^{l-1}}{\max(d_\mu^{l-1})} \geq 2^{4s+4} 2^{l-1-l^{\max}} \varepsilon$, then we reconstruct the children of K_γ^l, we compute new values in these cells from K_γ^l and its neighbors, using the interpolation of step 3, and we set the `keep-children` flag of K_γ^l to `true`. From then we apply steps 5 and 6 to obtain new sets Λ^1 and $\mathcal{M}^1$, and a new vector U^1. We re-apply S and steps 2 to 6 to obtain U^2, and so on

3. We can adapt multiple variables $u_1, u_2, \cdots u_m$ on the same grid: we perform steps 1 to 4 for each variable separately, and we set the flag of a cell to `true` if it is set to `true` for at least one of the variables. We then perform steps 5 and 6, and the grid obtained will be accurate enough for all the variables. We can also discretize two variables u_1 and u_2 on two completely separate grids, but on the same domain Ω. If we need values from u_2 to perform a computation on the grid of u_1 for example, we can always use the projection and prediction operators (steps 2 and 3) to perform these values transfers between grids

7.3.2 Spatial discretization

$\mathcal{M}$ is the adaptive mesh that partitions the computational domain. For every rectangular mesh $K \in \mathcal{M}$, we denote by x_K the center of K, $m(K)$ its (Lebesgue) measure, $\mathcal{N}_K$ its set of neighbors, $\mathcal{E}_K = \{\sigma_{K|L} \mid \sigma_{K|L}$ the edge

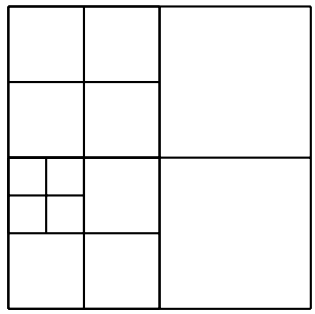


Figure 7.1: Example of a graded quadtree discretization

separating meshes K and L for $L \in \mathcal{N}_K\}$ its set of edge boundaries, $d_{K,\sigma}$ the Euclidian distance between x_K and σ, and $m(\sigma)$ the measure of σ, for $\sigma \in \mathcal{E}_K$. A discretized variable V on $\mathcal{M}$ is represented by its vector components $V = (v_K)_{K\in\mathcal{M}}$. We also denote l_K the level to which the mesh K belongs to in the grid hierarchy (cf. section 7.3.1).
The quantities that we are trying to approximate are the velocity $\boldsymbol{u} = (u_i(\boldsymbol{x}, t))_{i=1,2}$, the pressure $p(\boldsymbol{x}, t)$ and the transported scalar $v(\boldsymbol{x}, t)$. We start with the finite volume scheme to solve the PDE (7.1) that results in a Differential Algebraic Equation (D) for the quantities:

$$\boldsymbol{U} = (u_{i,K}(t))_{i=1,2,K\in\mathcal{M}}$$
$$P = (p_K(t))_{K\in\mathcal{M}}$$

The DAE (D) is found by approximating the differential spatial operators that appear in (7.1).
We follow [CEH09] in designing our finite volume scheme, and discretize (7.1) for every mesh K and component $i = 1, 2$ in the following way:

$$\begin{cases} m(K)\frac{du_{i,K}}{dt} + \underbrace{\nu \sum_{\sigma\in\mathcal{E}_K} F_{K,\sigma}(u_i)}_{\text{Diffusion}} + \underbrace{m(K)\partial_K^{(i)} P}_{\text{Pressure gradient}} + \underbrace{C_K^{(i)}(\boldsymbol{U})}_{\text{Convection}} = \int_K f_i(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = m(K)F_{i,K} \\ m(K)\,\mathrm{div}_K\,\boldsymbol{U} = \sum_{L\in\mathcal{N}_K} \Phi_{K|L}(\boldsymbol{U}) = 0 \end{cases} \tag{7.4}$$

$\nu \sum_{\sigma\in\mathcal{E}_K} F_{K,\sigma}(u_i)$ are approximations of the diffusive fluxes of the quantity u_i through the set of boundaries $\mathcal{E}_K$ of K. $\sum_{L\in\mathcal{N}_K} \Phi_{K|L}(\boldsymbol{U})$ are an approximation of the mass fluxes through the boundaries of K, with $\mathcal{N}_K$ its set of neighbors. Given the type of mesh we have to deal with (cf. section 7.3.1), we need to distinguish between 3 cases for the fluxes computation: the case where the meshes K and L are at the same level ($l_K = l_L$), the case where L has level

$l_L = l_K + 1$, and finally the case where L has level $l_L = l_K - 1$. The three cases are explicited in (figure 7.2).
The different equations (7.4) amount to a nonlinear system that can be written in matrix form (for example $\int_K \nabla \cdot \boldsymbol{u} \approx D \cdot \boldsymbol{U}$ where D is a *divergence* matrix). We formally define the *gradient* matrix as $-D^t$, that is, the discrete gradient is the *dual* operator of the discrete divergence [CET06, CEH09, GTG15]. This way, we make sure that our discretization mimics this property of the continuous PDE. In addition, we do not then have to specify boundary conditions for the pressure, which can be a tricky operation [GS87]. Finally, we define the convective term: $C_K^{(i)}(\boldsymbol{U}) = m(K)\,\mathrm{div}_K(\boldsymbol{U} \otimes \boldsymbol{U})^{(i)}$.
The scheme written here is complete for periodic boundary conditions. For Neumann or Dirichlet boundary conditions however, we need a special treatment for the discretization of the diffusion and mass fluxes of the velocity near the boundary of the domain. We will not get into the details of this implementation here, because the main goal of this article is not to describe the collocated spatial discretization. It is quite classical for collocated meshes in the literature, and will be presented in a subsequent paper in preparation.
We discretize (7.2) for every mesh K using the approximation fluxes of (7.4):

$$m(K)\frac{dS_K}{dt} + \underbrace{\kappa \sum_{\sigma \in \mathcal{E}_K} F_{K,\sigma}(s)}_{\text{Diffusion}} + \underbrace{m(K)\,\mathrm{div}_K(\boldsymbol{U}S)}_{\text{advection}} = \int_K \mathrm{f}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = m(K)\mathrm{F}_K \tag{7.5}$$

7.3.3 Temporal discretization

We now have to solve the following *Hessenberg index 2* DAE in the time parameter for the velocity and the pressure variables:

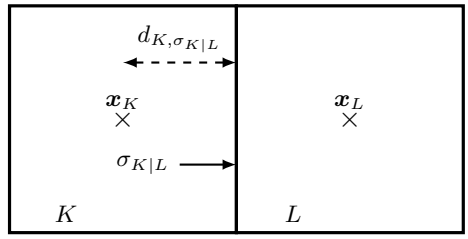
$$\begin{cases} \Gamma\dfrac{dU_i}{dt} = \nu L_i U_i + D_i^t P - \left(\displaystyle\sum_{j=1,2} D_j U_i U_j\right) + \Gamma S_i(t) \\ \displaystyle\sum_{i=1,2} D_i U_i = S_{\mathrm{div}}(t) \end{cases} \tag{7.6}$$

and the following ODE in the time parameter for the transported scalar:

$$\Gamma^{\mathrm{sc}}\frac{dS}{dt} = \kappa L^{\mathrm{sc}} S - \left(\sum_{j=1,2} D_j S U_j\right) + \Gamma^{\mathrm{sc}} S_{\mathrm{sc}}(t) \tag{7.7}$$

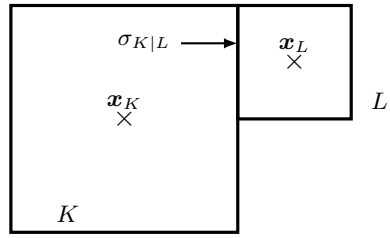
Here Γ, L_i and D_i are square matrices of size $\#(\mathcal{M}) \times \#(\mathcal{M})$. Γ is the mass matrix, the diagonal matrix so that $\Gamma_{i(K),i(K)} = m(K)$ for the mesh $K \in \mathcal{M}$. The L_i operators are the Laplacian matrices for the diffusive terms, and the D_i are the divergence matrices described above. The vectors $S_i(t)$ include

Case $l_L = l_K$



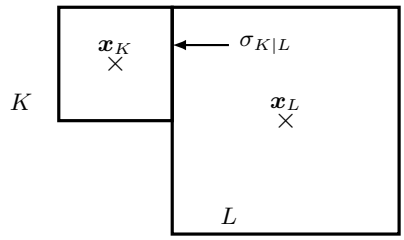
$$F_{k,\sigma_{K|L}}(u_i) = m(\sigma_{K|L})\frac{u_{i,K} - u_{i,L}}{d_{K,\sigma_{K|L}} + d_{L,\sigma_{K|L}}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{1,K} + u_{1,L}}{2}$$

Case $l_L = l_K + 1$



$$F_{k,\sigma_{K|L}}(u_i) = m(\sigma_{K|L})\frac{u_{i,K} - u_{i,L}}{\frac{d_{K,\sigma_{K|L}}}{2} + d_{L,\sigma_{K|L}}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{1,K} + u_{1,L}}{2}$$

Case $l_L = l_K - 1$



$$F_{k,\sigma_{K|L}}(u_i) = m(\sigma_{K|L})\frac{u_{i,K} - u_{i,L}}{d_{K,\sigma_{K|L}} + \frac{d_{L,\sigma_{K|L}}}{2}};$$
$$\Phi_{K|L}(\boldsymbol{U}) = m(\sigma_{K|L})\frac{u_{1,K} + u_{1,L}}{2}$$

Figure 7.2: Computation of the fluxes depending on the interface case

the discretized source terms F_i and the eventual boundary conditions, and the vector $S_{\text{div}}(t)$ include the boundary conditions for the divergence constraint. Since the flow and the scalar may not be solved on the same grid, we denote $\mathcal{M}^{\text{sc}}$ the grid for the scalar. Γ^{sc}, L^{sc} and D_i^{sc} are square matrices of size $\#(\mathcal{M}^{\text{sc}}) \times \#(\mathcal{M}^{\text{sc}})$. Γ^{sc} is the mass matrix, the diagonal matrix so that $\Gamma^{\text{sc}}_{i(K),i(K)} = m(K)$ for the mesh $K \in \mathcal{M}^{\text{sc}}$. The L^{sc} operator is the Laplacian matrix for the diffusive terms, and the D_i^{sc} are the divergence matrices described above. The vectors $S_{\text{sc}}(t)$ include the discretized source term F and the eventual boundary conditions.

We start with the resolution of (7.6) and we solve it with the Runge-Kutta **two-stage Radau IIA** method, which is 3^{rd} order for the velocity and 2^{nd} order for the pressure [HW96, San13]. We derive it here with the ε-embedding method [HW96] in the following way. Given velocities and pressure fields $(\boldsymbol{U^0}, P^0)$ at time t_0, we want to obtain an approximate solution of (7.6) $(\boldsymbol{U^1}, P^1)$, at time $t_0 + h = t_1$ with an implicit 2-stage Runge-Kutta method. The ε-embedding method recasts equation (7.6) in the following form:

$$
\begin{aligned}
\Gamma g_i^1 = \;& \Gamma U_i^0 + h a_{11}\big(\nu L_i g_i^1 + D_i^t\, k^1 - (\sum_{j=1,2} D_j g_i^1 g_j^1) + \Gamma S_i(t_0 + c_1\, h)\big) \\
& + h a_{12}\big(\nu L_i g_i^2 + D_i^t\, k^2 - (\sum_{j=1,2} D_j g_i^2 g_j^2) + \Gamma S_i(t_0 + c_2\, h)\big) \\
\Gamma g_i^2 = \;& \Gamma U_i^0 + h a_{21}\big(\nu L_i g_i^1 + D_i^t\, k^1 - (\sum_{j=1,2} D_j g_i^1 g_j^1) + \Gamma S_i(t_0 + c_1\, h)\big) \\
& + h a_{22}\big(\nu L_i g_i^2 + D_i^t\, k^2 - (\sum_{j=1,2} D_j g_i^2 g_j^2) + \Gamma S_i(t_0 + c_2\, h)\big) \\
\sum_{i=1,2} D_i g_i^1 = \;& S_{\text{div}}(t_0 + c_1 h) \\
\sum_{i=1,2} D_i g_i^2 = \;& S_{\text{div}}(t_0 + c_2 h) \\
\Gamma U_i^1 = \;& \Gamma U_i^0 + h b_1\big(\nu L_i g_i^1 + D_i^t\, k^1 - (\sum_{j=1,2} D_j g_i^1 g_j^1) + \Gamma S_i(t_0 + c_1\, h)\big) \\
& + h b_2\big(\nu L_i g_i^2 + D_i^t\, k^2 - (\sum_{j=1,2} D_j g_i^2 g_j^2) + \Gamma S_i(t_0 + c_2\, h)\big) \\
P^1 = \;& \big(1 - h \sum_{i,j=1}^{2} b_i \omega_{ij}\big) P^0 + \sum_{i,j\ =1}^{2} b_i \omega_{ij} k^j
\end{aligned}
\tag{7.8}
$$

h is the timestep, g_i^j, k^j are intermediate variables, and $W = (\omega_{ij})_{1 \le i,j \le 2}$ is the inverse of the matrix $A = (a_{ij})_{1 \le i,j \le 2}$ corresponding to the Radau IIA method (see table 7.1). Since this method is stiffly accurate, we actually have $U_i^1 = g_i^2$; and since it is L-stable, we actually have $(1 - h \sum_{i,j=1}^{2} b_i \omega_{ij}) = 0$ [HW96], which simplify the computations. The presence of the convective terms in (7.1) implies that (7.8) is a nonlinear system in the variables (g_i^j, k^j), that has been

solved here with simple fixed-point Picard iterations [Tur98].

We now treat the spurious pressure and velocity modes. The spurious modes are due to the linear part of equation (7.1) [GS00], so that we only have to consider the Stokes equation for their treatment. If we do not take into account the convective terms in (7.8), the velocity and pressure at time t^{n+1} are obtained by inverting the matrix $E = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix}$, where:

- B is block-diagonal matrix consisting of d (the space dimension) diagonal blocks B_i, with $B_i = \begin{pmatrix} \Gamma - a_{11}h\nu L_i & -a_{12}h\nu L_i \\ -a_{21}h\nu L_i & \Gamma - a_{22}h\nu L_i \end{pmatrix}$
- $M = \begin{pmatrix} M_1 \\ M_2 \end{pmatrix}$, with $M_i = \begin{pmatrix} -a_{11}hD_i^t & -a_{12}hD_i^t \\ -a_{21}hD_i^t & -a_{22}hD_i^t \end{pmatrix}$
- $N = \begin{pmatrix} N_1 & N_2 \end{pmatrix}$, with $N_i = \begin{pmatrix} D_i & 0 \\ 0 & D_i \end{pmatrix}$

The spurious modes that will affect the solution of our problem are the vectors of the form $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ which are in the null-space of E: $E \begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = \begin{pmatrix} B & M \\ N & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix} = 0$. If $\boldsymbol{v} \neq 0$, it will pollute the quantities g_i^1, g_i^2 in (7.8), causing the apparition of spurious modes in the computed solution; and of course the same is true with q regarding k^1, k^2. But as we wrote earlier, we only need the velocity for the scalar, not the pressure, so we are just going to ensure that there is no spurious mode for the velocity. There might be spurious mode in the pressure, but if we make sure that they do not affect the velocity, we are ensured of the precision of our velocity computation.
We remark here that by construction, $-\nu L_i$ is a symmetric and diagonally dominant matrix, so that it is diagonalizable (in $\mathbb{R}$) and all its eigenvalues are positive. The matrix B depends on h the timestep, and we use the following result:

Theorem 7.3.1. *There exists $h_0 > 0$ so that for each $0 < h \leq h_0$, if $\begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}$ is a null vector of the matrix E, then $\boldsymbol{v} = 0$.*

$\frac{1}{3}$	$\frac{5}{12}$	$\frac{-1}{12}$
1	$\frac{3}{4}$	$\frac{1}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$

Table 7.1: Butcher array of the Radau IIA scheme

We will use the following lemma:

Lemma 7.3.1. *If a matrix B is as defined above, then there exists $h_0 > 0$ so that for each $0 < h \leq h_0$ and for any real vector $\boldsymbol{z}$ the two following propositions are equivalent:*

1. $\boldsymbol{z}^t B \boldsymbol{z} = 0$
2. $\boldsymbol{z} = 0$

Proof. (2) $\implies$ (1) is obvious for any $h \in \mathbb{R}_+^*$. We then turn to (1) $\implies$ (2). For $h \to 0$, B converges to a diagonal matrix where each block is the mass matrix. Hence, in the limit, all its eigenvalues are positive. By the continuous dependency of the eigenvalues on the matrix coefficients, there is some $h_0 > 0$ such that all eigenvalues of B, for $h \leq h_0$, are positive too. In that case $\boldsymbol{z}^t B \boldsymbol{z} = 0$ implies that $\boldsymbol{z} = 0$, which concludes the proof of the lemma.

□

Proof. We have that $B\boldsymbol{v} + Mq = 0$. If we multiply this vector by $\boldsymbol{v}^t$, we have $\boldsymbol{v}^t B \boldsymbol{v} + \boldsymbol{v}^t M q = 0$. Or $\boldsymbol{v}^t M q = q^t (M^t \boldsymbol{v})$, and if we write $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}^1 \\ \boldsymbol{v}^2 \end{pmatrix}$, we have that $M^t \boldsymbol{v} = \begin{pmatrix} -a_{11} h(\sum_{i=1}^2 D_i \boldsymbol{v}^i) - a_{21} h(\sum_{i=1}^2 D_i \boldsymbol{v}^i) \\ -a_{12} h(\sum_{i=1}^2 D_i \boldsymbol{v}^i) - a_{22} h(\sum_{i=1}^2 D_i \boldsymbol{v}^i) \end{pmatrix} = 0$, because $N\boldsymbol{v} = 0$. Hence $\boldsymbol{v}^t B \boldsymbol{v} = 0$.
We see that if we choose a real h_0 that satisfies the conditions of the lemma, then necessarily, $\boldsymbol{v} = 0$, which concludes the proof of the theorem.

□

We see that we can always choose h small enough to preclude the apparition of spurious modes for the velocity field, the one that is detrimental for the transport of scalars.
The resolution of the ODE (7.7) is done with the RK4 method, which is 4^{th} order for the scalar. We discretize the time interval $[0, T]$ with fixed timesteps of size h, the timestep used to solve the flow. RK4 being an explicit method, we have to take into account stability restrictions when solving the scalar equation, that do not apply to the Radau IIA method. In practice we thus integrate (7.7) with a timestep h^{sc} smaller than h. We use the velocity at time nh to integrate the scalar from nh to $(n+1)h$ with $\lfloor \frac{h}{h^{\text{sc}}} \rfloor$ iterations of RK4, and then use one iteration of (7.8) to advance the flow variables from nh to $(n+1)h$.

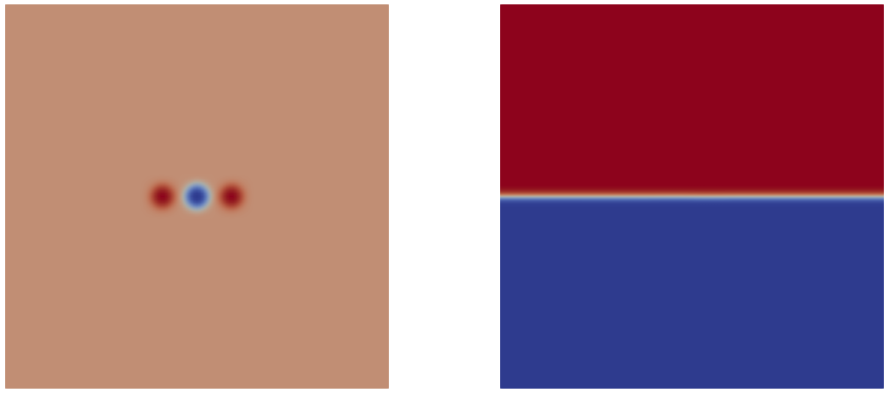


Figure 7.3: Initial values of the numerical experiment. Left: vorticity field; right: passive scalar

7.4 Numerical experiments

7.4.1 Initial configuration

The initial configuration of the flow consists of three vortices $(\omega_1, \omega_2, \omega_3)$ distributed on the horizontal axis with their centers located respectively at the center of the domain $(0, 0)$, and at $(-x_0, 0)$ and $(x_0, 0)$. They have the following shapes:

$$\omega_1 = -\frac{1}{2}\varphi\left(1 + \tanh\left(\frac{1}{\delta_0}(r_0 - \sqrt{x^2 + y^2})\right)\right)$$
$$\omega_2 = +\frac{1}{2}\frac{\varphi}{2}\left(1 + \tanh\left(\frac{1}{\delta_0}(r_0 - \sqrt{(x - x_0)^2 + y^2})\right)\right)$$
$$\omega_3 = +\frac{1}{2}\frac{\varphi}{2}\left(1 + \tanh\left(\frac{1}{\delta_0}(r_0 - \sqrt{(x + x_0)^2 + y^2})\right)\right)$$

The initial condition for the scalar is given by

$$s = \tanh\left(\frac{1}{\delta_1}y\right)$$

It aims at mimicking an initial configuration where the scalar is a constant value in the upper half of the domain, and another one in the lower half of the domain (we can think for example of a gas mixture with fresh fuel in the upper half, and hot air in the lower half). The source terms of (7.1) and (7.2) are set to 0. Figure (7.3) shows the initial vorticity and scalar field.

The computational domain Ω is the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$, the boundary conditions for the velocity are free-flow boundary conditions, and we impose homogeneous Neumann boundary conditions for the transported scalar. We set $\nu = 5.10^{-4}$, $\varphi = 100$, $\delta_0 = 0.01$, $r_0 = 0.03$, $x_0 = 0.09$, and $\delta_1 = 0.01$.
We turn to an analysis of the relevant spatial scales of this configuration. The characteristic scales of the flow are the core size of each vortex $2r_0$ and the distance between the vortices center x_0. The characteristic scale for the scalar is the thickness of the mixing layer δ_1. It can happen that δ_1 is an order of magnitude smaller than x_0 and $2r_0$, which means that the scalar needs a finer mesh than the flow to be accurately solved numerically. The dimensionless parameter of interest here is the Peclet number, $Pe = \dfrac{4\varphi r_0^2}{\kappa}$, which represents the ratio between advective and diffusive processes in the scalar transport [GS00]. If Pe is small then the diffusive forces are prevalent, and they tend to quickly thicken δ_1. In this case a grid resolving the velocity may be good enough to solve the scalar transport. But if Pe is large, then the advective forces are prevalent, and the initial small mixing layer is advected in a spiral way by the flow, almost without thickening due to diffusion. In that case the grid must capture the small initial structures of the scalar.
We rewrite the Peclet number $Pe = ReSc$, where $Re = \frac{4\varphi r_0^2}{\nu}$ is the Reynolds number of the flow, and $Sc = \frac{\nu}{\kappa}$ is the Schmidt number. We fix $Sc = 0.1$ to investigate the error control capability of our method, and we will later consider larger Schmidt numbers. With such parameters, a uniform grid with $l^{\max} = 7$ is fine enough to properly describe the flow, but in some situation this might be too coarse for the scalar. In each of the following computations, we fix $h = 2.5 \times 10^{-3}$ for the timestep for the flow, and $h^{\text{sc}} = 1.25 \times 10^{-4}$ for the scalar. We verified that, for each of the following computations, these timesteps were small enough to ensure convergence in time; also, the use of high order temporal integrators ensures the fact that the temporal errors are negligible compared to the spatial errors. We will denote by (MR) a classical multiresolution adaptive strategy where the flow and the scalar are computed on the same grid. Thus the initial uniform grid has to be refined enough to capture the spatial scales of both the scalar and the flow, and the adaptive process produces a grid adapted for both of them. We recall that we denote by (MMR) the new multihierarchy multiresolution adapative strategy that we developed here, with separate grids for the flow and the scalar. For the (MMR) strategy, we denote $l_v^{\max}$ the maximum level for the flow grid, and $l_{\text{sc}}^{\max}$ the maximum level for the scalar.

7.4.2 Numerical results

First we compare the result of our numerical strategy with a computation run on a uniform grid. For the (MMR) computation, we set $l_v^{\max} = 7$, $l_{\text{sc}}^{\max} = 8$ for the scalar, we choose $\varepsilon = 10^{-3}$ for thresholding parameter, and run the computation from 0 to 0.5. The uniform grid is set at $l^{\max} = 8$ for both the flow and the scalar. The results for three times, $t = 0.0075$, $t = 0.025$ and $t = 0.5$, are shown in figure (7.4), where for each time we have the adapted grid for the scalar on the left, the scalar value for the adapted grid on the center, and the scalar value for the uniform grid computation on the right. We see here a good agreement suggesting that our adaptive strategy is able to produce the same results as on a uniform fine grid. We also show the evolution of the vorticity field in (7.5).

Next we check the ability to control the multiresolution error with our new adaptive strategy. We recall that we adapt both the flow and the scalar, but on different grids. Here, we set $l_v^{\max} = 7$, and $l_{\text{sc}}^{\max} = 7$ or $l_{\text{sc}}^{\max} = 8$. In each case, we run the simulation until $t = 0.5$, and at this time we compute the L^2-norm of the error between the adapted scalar value, and the value obtained with the same two-grids algorithm, but with both the flow and the scalar computed on their respective uniform most refined grid. We do this for different values of ε. Figure (7.6) shows that we still preserve the ability to control the adaptation error with ε, the error decreasing linearly with the threshold parameter.

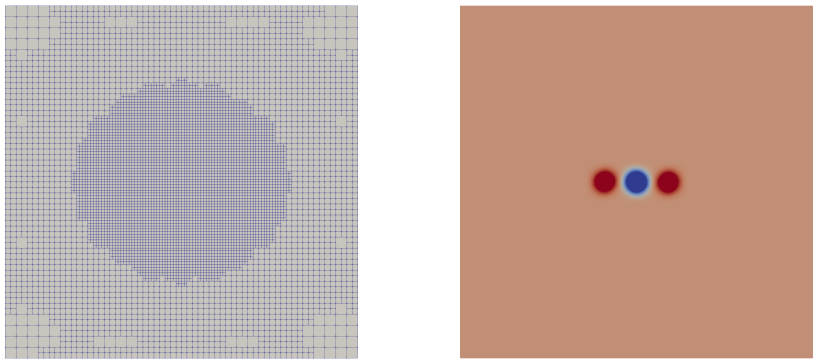
Our last numerical test deals with variable Péclet numbers. As we stated earlier, the flow and the scalar might need different resolutions to be adequately solved, and in our case this is mainly related to the relative importance of the advection versus the scalar diffusion. We set $\varepsilon = 10^{-3}$, and we run the computation until $t = 0.5$, with different Schmidt numbers. We study the values for $Sc = 0.1$, $Sc = 1$ and $Sc = 10$. For each Schmidt number, we run three computations: one with a uniform grid at $l^{\max} = 7$ for the flow and $l^{\max} = 10$ for the scalar (our reference for comparisons), one with (MR) where $l^{\max} = 7$, and one (MMR) with $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 9$. The results are shown in figure (7.7), where we zoom in closer to the mixing layer. We see that at $Sc = 0.1$ and $Sc = 1$, the (MR) and (MMR) are in good agreement with the reference simulation. But for $Sc = 10$, the (MR) solution is not accurate, and we actually have to resort to a grid with $l_{\text{sc}}^{\max} = 9$ for the scalar to properly describe the phenomenon. The computation would take much longer if we had to resort to a unique grid, even if we use the adaptive multiresolution for the flow and the scalar: in our case the velocity grid has around 7000 cells, while the scalar grid as around 63000 cells, so this would mean solving a linear system for the flow almost ten times higher, even though the physics of the flow does not require such accuracy. This case clearly advocates the usefulness of our method.

(a) $t = 0.0075$

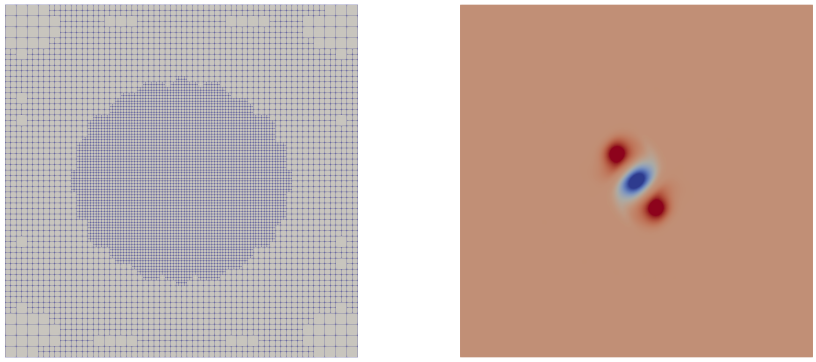
(b) $t = 0.25$

(c) $t = 0.5$

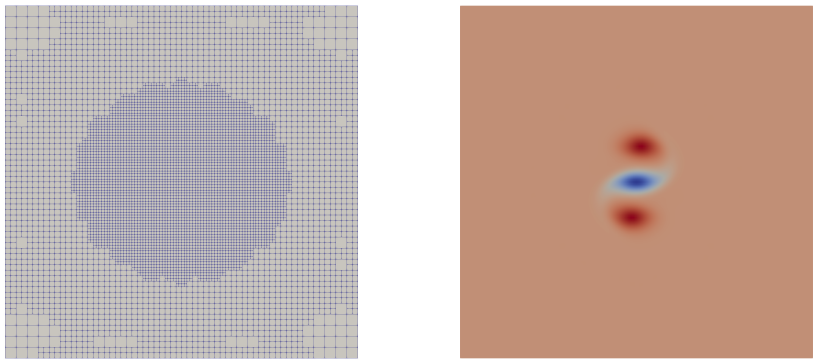
Figure 7.4: Interaction of a passive scalar and a vortex field. Comparison between a computation on a uniform grid at $l^{\max} = 8$, and a computation done with (MMR), where $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 8$. Left: adapted grid for the scalar; center: scalar value on the adapted grid; right: scalar value on the uniform grid



(a) $t = 0.0075$



(b) $t = 0.25$



(c) $t = 0.5$

Figure 7.5: Evolution of the vorticity field with grid adaptation by multiresolution, where $l_v^{\max} = 7$. Left: adapted grid; right: vorticity field

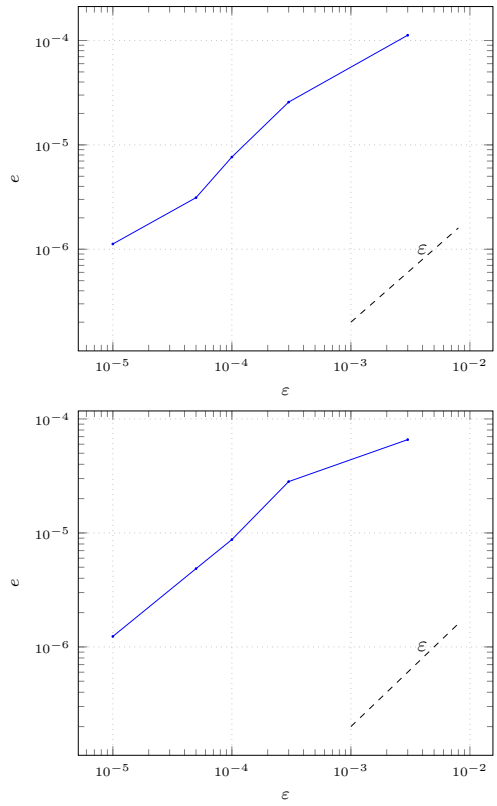


Figure 7.6: L^2 norm of the multiresolution error versus the threshold parameter ε for the scalar. Left: $l_{\text{sc}}^{\max} = 7$; right: $l_{\text{sc}}^{\max} = 8$

ε	3.10^{-2}	3.10^{-3}	3.10^{-4}
Speedup	$\times 8.3$	$\times 8.04$	$\times 8.13$

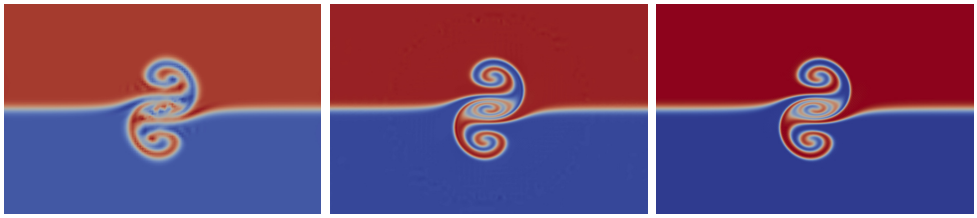
Table 7.2: Interaction of a passive scalar and a vortex field. Comparison between the (MR) and (MMR) computing times for different thresholding parameters. (MR) with $l^{\max} = 8$; (MMR) with $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 8$. For a given value ε of the thresholding parameter, the adapted grid computation with (MR) is *speedup* times longer than the computation with (MMR)

ε	3.10^{-2}	3.10^{-3}	3.10^{-4}
Speedup	$\times 71.32$	$\times 38.40$	$\times 16.5$

Table 7.3: Interaction of a passive scalar and a vortex field. Comparison between the uniform grid and (MMR) computing times for different thresholding parameters. Uniform grid with $l^{\max} = 8$; (MMR) with $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 8$. For a given value ε of the thresholding parameter, the uniform grid computation is *speedup* times longer than the computation with (MMR)

(a) $Sc = 0.1$

(b) $Sc = 1$



(c) $Sc = 10$

Figure 7.7: Interaction of a passive scalar and a vortex field. Comparison between uniform, (MR) and (MMR) solutions at $t = 0.5$ for different Schmidt numbers. From left to right: (MR) with $l^{\max} = 7$; (MMR) with $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 9$; uniform with $l_v^{\max} = 7$ and $l_{\text{sc}}^{\max} = 10$, reference

7.4.3 Cost comparison

Finally, we want to discuss the computational advantages of the (MMR) method. Before doing so, we mention that the computing costs are determined in huge part by the software implementation. The code used here is a research/development software, which primary purpose was not to obtain high-performance computing, but rather to showcase the ability of the (MMR) method to obtain correct and precise simulation results. The three computational settings, the uniform grid, the (MR) and the (MMR) all share the same routines for the linear space operations (matrix–vector multiplication, vector–vector addition, vector–vector inner product, linear solver): we use the optimized PETSc libraries [BAA+19b, BAA+19a, BGMS97] which are written in C. But the data structures and routines needed for the adaptive multiresolution implementation in the (MR) and (MMR) simulations are written in Python, for sequential computations only.

We will proceed to two different comparisons. The first one will help us to determine the computational gain between the (MR) and the (MMR) methods. While both methods use grid adaptation by multiresolution, the single grid algorithm has to be set at the refinement requirement of the scalar transport, which is not the case for the two grids algorithm. Clearly, the (MMR) method will generate much less meshes than the (MR) one for the flow, and we want to quantify the resulting speedup.

We set $Sc = 0.1$, for both the (MR) and (MMR) methods we set $l_{\text{sc}}^{\max} = 8$, and for the (MMR) method we also fix $l_v^{\max} = 7$. We run the computation from 0 to 0.5, for various values of the thresholding parameters. We save the computing time for the (MR) and (MMR) computations, and we report the ratio between the (MR) computing time and the (MMR) computing time in table (7.2). The (MMR) method is 8 times faster than the (MR) method here, which is a very good speedup given the precision that we are still able to obtain.

The second comparison concerns the (MMR) method and the uniform grid computation. These kinds of comparisons are tricky, because we are dealing with two different softwares. A common mistake is to use the code for the adapted grid to do a computation on a uniform grid; that is not the case here, because we built a specific code for the uniform grid computation, that does not suffer at all from the overhead of the multiresolution. What is more, this code runs entirely on PETSc, so that is fairly optimized compared to the (MMR) code that relies heavily on Python. Nevertheless, we can take full advantage of working on a coarser grid for the flow, and we quantify the resulting speedup.

We set $Sc = 0.1$, and for the (MMR) method we set $l_{\text{sc}}^{\max} = 8$, and $l_v^{\max} = 7$. For the uniform grid computation we set $l^{\max} = 8$, and we run the computation from 0 to 0.5, for various values of the thresholding parameters. We save the computing time for the uniform grid algorithm once, the different computing times for the (MMR) computations, and we report the ratio between the uni-

form grid computing time and the (MMR) computing time in table (7.3). The speedup obtained with the (MMR) method, even with a threshold parameter as small as 3.10^{-4}, is already impressive, and we expect it to be even greater with an optimized implementation of the adaptive multiresolution.
We finish this section by recalling that these results were obtained with an in-house code, on relatively small test cases. But we expect them to be even better for the (MMR) on a more optimized code for two reasons: (i) the multiresolution algorithms are linear in time, while the matrix inversion necessary in incompressible flow computations are polynomial in time, with a degree greater than 2 generally, thus the overhead of the multiresolution algorithms is negligible when compared to the grid size reduction, and (ii) with the (MMR) we can be at an even higher level of refinement for the scalar.

7.5 Conclusion

We introduce a new spatial adaptive strategy to efficiently solve the transport of a passive scalar by an incompressible flow. The flow and the scalar are discretized in a finite-volume context on different grids, and multiresolution adaptive refinement techniques are applied to both variables. Regarding the incompressible Navier-Stokes equations, the velocity and pressure unknowns are discretized on a collocated setting, and the DAE resulting from the spatial discretization is solved with an implicit high-order Runge-Kutta method, allowing third-order accuracy in time for the velocity. The ODE resulting from the spatial discretization of the scalar is solved with an explicit fourth order Runge-Kutta method. This new strategy is particularly suited for configurations where the characteristic scales of the scalar are much smaller than the characteristic scales of the flow, as is often the case in chemically reacting flows. We can achieve an accurate description for the scalar, without paying the price of solving the flow on a grid with a much larger number of unknowns that would be necessary for the description of the latter. We review this on a manufactured case of 3 vortices interacting with a passive scalar mixing layer. The next step will consist in combining these techniques with high-order operator splitting techniques for an efficient and accurate resolution of chemically reacting flows.

Acknowledgments

This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

— Conclusion Perspectives —

General Conclusion and Prospects

This work takes place in the context of the development of new numerical strategies to efficiently solve stiff PDEs originating from the modelling of multiscale reactive flows. We build a new numerical scheme to solve the incompressible Navier-Stokes equations in the primitive variables, with grid adaptation based upon a fully adaptive finite-volume multiresolution scheme, and high-order temporal integration methods. The spatial discretization is performed on collocated grids, in order to implement the multiresolution algorithms. We demonstrate that we are able to control the errors due to the mesh adaptation technique, an important issue that is often disregarded or underestimated in similar numerical strategies. Regarding the temporal integration method, special care is given to the theoretical properties of the proposed scheme, namely the stability of the method, and its ability to deal properly with the stiffness coming from the algebraic constraints in the semi-discretized incompressible Navier-Stokes equations. This numerical strategy is implemented from scratch in an in-house numerical code, with modularity features enabling the comparisons of various integration techniques. We assess the capabilities of this new hydrodynamic solver in terms of speed and efficiency, in the context of scalar transport on adaptive grids. The numerical simulation of incompressible flows is a very difficult task, and the achievements presented in this thesis rest upon complex and advanced notions in functional analysis, numerical analysis, linear algebra, algorithms and data structures. We believe that the numerical techniques developed in this study pave the way to a new generation of time/space adaptive numerical methods for the simulations of multicomponent reactive flows exhibiting severe stiffness. In the Introduction, we mentioned that the present work aims at complementing the strategy introduced by M. Duarte in [Dua11], to obain a low-Mach solver with adaptation in time and space with error control. We believe that we have completed the first part of this task, with the design of a high-order hydrodynamics solver for the incompressible Navier-Stokes equations with grid adaptation by multiresolution.

In what follows we summarise the main contributions of this work, that have extensively been showcased throughout this dissertation:

- In terms of mathematical analysis:
  - we developed a new general framework to build high order Implicit-Explicit Additive Runge-Kutta methods to solve the incompressible Navier-Stokes equations. The main advantage of this new generation of additive schemes is that the stiff and linear part of the equations are treated with a dedicated implicit scheme, whereas the nonlinear convective part of the equations is treated with an explicit scheme, enabling important computational time savings.
- In terms of numerical schemes:
  - a new finite-volume scheme for the incompressible Navier-Stokes equations on adaptive rectangular grids, in a collocated layout. This scheme showcases an original treatment of the spurious pressure and velocity modes, that does not alter the precision of the discretization technique. Although this scheme was designed here in the context of adaptive multiresolution, it can be applied to a more general context, as long as the computational grids share the features described in Chapter 2;
  - an original strategy to perform adaptive multiresolution coupled to the resolution of the incompressible Navier-Stokes equations in a finite-volume context. To our knowledge, this constitutes the first derivation of fully adaptive finite-volume multiresolution schemes for incompressible flows;
  - a new 3^{rd}-order additive Runge-Kutta method for the incompressible Navier-Stokes equations, combining a 3^{rd}-order, A-stable, stiffly accurate, 4-stage ESDIRK method for the algebraic linear part of these equations, and a 4^{th}-order explicit Runge-Kutta scheme for the nonlinear convective part. This scheme enforces the mass conservation to round-off errors, *i.e.* the velocity vector is divergence-free to machine accuracy at each timestep.
- In terms of scientific computing:
  - a new software implementation of the algorithmic procedures of adaptive multiresolution. It is a generic code to perform multiresolution analysis in $1D/2D/3D$, in a finite-volume context. The code considers the hashtable data-structures for the multiresolution representation, and it comes with a new set of algorithms to perform the multiresolution algorithms without recursive tree navigation;

- the software implementation of relatively efficient preconditioned Uzawa algorithms, to solve the Stokes matrices that appear in the application of high-order implicit Runge-Kutta methods to solve the incompressible Navier-Stokes equations. This is a tailored implementation, using both direct and iterative inner solvers for specific submatrices;
- A new academic, generic code developed for the numerical solution of the incompressible Navier-Stokes equations with space adaptive, and high-order temporal integration schemes. It is written in Python, for modularity purposes, and relies heavily on the PETSc library for most of the computationally intensive operations.

We conclude by mentioning some further developments:

- In terms of mathematical analysis:
  - The construction of discrete spatial operators with better properties for the adaptive non-uniform grids. In particular, we need to introduce a stabilizing technique à la Brezzi-Pitkäranta [BP84] in order to preclude the apparition of spurious pressure modes. This should allow to obtain convergence properties of the spatial discretization scheme on non-conforming grids, a piece that is missing in our work;
  - an analysis of the convergence speed of the spatial discretization scheme is also missing. We suspect that in practice the order of convergence is between 1 and 2 when the mesh size tends to zero. We believe that the ideas that we exposed in this work can be a good starting point to build collocated finite-volume schemes with a higher convergence order, and good convergence properties;
  - more work is needed to exhibit the order conditions of additive Runge-Kutta methods applied to Hessenberg index 2 DAEs in general, and to the incompressible Navier-Stokes equations in particular. This would help to build a more precise framework for the development of high-order IMEX Runge-Kutta methods for the simulation of incompressible flows.
- In terms of numerical schemes:
  - It would be interesting to build a more specific preconditioner for the Uzawa algorithm that we implement to solve the Stokes problem that appears in our additive Runge-Kutta scheme. It is well known that in the simulation of incompressible flows, the most time consuming task is the resolution of the linear systems. A first improvement regarding our methodology would be to use multigrid techniques to solve the Poisson equation for the pressure;

– the new high-order adaptive solver for the incompressible Navier-Stokes equations that we introduce here can be extended in a straightforward manner to implement a low-Mach solver. The mass conservation constraint needs to be adapted, as well as the computation of the diffusive terms in the momentum equations. We are currently working on such an extension by developing a space adaptive low-Mach solver for the simulation of laminar buoyant jets;

Finally, we come back to the second building block that was missing in the work of M. Duarte [Dua11]: an efficient computational implementation of the adaptive multiresolution algorithms. Indeed, the data structure that we use to perform the multiresolution analysis is not the best suited to take full advantage of modern processors architectures. The construction of a tree in the form of a quadtree (2D) or octree (3D) is probably the simplest way to represent the multiresolution mesh. Nevertheless, if we look a little closer at the operators used (prediction, projection, ...), we can observe that we very often need to know the neighbors of a given cell. However, the search of nodes has a significant cost if we use a naive tree data structure based on pointers as originally done in M. Duarte's work [TD11a]. Such a data structure was essential in order to obtain a proof of concept but is not adapted to an optimized and efficient implementation of the multiresolution approach. Most of the software using multiresolution methods or cell-based AMR methods, which are very close to multiresolution methods, use a tree data structure. But, to improve the locality and thus speed-up the search for neighbors, a space filling curve (Morton curve) [BWG11] is used in this work. This technique improves memory access compared to tree data structure based on pointers and is well suited for parallel use [BWG11]. Nevertheless, the data structure based on space filling curve is not efficient for finding the neighbors during the multiresolution process. Indeed, to search a neighbor cell, tests are necessary to know if the cell exists at the same level l. If not, one can search neighbor cell at the coarsen level $l-1$ or at the finer level $l+1$. Thus, even if the data structure based on Morton curve improves the locality, a lot of tests have to be performed to find the existing neighbor cell and the recursive nature the prediction step is a stumbling block in the search of locality. A first attempt to overcome these difficulties was proposed in [DDD+15] in collaboration with INTEL relying on the TBB library on Xeon Phi, that is on shared-memory architecture. It still relied on a space-filling curve and Morton index ordering.
During this work, the need for a new paradigm for the data structure has emerged through a collaboration with Loïc Gouarin, Laurent Séries and Hugo Leclerc [1] [LNS+18]. A new data structure, better suited to multiresolution methods and not based on a tree approach nor space-filling curves, has been

[1] L.G. at CMAP, L.S. at CMAP and H. Leclerc 4 months at CMAP before moving to LMO.

created. Instead of storing the coordinates of each cell independently (e.g. as in Morton ordering based codes), the idea is then to store contiguous ranges along a given direction (x by default). These ranges are then sorted first by refinement level (allowing traversals one refinement level at a time) and then to coordinates, ending with the "main" one (which is x by default). This data structure allows keeping contiguous access in the x direction, which is a good point for most of the operators used to solve PDE based on stencil operators. It is also interesting for the prediction and projection operators described in this work. Since ranges are used for each space direction, finding the neighbors of a cell using the algebra of sets (intersection, union, difference, *etc.*) becomes much easier. For example, to make the projection from level $l+1$ to l, the intersection between cells of level $l+1$ and l is performed. If the set is not empty, then we can apply the projection operator. The algorithms of the algebra of sets construct new sets of ranges. There is no need to test if a cell at a given level exists which improves considerably the execution time [LNS+18]. Our primary tests show a speed-up of 5 compared to data structures based on Morton index and z-curve. This speed-up is mainly related to mesh handling and not to PDE resolution. Several developments are in progress in order to identify the precise impact on global time-to-solution on a series of representative test cases ranging from hyperbolic PDEs to parabolic reaction fronts of reaction-diffusion type. It seems that the new approach leads to better performance and nice parallelization features. This software, called `MuRe`, should be available by the end of 2020 with an open source license and we aim at implementing the proposed approach for the incompressible Navier-Stokes equations in it, as well as, eventually, the final low-Mach solver.

References

[ABB+10] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, and M. Zingale. CASTRO: a new compressible astrophysicial solver. I. Hydrodynamics and self-gravity. *The Astrophysical Journal*, 715(2):1221–1238, 2010.

[ABC+98] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142(1):1 – 46, 1998.

[Abd02] A. Abdulle. Fourth order Chebyshev methods with recurrence relation. *SIAM Journal on Scientific Computing*, 23(6):2041–2054, 2002.

[ABS96] A. S. Almgren, J. B. Bell, and W. G. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM Journal on Scientific Computing*, 17(2):358–369, 1996.

[ADB15] A.J. Aspden, M.S. Day, and J.B. Bell. Turbulence-chemistry interaction in lean premixed hydrogen combustion. *Proceedings of the Combustion Institute*, 35(2):1321 – 1329, 2015.

[ADKL01] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[AGLP06] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[AMSS97] A. L. Araujo, A. Murua, and J. M. Sanz-Serna. Symplectic methods based on decompositions. *SIAM Journal on Numerical Analysis*, 34(5):1926–1947, 1997.

[AP98] U. Ascher and L. Petzold. *Computer methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.

[ARW95] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton. Implicit-Explicit methods for time-dependent Partial Differential Equations. *SIAM Journal on Numerical Analysis*, 32(3):797–823, 1995.

[AV13] A. Abdulle and G. Vilmart. PIROCK: A swiss-knife partitioned implicit-explicit orthogonal Runge-Kutta Chebyshev integrator for stiff diffusion-advection-reaction problems with or without noise. *Journal of Computational Physics*, 242:869–888, 06 2013.

[BAA+19a] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.12, Argonne National Laboratory, 2019.

[BAA+19b] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Web page. https://www.mcs.anl.gov/petsc, 2019.

[BC89] M.J. Berger and P. Colella. Local Adaptive Mesh Refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64 – 84, 1989.

[BDA+06] J.B. Bell, M.S. Day, A. Almgren, M. Lijewski, C. Rendleman, R. Cheng, and I. Shepherd. Simulation of lean premixed turbulent combustion. *Journal of Physics: Conference Series*, 46(1):1, 2006.

[BDG02] J.B. Bell, M.S. Day, and J.F. Grcar. Numerical simulation of premixed turbulent methane combustion. *Proc. Combust. Inst.*, 29(2):1987–1993, 2002.

[BDG+07] J.B. Bell, M.S. Day, J.F. Grcar, M.J. Lijewski, J.F. Driscoll, and S.A. Filatyev. Numerical simulation of a laboratory-scale turbulent slot flame. *Proc. Combust. Inst.*, 31(1):1299–1307, 2007.

[BDS+05] J. B. Bell, M. S. Day, I. G. Shepherd, M. R. Johnson, R. K. Cheng, J. F. Grcar, V. E. Beckner, and M. J. Lijewski. Numerical

simulation of a laboratory-scale turbulent V-flame. *Proceedings of the National Academy of Sciences*, 102(29):10006–10011, 2005.

[Ber82] M.J. Berger. *Adaptive Mesh Refinement for hyperbolic differential equations*. PhD thesis, Stanford University, 1982.

[BGL05] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[BGMS97] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in Object Oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[BH96] B. L. Bihari and A. Harten. Multiresolution schemes for the numerical solution of 2D conservation laws I. *SIAM Journal on Scientific Computing*, 18(2):315 – 354, 1996.

[BMBL+13] I. Bermejo-Moreno, J. Bodart, J. Larsson, B. Barney, J. Nichols, and S. Jones. Solving the compressible navier-stokes equations on up to 1.97 million cores and 4.1 trillion grid points. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2013.

[BO84] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[BP84] F. Brezzi and J. Pitkäranta. On the stabilization of finite element approximations of the Stokes equations. *Proceedings of a GAMM-Seminar Kiel. Notes on Numerical Fluid Mechanics*, 10:11 – 19, 1984.

[BP98] O. Botella and R. Peyret. Benchmark spectral results on the lid-driven cavity flow. *Computers and Fluids*, 27(4):421 – 433, 1998.

[BS99] B. A. V. Bennett and M. D. Smooke. Local rectangular refinement with application to nonreacting and reacting fluid flow problems. *Journal of Computational Physics*, 151(2):684 – 727, 1999.

[BT69] W. D. Baines and J. S. Turner. Turbulent buoyant convection from a source in a confined region. *Journal of Fluid Mechanics*, 37(1):51–80, 1969.

[But64] J.C. Butcher. Implicit runge-kutta processes. *Math. Comp.*, 18:50–64, 1964.

[BWG11] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[CCdS+09] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery*, 2(1):015001, 2009.

[CDD04] A. Cohen, W. Dahmen, and R. DeVore. *Adaptive Wavelet Techniques in Numerical Simulation.* John Wiley & Sons, Ltd., 2004.

[CDF92] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure and Applied Math.*, 45(5):485–560, 1992.

[CEGH14] E. Chénier, R. Eymard, T. Gallouët, and R. Herbin. An extension of the MAC scheme to locally refined meshes : convergence analysis for the full tensor time-dependent navier-stokes equations. *Calcolo*, 52(1):69–107, 2014.

[CEH09] E. Chénier, R. Eymard, and R. Herbin. A collocated finite volume scheme to solve free convection for general non-conforming grids. *J. Comput. Physics*, 228:2296–2311, 2009.

[CET06] E. Chénier, R. Eymard, and O. Touazi. Numerical results using a colocated finite-volume scheme on unstructured grids for incompressible fluid flows. *Numerical Heat Transfer, Part B: Fundamentals*, 49(3):259–276, 2006.

[CH52] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *Proceedings of the National Academy of Sciences*, 38(3):235–243, 1952.

[Che11] Jacqueline H. Chen. Petascale direct numerical simulation of turbulent combustion—fundamental insights towards predictive models. *Proceedings of the Combustion Institute*, 33(1):99–123, 2011.

[Cho68] A. Chorin. Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22:745–762, 1968.

[CK98] S. H. Chou and D. Y. Kwak. A covolume method based on rotated bilinears for the generalized Stokes problem. *SIAM Journal on Numerical Analysis*, 35(2):494–507, 1998.

[CKMP03] A. Cohen, S. M. Kaber, S. Muller, and M. Postel. Fully adaptive multiresolution finite volume schemes for conservation laws. *Mathematics of computation*, 72(241), 2003.

[Dah63] G. Dahlquist. A special stability problem for linear multistep methods. *Nordisk Tidskr. Informations-Behandling*, 3:27–43, 1963.

[Dau88] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. Pure and Applied Math.*, 41(7):909–996, 1988.

[Dau92] I. Daubechies. *Ten Lectures on Wavelets*, volume 61 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[DB00a] M. Day and J. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combustion Theory and Modelling*, 4(4):535–556, 2000.

[DB00b] M.S. Day and J.B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4:535–556, 2000.

[DB00c] M.S. Day and J.B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4:535–556, 2000.

[DBM+12] M. Duarte, Z. Bonaventura, M. Massot, A. Bourdon, S. Descombes, and T. Dumont. A new numerical strategy with space-time adaptivity and error control for multi-scale streamer discharge simulations. *Journal of Computational Physics*, 231(3):1002 – 1019, 2012. Special Issue: Computational Plasma Physics.

[DBMB15] M. Duarte, Z. Bonaventura, M. Massot, and A. Bourdon. A numerical strategy to discretize and solve the poisson equation on dynamically adapted multiresolution grids for time-dependent streamer discharge simulations. *Journal of Computational Physics*, 289:129 – 148, 2015.

[DDD+11] S. Descombes, M. Duarte, T. Dumont, V. Louvet, and M. Massot. Adaptive time splitting method for multi-scale evolutionary partial differential equations. *Confluentes Mathematici*, 3(3):413–443, 2011.

[DDD+13] T. Dumont, M. Duarte, S. Descombes, M.-A. Dronne, M. Massot, and V. Louvet. Simulation of human ischemic stroke in realistic 3D geometry. *Communications in Nonlinear Science and Numerical Simulation*, 18(6):1539–1557, 2013.

[DDD+15] S. Descombes, M. Duarte, T. Dumont, T. Guillet, V. Louvet, and M. Massot. Task-based adaptive multiresolution for time-space multi-scale reaction-diffusion systems on multi-core architectures. *SMAI Journal of Computational Mathematics*, 3, 2015.

[DDT+13] M. Duarte, S. Descombes, C. Tenaud, S. Candel, and M. Massot. Time–space adaptive numerical methods for the simulation of combustion fronts. *Combustion and Flame*, 160(6):1083–1101, 2013.

[Dei05] R. Deiterding. Construction and application of an AMR algorithm for distributed memory computers. In T.J. Barth *et al.*, editor, *Adaptive Mesh Refinement - Theory and Applications*, volume 41 of *Lecture Notes in Computational Science and Engineering*, pages 361–372. Springer Berlin Heidelberg, 2005.

[DeV98] R. A. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998.

[DJOR16] S. Dellacherie, J. Jung, P. Omnes, and P.-A. Raviart. *Construction of modified godunov-type schemes accurate at any mach number for the compressible euler system*, volume 26 of *Mathematical models and methods in ppplied sciences*. World Scientific Publishing, 2016.

[Dru17] F. Drui. *Eulerian modeling and simulations of separated and disperse two-phase flows : development of a unified modeling approach and associated numerical methods for highly parallel computations*. PhD thesis, École Centrale Paris, France, 2017.

[DS84] R. DeVore and R. Sharpley. *Maximal Functions Measuring Smoothness*, volume 47. Mem. Am. Math. Soc., 1984.

[DT09] V. Daru and C. Tenaud. Numerical simulation of the viscous shock tube problem by using a high resolution monotonicity-preserving scheme. *Computers & Fluids*, 38(3):664 – 676, 2009.

[DTB+15] M. Day, S. Tachibana, J. Bell, M. Lijewski, V. Beckner, and R. K. Cheng. A combined computational and experimental characterization of lean premixed turbulent low swirl laboratory flames ii. hydrogen flames. *Combustion and Flame*, 162(5):2148 – 2165, 2015.

[Dua11] M. Duarte. *Adaptive numerical methods in time and space for the simulation of multi-scale reaction fronts.* PhD thesis, École Centrale Paris, France, 2011.

[EGH00] R. Eymard, T. Gallouët, and R. Herbin. In *Finite volume methods*, volume 7 of *Handbook of Numerical Analysis*, pages 713 – 1018. Elsevier, 2000.

[EGH10] R. Eymard, T. Gallouët, and R. Herbin. Discretization of heterogeneous and anisotropic diffusion problems on general nonconforming meshes SUSHI: a scheme using stabilization and hybrid interfaces. *IMA Journal of Numerical Analysis*, 30(4):1009–1043, 2010.

[Ehl69] B.L. Ehle. On padé approximations to the exponential function and a-stable methods for the numerical solution of initial value problems. *Research Report CSRR 2010*, 1969.

[EMB+06] U. Ebert, C. Montijn, T. M. P. Briels, W. Hundsdorfer, B. Meulenbroek, A. Rocco, and E.M. van Veldhuizen. The multiscale nature of streamers. *Plasma Sources Sci. Technol.*, 15:S118–S129, 2006.

[FL06] F. N. Felten and T. S. Lund. Kinetic energy conservation issues associated with the collocated mesh scheme for incompressible flow. *Journal of Computational Physics*, 215(2):465 – 484, 2006.

[For16] C. J. Forster. *Parallel wavelet-adaptive Direct Numerical Simulation of multiphase flows with phase-change.* PhD thesis, Georgia Institue of Technology, Atlanta, Gerogia, 2016.

[FP99] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics.* Springer, Berlin, 1999.

[FPG15] L. Freret and C. P. Groth. Anisotropic non-uniform block-based adaptive mesh refinement for three-dimensional inviscid and viscous flows. In *22nd AIAA Computational Fluid Dynamics Conference*, 2015.

[GC90] M.P. Gresho and T.S. Chan. On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 2: Implementation. *International Journal for Numerical Methods in Fluids*, 11:621–659, 1990.

[Get98] A. V. Getling. *Rayleigh-Bénard convection: structure and dynamcis*, volume 11 of *Advanced series in nonlinear dynamcis*. World Scientific Publishing, Singapore, 1998.

[GGS82] U Ghia, K.N Ghia, and C.T Shin. High-re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387 – 411, 1982.

[GHLM16] T. Gallouët, R. Herbin, J.-C. Latché, and K. Mallem. Convergence of the Marker-and-Cell scheme for the incompressible Navier-Stokes equations on non-uniform grids. *Foundations of Computational Mathematics*, 18(1):249–289, 2016.

[Gio99] V. Giovangigli. *Multicomponent flow modeling*. Modeling and simulation in science, engineering and technology. Birkhauser Basel, 1999.

[GM19] J.-L. Guermond and P. Minev. High-order adaptive time stepping for the incompressible Navier-Stokes equations. *SIAM Journal on Scientific Computing*, 41:A770–A788, 01 2019.

[GMS06] J.L. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195:6011–6045, 2006.

[GP82] C. Gear and L. Petzold. ODE methods for the solution of Differential / Algebraic systems. *SIAM Journal on Numerical Analysis*, 21, 1982.

[GPMD88] G. Goyal, P.J. Paul, H.S. Mukunda, and S.M. Deshpande. Time dependent operator-split and unsplit schemes for one dimensional premixed flames. *Combust. Sci. Technol.*, 60:167–189, 1988.

[GQ98a] J.-L. Guermond and L. Quartapelle. On stability and convergence of projection methods based on pressure Poisson equation. *International Journal for Numerical Methods in Fluids*, 26:1039–1053, 05 1998.

[GQ98b] J.-L. Guermond and L. Quartapelle. On the approximation of the unsteady Navier-Stokes equations by finite element projection methods. *Numerische Mathematik*, 80:207–238, 1998.

[Gre90] M.P. Gresho. On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 1: Theroy. *International Journal for Numerical Methods in Fluids*, 11:587–620, 1990.

[GS87] M.P. Gresho and R.L. Sani. On pressure boundary conditions for the incompressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 7:621–659, 1987.

[GS00] P.M. Gresho and R.L. Sani. *Incompressible flow and the finite element method, advection-diffusion and isothermal laminar flow.* Incompressible Flow and the Finite Element Method. John Wiley & Sons, 2000.

[GS01] J.-L. Guermond and J. Shen. Quelques résultats nouveaux sur les méthodes de projection. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 333(12):1111 – 1116, 2001.

[GS03] J. L. Guermond and J. Shen. A new class of truly consistent splitting schemes for incompressible flows. *J. Comput. Phys.*, 192(1):262–276, 2003.

[GTG15] A. Guittet, M. Theillard, and F. Gibou. A stable projection method for the incompressible Navier-Stokes equations on arbitrary geometries and adaptive quad/octrees. *Journal of Computational Physics*, 292:215 – 238, 2015.

[Gue97] J.-L. Guermond. Un résultat de convergence d'ordre deux pour l'approximation des équations de Navier-Stokes par projection incrémentale. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 325(12):1329 – 1332, 1997.

[GV99] Hervé Guillard and Cécile Viozat. On the behaviour of upwind schemes in the low Mach number limit. *Computers and Fluids*, 28(1):63–86, 1999.

[Har94a] A. Harten. Adaptive multiresolution schemes for shock computations. *Journal of Computational Physics*, 115(2):319 – 338, 1994.

[Har94b] A. Harten. Multiresolution representation and numerical algorithms: A brief review. *ICASE Rep. 94-59*, 1994.

[Har95] A. Harten. Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. *Communications on Pure and Applied Mathematics*, 48(12):1305–1342, 1995.

[HB97] L. H. Howell and J. B. Bell. An adaptive mesh projection method for viscous incompressible flow. *SIAM Journal on Scientific Computing*, 18(4):996–1013, 1997.

[HCK+12] E. R. Hawkes, O. Chatakonda, H. Kolla, A. R. Kerstein, and J. H. Chen. A petascale Direct Numerical Simulation study of the modelling of flame wrinkling for large-eddy simulations in intense turbulence. *Combustion and Flame*, 159(8):2690 – 2703, 2012. Special Issue on Turbulent Combustion.

[HK05] G. R. Hunt and N. B. Kaye. Lazy plumes. *Journal of Fluid Mechanics*, 533:329–338, 2005.

[HLR89] E. Hairer, C. Lubich, and M. Roche. *The numerical solution of Differential-Algebraic systems by Runge-Kutta methods*. Lectures notes in mathematics. Springer-Verlag Berlin Heidelberg, 1989.

[HNW87] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I*, pages xiv+480. Springer-Verlag, Berlin, 1987. Nonstiff Problems.

[HW65] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2189, 1965.

[HW96] E. Hairer and G. Wanner. *Solving ordinary differential equations II*. Springer series in computational mathematics. Springer-Verlag Berlin Heidelberg, 1996. Stiff and differential-algebraic problems.

[HW99] E. Hairer and G. Wanner. Stiff differential equations solved by Radau methods. *Journal of Computational and Applied Mathematics*, 111(1):93–111, 1999.

[JL15] S. Jones and A. Lichtl. GPUs to Mars, full scale simulation of SapceX's Mars rocket engine. In *Proceedings of the GPU Technology Conference*, Silicon Valley, 2015.

[KC03] C. A. Kennedy and M. H. Carpenter. Additive Runge-Kutta schemes for convection-diffusion-reaction equations. *Applied Numerical Mathematics*, 44(1):139 – 181, 2003.

[KC16] C. Kennedy and M. Carpenter. Diagonally Implicit Runge-Kutta methods for Ordinary Differential Equations. A review, 03 2016.

[KM85] J Kim and P. Moin. Application of a fractional-step method to incompressible Navier-Stokes equations. *Journal of Computational Physics*, 59(2):308–323, 1985.

[KNW99] O.M. Knio, H.N. Najm, and P.S. Wyckoff. A semi-implicit numerical scheme for reacting flow. II. Stiff, operator-split formulation. *J. Comput. Phys.*, 154:482–467, 1999.

[Lad69] 0. Ladyzhenskaya. *The mathematical theory of viscous incompressible flows*. Gordon and Breach, 1969.

[LNS+18] H. Leclerc, M.-A. N'Guessan, L. Séries, L. Gouarin, and M. Massot. Parallel dedicated data structures and adaptive multiresolution implementations: application to the resolution of multi-scale PDEs. In *NASA Technical Memorandum, Proceedings of the 2018 Summer Program*, NASA Ames Research Center, 2018.

[LP86a] P. Lötstedt and L. Petzold. Numerical solution of nonlinear differential equations with algebraic constraints I: convergence results for backward differentiation formulas. *Mathematics of Computation*, 46(174):491–516, 1986.

[LP86b] P. Lötstedt and L. Petzold. Numerical solution of nonlinear differential equations with algebraic constraints II: practical implications. *SIAM Journal on Scientific and Statistical Computing*, 7(3):720–733, 1986.

[Mal89] S. Mallat. Multiresolution approximation and wavelets orthonormal bases of $L^2(R)$. *Trans. Amer. Math. Soc.*, 315:69–87, 1989.

[MC00] D. F. Martin and P. Colella. A cell-centered adaptive projection method for the incompressible euler equations. *Journal of Computational Physics*, 163(2):271 – 312, 2000.

[MG06] C. Min and F. Gibou. A second order accurate projection method for the incompressible Navier-Stokes equations on non-graded adaptive grids. *J. Comput. Phys.*, 219(2):912–929, 2006.

[MGDV02] M. Massot, S. Génieys, T. Dumont, and V. A. Volpert. Interaction of thermal explosion and natural convection: critical conditions and new oscillating regimes. *SIAM Journal of Applied Mathematics*, 63:351–372, 2002.

[MLVM98] Y. Morinishi, T.S. Lund, O.V. Vasilyev, and P. Moin. Fully conservative higher order finite difference schemes for incompressible flow. *Journal of Computational Physics*, 143(1):90 – 124, 1998.

[MS85] A. Madja and J. Sethian. The derivation and numerical solution of the equations for Zero Mach Number combustion. *Combustion Science and Technology*, 42(3-4):185–205, 1985.

[MTS+98] R. K. Mohammed, M. A. Tanoff, M. D. Smooke, A. M. Schaffer, and M. B. Long. Computational and experimental study of a forced, time varying, axisymmetric, laminar diffusion flame.

Symposium (International) on Combustion, 27(1):693 – 702, 1998. Twenty-Seventh Symposium (International) on Combustion Volume One.

[NFG17] Nishant Narechania, L. Freret, and C. Groth. Block-based anisotropic AMR with a posteriori adjoint-based error estimation for three-dimensional inviscid and viscous flows. In *23rd AIAA Computational Fluid Dynamics Conference*, 2017.

[NG09] S. A. Northrup and C. P.T. Groth. Solution of laminar combusting flows using a parallel implicit Adaptive Mesh Refinement algorithm. In *Computational Fluid Dynamics 2006*, pages 341–346, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Nic92] R. A. Nicolaides. Analysis and convergence of the MAC scheme I. The linear problem. *SIAM Journal on Numerical Analysis*, 29(6):1579–1591, 1992.

[Nic00] F. Nicoud. Conservative high-order Finite-Difference schemes for low-Mach number flows. *Journal of Computational Physics*, 158(1):71 – 97, 2000.

[NK05] H.N. Najm and O.M. Knio. Modeling Low Mach number reacting flow with detailed chemistry and transport. *J. Scientific Computing*, 25(1/2):263–287, 2005.

[NMST19] M.-A. N'Guessan, M. Massot, L. Séries, and C. Tenaud. High order time integration and mesh adaptation with error control for incompressible Navier-Stokes and scalar transport resolution on dual grids. *Journal of Computational and Applied Mathematics*, page 112542, 2019.

[NSTM18] M.-A. N'Guessan, L. Séries, C. Tenaud, and M. Massot. A high-order runge-kutta method coupled to a multiresolution strategy to solve the incompressible Navier-Stokes equations. In *NASA Technical Memorandum, Proceedings of the 2018 Summer Program*, NASA Ames Research Center, 2018.

[NW77] Syvert P. Nørsett and Arne Wolfbrandt. Attainable order of rational approximations to the exponential function with only real poles. *BIT Numerical Mathematics*, 17(2):200–208, Jun 1977.

[NW96] R. A. Nicolaides and X. Wu. Analysis and convergence of the MAC scheme II. Navier-Stokes equations. *Mathematics of Computation*, 65(213):29–44, 1996.

[NW97] H. N. Najm and P. S. Wyckoff. Premixed flame response to unsteady strain rate and curvature. *Combustion and Flame*, 110(1):92 – 112, 1997.

[NWK98a] H.N. Najm, P.S. Wyckoff, and O.M. Knio. A semi-implicit numerical scheme for reacting flow. I. stiff chemistry. *J. Comput. Phys.*, 143:381–402, 1998.

[NWK98b] H.N. Najm, P.S. Wyckoff, and O.M. Knio. A semi-implicit numerical scheme for reacting flow. I. stiff chemistry. *J. Comput. Phys.*, 143:381–402, 1998.

[NWK99] H.N. Najm, P.S. Wyckoff, and O.M. Knio. A semi-implicit numerical scheme for reacting flow: II. stiff, operator-split formulation. *Journal of Computational Physics*, 154(2):428 – 467, 1999.

[OID86] S. A. Orszag, M. Israeli, and M. O. Deville. Boundary conditions for incompressible flows. *Journal of Scientific Computing*, 1(1):75–111, 1986.

[Pat80] S. V. Patankar. *Numerical heat transfer and fluid flow.* Series on Computational Methods in Mechanics and Thermal Science. Hemisphere Publishing Corporation (CRC Press, Taylor & Francis Group), 1980.

[Pet82] L. Petzold. Differential / Algebraic equations are not ODE's. *SIAM J. Sci. Stat. Comput.*, 3(3):367–384, 1982.

[PHB+98] R. B. Pember, L. H. Howell, J. B. Bell, P. Colella, W. Y. Crutchfield, W. A. Fiveland, and J. P. Jessee. An adaptive projection method for unsteady, low-Mach number combustion. *Combustion Science and Technology*, 140(1-6):123–168, 1998.

[Pop03] S. Popinet. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572 – 600, 2003.

[Pos01] M. Postel. Approximations multiéchelles. In *École de printemps de mécanique des fluides numérique*, Aussois, 2001.

[Pos05] M. Postel. Approximations multiéchelles. In *Neuvième École Mécanique des Fluides Numérique, Roscoff*, pages 1–59. Université Pierre et Marie Curie, Lab. Jacques-Louis Lions, 2005.

[PR74] A Prothero and A Robinson. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential

equations. *Mathematics of Computation - Math. Comput.*, 28:145–145, 1974.

[PR01] L. Pareschi and G. Russo. *Implicit-Explicit Runge-Kutta schemes for stiff systems of differential equations*, volume 3, pages 269–288. 01 2001.

[Ran06] R. Rannacher. *On Chorin's projection method for incompressible Navier-Stokes cquations*, volume 1530, pages 167–183. 11 2006.

[RC83] M. Rhie and W. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21:1525–1532, 1983.

[RP08] Z. Ren and S.B. Pope. Second-order splitting schemes for a class of reactive systems. *J. Comput. Phys.*, 227(17):8165–8176, 2008.

[RP13] R. Ranjan and C. Pantano. A collocated method for the incompressible Navier-Stokes equations inspired by the box scheme. *J. Comput. Phys.*, 232(1):346–382, 2013.

[RPB14] J. Reveillon, C. Pera, and Z. Bouali. Examples of the potential of DNS for the understanding of reactive multiphase flows. *International Journal of Spray Combustion Dynamics*, 2014.

[RS05] O. Roussel and K. Schneider. An adaptive multiresolution method for combustion problems: Application to flame ball-vortex interaction. *Computers & Fluids*, 34(7):817–831, 2005.

[Saa03] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

[San13] B. Sanderse. Energy-conserving runge-kutta methods for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 233:100–131, 2013.

[Sch03] Alfred Schmidt. A multi-mesh finite element method for phase-field simulations. In *Interface and Transport Dynamics*, pages 208–217, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Sch04] Alfred Schmidt. A multi-mesh finite element method for 3D phase field simulations. In *Free Boundary Problems*, pages 293–301, Basel, 2004. Birkhäuser Basel.

[Sha12] S. Shashank. *High fidelity simulations of reactive liquid-fuel jets*. PhD thesis, Stanford University, United States, 2012.

[She92] J. Shen. On error estimates of projection methods for Navier-Stokes equations: first-order schemes. *SIAM Journal on Numerical Analysis*, 29(1):57–77, 1992.

[She96] J. Shen. On error estimates of the projection methods for the Navier-Stokes equations: second-order schemes. *Mathematics of Computation*, 65(215):1039–1065, 1996.

[SLGS03] D.A. Schwer, P. Lu, W.H. Green, and V. Semião. A consistent-splitting approach to computing stiff steady-state reacting flows with adaptive chemistry. *Combust. Theory Modelling*, 7(2):383–399, 2003.

[SLI10] Shashank, Johan Larsson, and Gianluca Iaccarino. A co-located incompressible Navier-Stokes solver with exact mass, momentum and kinetic energy conservation in the inviscid limit. *J. Comput. Physics*, 229:4425–4430, 2010.

[SPN06] M.A. Singer, S.B. Pope, and H.N. Najm. Operator-splitting with ISAT to model reacting flow with detailed chemistry. *Combust. Theory Modelling*, 10(2):199–217, 2006.

[SRN10] C. Safta, J. Ray, and H. N. Najm. A high-order low-Mach number AMR construction for chemically reacting flows. *Journal of Computational Physics*, 229(24):9299 – 9322, 2010.

[Str63] G. Strang. Accurate partial difference methods. I. Linear Cauchy problems. *Arch. Ration. Mech. Anal.*, 12:392–402, 1963.

[Str68] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5:506–517, 1968.

[TB14] F. Tholin and A. Bourdon. Influence of the external electrical circuit on the regimes of a nanosecond repetitively pulsed discharge in air at atmospheric pressure. *Plasma Physics and Controlled Fusion*, 57(1):014016, 2014.

[TD11a] C. Tenaud and M. Duarte. Tutorials on adaptive multiresolution for mesh refinement applied to fluid dynamics and reactive media problems. *ESAIM: Proc.*, 34:184–239, 2011.

[TD11b] Tenaud, Christian and Duarte, Max. Tutorials on adaptive multiresolution for mesh refinement applied to fluid dynamics and reactive media problems. *ESAIM: Proc.*, 34:184–239, 2011.

[Tem69] R. Temam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires. *Archive for Rational Mechanics and Analysis*, 32:135–153, 1969.

[Tem77] R. Temam. *Navier-Stokes Equations: Theory and Numerical Analysis*. Studies in Mathematics and its Applications. Lions, J.L. and Papanicolaou, G. and Rockafellar, R.T., 1977.

[Ten10] C. Tenaud. An adaptive multiresolution technique for unsteady compressible flow simulations. In *Symposium on Applied Aerodynamics*, Marseille, France, 2010.

[TMVdV96] L. Timmermans, P.D. Minev, and F. Van de Vosse. An approximate projection scheme for incompressible flow using spectral elements. *International Journal for Numerical Methods in Fluids*, 22:673 – 688, 1996.

[Tur98] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. 1998.

[UAH58] H. Uzawa, K. J. Arrow, and L. Hurwicz. *Studies in non-linear programming*. Stanford University Press, Stanford, 1958.

[VK86] J. Van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *Siam Journal on Scientific and Statistical Computing*, 7, 1986.

[VPB94] P. A. Vitello, B. M. Penetrante, and J. N. Bardsley. Simulation of negative-streamer dynamics in nitrogen. *Physical Review E*, 49:5574–5598, 1994.

[War19] Q. Wargnier. *Mathematical modeling and simulation of non-equilibrium plasmas: application to magnetic reconnection in the Sun atmosphere*. PhD thesis, École Polytechnique, France, 2019.

[WBCK02] R. Williams, K. Burrage, I. Cameron, and M. Kerr. A four-stage index 2 Diagonally Implicit Runge-Kutta method. *Applied Numerical Mathematics*, 40(3):415 – 432, 2002.

[YP98] B. Yang and S.B. Pope. An investigation of the accuracy of manifold methods and splitting schemes in the computational implementation of combustion chemistry. *Combust. and Flame*, 112(1-2):16–32, 1998.

[ZAB+19] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers,

T. Nguyen, A. Nonaka, M. Rosso, S. Williams, and M. Zingale. Amrex: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4:1370, 2019.

[ZHA+11] W. Zhang, L. Howell, A. Almgren, A. Burrows, and John Bell. CASTRO: a new compressible astrophysical solver. II. Gray radiation hydrodynamics. *The Astrophysical Journal Supplement Series*, 196, 2011.

[ZHA+12] W. Zhang, L. Howell, A. Almgren, A. Burrows, J. Dolence, and John Bell. CASTRO: a new compressible astrophysical solver. III. Multigroup radiation hydrodynamics. *The Astrophysical Journal Supplement Series*, 204, 2012.

[ZSK94] Y. Zang, R. Street, and J. Koseff. A non-staggered grid, fractional step method for time-dependent incompressible Navier-Stokes equations in curvilinear coordinates. *Journal of Computational Physics*, 114(1):18–33, 1994.