Efficient methods for steady multi-physics simulations

Pierre Seize

ONERA
THE FRENCH AEROSPACE LAB

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *13 mars 2023* **par :**

Pierre Seize

Méthodologies permettant l'obtention efficace de solutions multi-physiques stationnaires pour des applications en énergétique

---

JURY

Pierre-Henri MAIRE	Directeur de recherche	Président du jury
Jocelyne ERHEL	Directrice de recherche	Membre du jury
Xavier VASSEUR	Docteur, HDR	Membre du jury
Rodolphe TURPAULT	Professeur des Universités	Rapporteur
Vincent PERRIER	Chargé de recherche, HDR	Rapporteur
Guillaume PUIGT	Directeur de recherche	Directeur de thèse
Lionel MATUSZEWSKI	Docteur	Co-Directeur de thèse

École doctorale et spécialité :
MEGEP : Dynamique des fluides

Unité de Recherche :
Office National d'Études et de Recherches Aérospatiales - Département Multi-Physique pour l'Énergétique

Directeurs de Thèse :
Guillaume PUIGT et *Lionel MATUSZEWSKI*

Rapporteurs :
Rodolphe TURPAULT et *Vincent PERRIER*

Remerciements

Même si trois années passent vite, une thèse est une véritable période de vie, durant laquelle beaucoup de personnes se retrouvent investies. Je tiens à remercier ici ces personnes qui m'auront accompagné, aidé et soutenu durant cette période.

Je commence par remercier les membres du jury qui ont accepté leur rôle. Je remercie en particulier les rapporteurs. Comme j'ai eu l'occasion de leur dire, leurs rapports sont d'un grand intérêt car ils permettent de prendre du recul sur son propre travail, en donnant un point de vue critique mais surtout extérieur sur la thèse. Enfin, je remercie Xavier Vasseur qui n'était pas à priori lié à cette thèse mais qui y a contribué, en répondant à mes questions, me fournissant des références et enfin en acceptant le rôle de juré.

Ma thèse s'étant déroulée initialement à Toulouse puis à Châtillon, je n'ai jamais été physiquement avec mes deux encadrants en même temps, à part de rares occasions. Cependant, je ne me suis jamais senti seul dans mon travail. Que ce soit au téléphone, par chat, par mail ou lors du point hebdomadaire, vous étiez toujours disponible. Pour ceci, Guillaume et Lionel, je vous suis extrêmement reconnaissant et je ne peux que recommander un tel encadrement de thèse. Je pense aussi que vos qualités d'encadrants sont liées à vos personnalités : en dehors du contexte professionnel, nous nous sommes toujours bien entendus, ce qui facilite d'autant plus nos échanges. Merci à tous les deux, à Guillaume pour ton point de vue optimiste lorsque je n'en avais pas, à Lionel de m'avoir laissé te harceler par chat et au téléphone. Je suis convaincu que vous avez contribué au bon déroulement de cette aventure, et pour cela merci encore.

J'ai donc commencé ma thèse dans l'unité HEAT avant de déménager dans l'unité PLM. Ce fut pour moi l'occasion de rencontrer davantage de personnes qui ont ajouté leur pierre à l'édifice. Certes, cela veut en fait dire que j'avais d'avantage de personnes à aller embêter avec mes questions, mais j'avais toujours de l'aide en retour. Je remercie donc les membres de ces deux unités, et même plus largement les membres du DMPE, qui m'ont aidé durant ces trois ans.

Parmi eux, je remercie en particulier Jean-Michel Lamet. C'est parce qu'il s'est intéressé à ma thèse que nous avons pu trouver un avantage immédiat à l'utilisation de mes travaux. Merci pour le temps que tu m'as consacré, à re-travailler ton modèle, échanger avec moi, et même relire une partie de ce mémoire !

Plus largement, je tiens à remercier l'ONERA et son personnel. Je m'adresse ici aux personnes qui ont n'ont pas contribué à ma thèse d'un point de vue scientifique, mais qui ont tout de même apporté leur aide.

En-dehors du personnel ONERA, je remercie les différentes personnes qui ont fait partie du support CEDRE. Je pense notamment à Julien R. qui m'impressionna par ses compétences et son investissement. Je pense également à Alexandre et Christ, avec qui j'ai pu râler sur les problèmes et les bugs que je rencontrais. À ces personnes s'ajoutent Julien V., Yacine, ou encore mes collègues doctorants. Merci à vous tous pour ces moments d'échanges parfois scientifique, parfois moins.

Il ne faut pas oublier le plus grand contributeur de cette thèse, sans qui ce mémoire n'existerait pas aujourd'hui. Merci Zhao pour ton ordi !

Je remercie enfin ma famille pour son soutien, et en particulier mes parents. Vous dites être fiers de moi, mais je suis comme je suis de par votre éducation et votre amour, merci. Merci Manon, de partager ma vie, et d'en faire un bonheur.

Pour mamie Raymonde, qui disait que plus tard, je serai ingénieur.

Contents

Introduction

General context

This work is rooted in the field of numerical simulation of fluid dynamics, applied to aeronautics, defense, and space industries. This field gathers many industrial actors (the French Direction Générale de l'Armement, ArianeGroup, Safran, Airbus, etc) as well as academics (ONERA, CERFACS, DLR, VKI, universities, etc). It is interested in how to simulate a fluid flow with a computer, trying to represent faithfully the physical reality. The various actors in this field need to be able to access certain physical quantities associated with specific phenomena and operating regimes. These regimes are often not feasible on our scale, due to material or financial limitations. Examples include the study of icing on the wing of an aircraft, which is experimentally feasible but represents an imposing budget for the aircraft manufacturer, or the study of heat transfer in an atmospheric reentry capsule, which is much more difficult to achieve experimentally. To overcome these limitations, numerical simulation is the best option, as it allows such a case study to be modelled by the execution of a computer program, and to obtain a large set of data that will be analysed afterwards to answer the desired questions.

The analysis of physics usually produces a set of equations, often partial differential equations, representing the real system that one wishes to study. Algorithms are then required to determine the fluid flow from these equations, in the working domain, as a function of time. Thus, to obtain the desired quantities, the physical system is integrated using mathematical algorithms to obtain its evolution in time. Often, the expected result is not the complete temporal evolution of the system, but only its equilibrium state. This is called a steady numerical simulation or steady computation. Steady computations are opposed to unsteady computations that aim to accurately describe the system's temporal evolution.

For all computational fluid dynamics users, the most crucial issue is to achieve a satisfying compromise between computational cost and the accuracy of the results. Indeed, a quick and inexpensive computation tends to be not very faithful to physics, while an accurate computation tends to use more computational resources and time. For steady computations, speed corresponds to getting the final steady state at a low time cost, for both computational time and the time it actually took to a user. It results in a compromise between methods that are expensive in terms of computational resources and take a long time but give accurate results that are close to the physical reality, and faster methods that save resources but give lesser quality results. A software developer working on a numerical simulation tool needs to choose methods and algorithms to obtain a compromise that is considered satisfactory. The final interest for a player in computational fluid dynamics is therefore to have a result that is sufficiently precise and inexpensive enough to obtain. An accurate result is needed to answer the questions that required

the simulation. The search for a result that is inexpensive to obtain is motivated by questions of savings in computational cost. This cost is applied to the user, while he waits for the simulation to end, and to the company through the cost of computer resources, electricity, investments in more efficient machines, etc.

Depending on the problem to be solved, there are more or less suitable algorithms and methods. Performant methods were originally developed to answer the needs of the aerodynamics community. In the case of energetics and multiphysics problems, the methods issued from the more traditional aerodynamics are limited by the coupling between the different physics which have their distinct characteristic times. Thus, the algorithms used for the numerical simulation of classical fluid dynamics are focused on the resolution of the Navier–Stokes equations, and are not necessarily the most adapted to a simulation in the field of energetics. The involvement of several distinct physical phenomena imposes constraints on the choice and use of algorithms. Consequently, it would be advisable to adapt or replace the algorithms involved in the time integration for multiphysics problems.

Context of this thesis

This thesis was conducted at ONERA, the French Aerospace Lab, in the multiphysics for energetics department. Although numerical simulation has been greatly developed in the aeronautical field, it has not been so well adapted to the multiphysics field, and many industrial codes are content to reuse the same algorithms. This is the example of the software system CEDRE, developed at ONERA by the multiphysics for energetics department [1]. This software system constitutes a platform grouping several solvers to integrate several physics: each solver is dedicated to its physical model. There is a solver for the resolution of compressible, multifluid, reactive, and turbulent flows, two solvers for the calculation of dispersed phase (drops, crystals, particles) in the Eulerian and Lagrangian approach respectively, a solver dedicated to the calculation of liquid films, a solver dedicated to radiation, etc. Thus, CEDRE is in fact a global platform adapted to multiphysics problems. Older work has been done to set up a time integration adapted to the problems solved by CEDRE [2]. Work on time integration has led to the development of implicit integration methods for the integration of steady problems, and to the development of a GMRES method for the solution of linear systems. Users can choose from a panel of integration methods to obtain methods adapted to their problems. However, the weak coupling between solvers hampers the convergence to the steady state and the choice in integration methods is limited, compared to what can be found in the literature. This is at least the opinion of the actors of the CEDRE code, i.e. its developers and users, who would like more robust methods to use CEDRE on steeper problems and converge faster to save on computational costs.

On the research side, however, many efforts have been made in the direction of multiphysics, but have not yet left the academic framework. This is for example the case of [3], who were interested in the temporal integration of coupled equations. They designed an integration method adapted to coupled equations, which is an evolution of a fixed point method with the addition of a step of a Newton method. They then compared this method to the standard fixed point method, which is more commonly used for coupled computations. By implementing their method on two simple coupling problems, they finally showed the value of their method compared to the more standard method. If this method is well fitted for their calculations, it is however only implemented for simple problems, less complex than the multiphysics problems that CEDRE

wishes to solve. Moreover, the tests carried out are on problems on an academic scale, and not on an industrial scale. The same issue concerns [4], in which authors used relaxation techniques to transform the original equation into a simpler one. Furthermore, this seems to be specific to a fluid model, and would not scale well in CEDRE where many different models coexist. A similar idea uses splitting of the multiphysics parts to use IMplicit-Explicit methods [5]. It seems well fitted for out multiphysics applications, however we assumed that having a better implicit solver was a priority.

At the same time, tools used in aerodynamic simulation could prove interesting for multiphysics problems. The Jacobian-Free Newton–Krylov method, or JFNK, is already well used in the numerical simulation of the Navier–Stokes equations. In [6], a time integration mechanism is implemented around a JFNK formulation. A physics-based preconditioning is developed to solve the linear problem more precisely. This time integration is then tested on a thermally driven cavity problem. However, this work is only concerned with the Navier–Stokes equations and is not extended to more general energetics problems. Finally, it is only tested on a two-dimensional academic problem. In [7], it was also used on academic problems for the turbulent Navier–Stokes equations. The Jacobian-Free Newton–Krylov method seems to be well fitted to problems that couple multiple physics. It was used in [8] on applications coupling radiative transfer with hydrodynamics. It could prove interesting as a time integration method for CEDRE multiphysics problems.

This thesis was co-funded by ONERA and AID, the French agency for innovation and defence.

AGENCE
INNOVATION
DÉFENSE

General overview and motivations

It can therefore be seen that numerical methods are already available to solve computational fluid dynamics problems. In particular, there are already solvers capable of solving steady multiphysics problems of industrial scales. However, such solver use methods inherited from the aerodynamic context, that are not always adapted to handle the difficulties arising from the multiphysics properties of our problems. It is at least the case for our solver CEDRE. On the other hand, other methods developed in an academic context have shown their interest in simple multiphysics problems. It would now seem interesting to adapt these new methods for the resolution of multiphysics problems in an industrial code. This is the reason for this study. It consists in improving the convergence, speed and robustness of the time integration of the CEDRE platform on steady multiphysics problems by adding numerical methods not used in industrial numerical simulation. The aim of this work is to improve the current performances of our solver by using such more recent methods.

Outline

This document is divided into two parts. The first part, consisting of Chapters 1, 2, and 3, describes the work we did with the solver CEDRE, around the Jacobian-Free Newton–Krylov method. Some of it was presented at the 2022 Aviation Forum [9]. The second part, consisting of Chapters 4 and 5, focuses on exponential time integration methods, initially investigated in CEDRE and then in another solver: JAGUAR. The exponential time integrator shares many ingredients with the JFNK method but is dedicated to unsteady flows. The novelty of the present part is the association of exponential integration and an advanced discretization technique called the Spectral Difference method. We plan to submit an article just after the defense on this research line.

The objective of Chapter 1 is to identify, from the literature, methods that we think can improve CEDRE performances, meaning speed, accuracy and robustness. We will introduce time integration methods and the mathematical tools we need to analyse them. We will go into more detail about implicit time integration methods by describing each step involved in its implementation. Those details will lead us from the differential equation to the algebraic solver used to solve linear systems. This chapter also describes what already existed in CEDRE before this thesis. It shows the strengths and the weaknesses of the traditional time integration methodology. The idea is then to work on those weaknesses by using previously identified methods. Indeed, we will look at the literature and choose new methods to implement that go well with what we want to keep, in place of what we want to discard. At this stage, we will have identified methods that we hope will improve our solver performances.

Chapter 2 aims to present the implementation of the new methods in CEDRE. To do so, we will briefly describe the structure of the solver and its different parts. We will then go into more detail and explain some implementation choices. Finally, we will justify those choices and show they are indeed valid to accomplish what set out to do. After this, we have a method that we need to test to see if it improves our solver performances.

The goal of Chapter 3 is to evaluate the robustness, accuracy and speed of the previously implemented method, and compare those performances to those of previously existing methods. The idea is to select representative test cases, that correspond to typical CEDRE applications. We will present at first simple aerodynamic test cases, and we will increase the complexity of the physical model as we go. For each of those test cases, we will compare the newly implemented method to the older one. This will justify the choices made in Chapter 1 and their implementation in Chapter 2. Finally, we will use the newly implemented method on an also newly implemented fluid model and show that this lead to some benefits for the solver users.

In Chapter 4, we will step out of the context of steady problems and extend our work to unsteady computations that use larger time steps. The objective of this chapter is to introduce a new category of time integration methods and give some of their characteristics. Then, we will show on a simple analytic test case why those new methods may be of interest to our computational fluid dynamics applications. We will also see that we need to use another solver to study properly those methods.

Chapter 5 aims to show the potential of this new class of time integration methods in our field. To do so, we use another solver, as was recommended in the previous chapter. After introducing the novelties of this other solver, we will define a set of methods we want to compare to one another, made of previously existing and newly added methods. Then, we describe several test cases to characterise and compare those methods.

Finally, we will conclude this thesis and give perspectives that can continue this work further.

Part I

Solving efficiently multiphysics steady problems

Chapter 1

Analysis of existing methods

Résumé du chapitre : Analyse des méthodes existantes

Le but de ce chapitre est d'identifier les méthodes qui serviront pour la suite.

Le contexte mathématique et ses notations sont introduits en premier. L'objectif est de donner les étapes permettant de passer d'une équation aux dérivées partielles, issue du modèle physique, à une équation différentielle ordinaire, purement numérique. Pour fermer le système d'équations, les méthodes de discrétisation spatiale sont brièvement introduites et en particulier les deux principales du solveur CEDRE. Grâce à ces méthodes de discrétisation appliquées à un maillage, l'équation différentielle ordinaire finale est donc obtenue et il faut alors l'intégrer avec des méthodes d'intégration temporelle.

La stabilité et la notion d'ordre sont les deux principales caractéristiques nécessaires à de telles méthodes. Puis, les méthodes d'intégration que l'on trouve classiquement dans la littérature sont rappelées, en soulignant la dichotomie explicite/implicite.

Après avoir détaillé les raisons qui nous poussent à analyser les méthodes d'intégration implicite, plusieurs éléments interviennent dans la mise en place : le solveur non-linéaire, le solveur linéaire, et en particulier le passage de l'un à l'autre. Ce passage étant jugé insatisfaisant pour notre solveur, nous présentons la méthode JFNK, ou plus généralement une formulation sans matrice, qui s'appuie sur des parties déjà présentes de CEDRE et corrige, *a priori*, ce défaut.

As we aim to enhance the performances of the solver CEDRE on steady problems, we first have to define some notions. In particular, we need to explain what performance means. In this chapter, we will introduce the general problem, and the tools used to solve it. We will confirm some choices already made in the solver, and identify the features we would like to modify or replace.

1.1 Problem setup

In order to set the mathematical framework for this study, let us start with a partial differential equation arising from the physical model, in the form of

$$\frac{\partial \xi}{\partial t} + \mathrm{F}\left(\xi\right) = 0 \tag{1.1}$$

where the function F uses some space derivatives of the state variable ξ. This equation then describes the temporal evolution of the state variables ξ.

A particular class of such partial differential equations are conservative equations. They correspond to the case where the function F can be written as a divergence term. Finally, with a source term S, those equations look like:

$$\frac{\partial \xi}{\partial t} + \nabla \cdot \underline{\mathrm{f}}\left(\xi\right) = \mathrm{S}\,. \tag{1.2}$$

One might notice that equation (1.2) is indeed a particularisation of equation (1.1), with $\mathrm{F}\left(\xi\right) = \nabla \cdot \underline{\mathrm{f}}\left(\xi\right) - \mathrm{S}$. Those conservative equations are the ones we will focus on in this study, as they describe the physical systems we are interested in.

In this work, attention is paid to computational fluid dynamics and in particular to solving the Navier–Stokes equation and its variants: the reactive Navier–Stokes equation, the Reynold-averaged Navier–Stokes equation, etc. A simple form of this equation can be:

$$\begin{cases} \partial_t\left(\rho\right) & + \nabla \cdot \left(\rho \underline{u}\right) & = 0 \\ \partial_t\left(\rho \underline{u}\right) & + \nabla \cdot \left(\rho \underline{u} \otimes \underline{u} + p \underline{\underline{\mathrm{Id}}}\right) & = \nabla \cdot \underline{\underline{\tau}} \\ \partial_t\left(\rho E\right) & + \nabla \cdot \left(\left(\rho E + p\right) \underline{u}\right) & = \nabla \cdot \left(\underline{\underline{\tau}} \cdot \underline{u} - \underline{q}\right) \end{cases} \tag{1.3}$$

with the closing relation $\rho E = \frac{p}{\gamma - 1} + \rho \frac{\underline{u} \cdot \underline{u}}{2}$. This relation is quite simple, as it is the one that corresponds to ideal gas with constant heat capacity. In the solver CEDRE, this relation is usually more complex as we are working with multifluid flows and with variable heat capacities. The deviatoric stress tensor τ accounts for the fluid's viscosity, and its computation depends on the model used. The heat flux term $\underline{q}$ enables heat exchange in the fluid, through the heat diffusivity, and it is generally related to the temperature gradient. Without those last two terms, one recovers the Euler equations. To this simple form can be added source terms from the reactive model, source terms from the turbulence model, divergence terms from a molecular diffusion model, etc. Yet it is clear that with a bit of rewriting, it is possible to get back to the starting form (1.1) and even the conservative form with source terms (1.2). The quantity ξ is no longer a scalar but a vector with the density ρ, each component of the momentum $\rho \underline{u}$ and the energy ρE as its components. Apart from this small change, the idea is the same.

When solving equations like (1.1) numerically, the domain of interest is first bounded since the infinite domain cannot be represented by a discrete set of variables. Let $\mathcal{D}$ be this computational domain. For numerical computation, the different quantities, such as the state variable ξ, are represented discretely over the domain $\mathcal{D}$ and are stored in the memory of a computer. To do so, the domain $\mathcal{D}$ is split into a finite number of cells, or elements, associated with the degrees of freedom of the solution: this is the mesh. Those cells are small disjoints volumes in 3D, faces in the two-dimensional space or segments in 1D, such as their union recovers the original domain.

Interest quantities, such as the fluid velocity, density, ..., are then stored at each node, averaged at the center of each cell or sometimes in a more complex fashion depending on the method. They are no longer mathematically represented by a function of the continuous physical domain $\xi : \mathcal{D} \to \mathbb{R}$ but by a finite-sized vector Ξ gathering all the information across the discretised domain. For some simple discretisation methods, this vector consists of the quantity evaluated at the mesh nodes or averaged at the center of the cells. For more complex methods, this vector consists of information used to construct the solution over the domain: polynomial coefficients, spectral decomposition coefficients, etc. Anyway, a discretised domain is used rather than $\mathcal{D}$, the continuous one.

The partial differential equation (1.1) transforms then into an ordinary differential equation:

$$\frac{\mathrm{d}\Xi}{\mathrm{d}t} - \mathrm{G}\left(\Xi\right) = 0. \tag{1.4}$$

The difference here is that the function G is a function of a discrete vector whereas F was a function of continuous state variables that are in turn functions of the space variable, and therefore G does not use any spatial derivatives. The negative sign is here so that this new equation is similar to the original partial differential equation (1.1) and the new equation can be written in the form that corresponds to most of the literature:

$$\frac{\mathrm{d}\Xi}{\mathrm{d}t} = \mathrm{G}\left(\Xi\right). \tag{1.5}$$

Thanks to the spatial discretisation method, the only derivative remaining is with regard to time. The rest is then up to the temporal integration method, which is the main topic of this thesis. This integration method will works on equation (1.5) no matter where the function G comes from, but sometimes understanding the origin of this function can help so we will now introduce the spatial discretisation method used in our solver.

1.2 Brief introduction to the spatial integration schemes

A *spatial discretisation method* is what tells how to represent a quantity over a discretised domain, and how to compute the spatial derivative of this quantity from this representation. Indeed, before solving equation (1.1), one must decide how to transform the continuous model into a discretised one. We also have to look at how the spatial derivatives arising from equation (1.1) translate in the discretised model.

1.2.1 The Finite Volume method

The spatial discretisation method used in the fluid solver CHARME is called the Finite Volume method [10, 11]. This method is particularly well fitted for conservative equations such as equation (1.2). A such equation has the property that the quantity ξ is conserved: without source terms, the variation of the total quantity ξ over the domain $\mathcal{D}$ is equal to the flux $f\left(\xi\right)$ coming through the boundary $\partial\mathcal{D}$. In the case of the Navier–Stokes equations (1.3), the density, the momentum and the energy are conserved throughout time, apart from fluxes going through domain boundaries. In a close domain where nothing comes in or out, they are indeed conserved. The main interest of the Finite Volume method is that this property stays true through the spatial discretisation step.

The Finite Volume method consists in integrating the partial differential equation over each cell of the mesh. Writing $\mathcal{V}_i$ the volume of the ith cell:

$$\int_{\mathcal{V}_i} \frac{\partial \xi}{\partial t} \mathrm{d}v + \int_{\mathcal{V}_i} \nabla \cdot \underline{\mathrm{f}}\left(\xi\right) \mathrm{d}v = \int_{\mathcal{V}_i} \mathrm{S}\, \mathrm{d}v. \tag{1.6}$$

Then the Green–Ostrogradski theorem transforms the flux divergence into a surface integral:

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\mathcal{V}_i} \xi \mathrm{d}v + \oint_{\partial \mathcal{V}_i} \underline{\mathrm{f}}\left(\xi\right) \cdot \underline{\mathrm{d}s} = \int_{\mathcal{V}_i} \mathrm{S}\, \mathrm{d}v. \tag{1.7}$$

By writing $\square_i = \frac{1}{\|\mathcal{V}_i\|} \int_{\mathcal{V}_i} \square \mathrm{d}v$ the average in the ith cell, it comes:

$$\frac{\mathrm{d}\xi_i}{\mathrm{d}t} + \frac{1}{\|\mathcal{V}_i\|} \oint_{\partial \mathcal{V}_i} \underline{\mathrm{f}}\left(\xi\right) \cdot \underline{\mathrm{d}s} = \mathrm{S}_i\,. \tag{1.8}$$

As stated before, the spatial discretisation method does transform the partial differential equation into an ordinary differential equation. It tells to store quantities as their averaged values represented at the center of gravity of each cell as the vector Ξ. An important aspect concerns the transformation of the divergence from equation (1.2) into a flux surface average, and the flux computation is still here an open question. The mesh cells are supposed to be polyhedrons and they have a finite number of planar faces. The integral over the boundary of the cell can be decomposed by the faces, to get the approximation:

$$\oint_{\partial \mathcal{V}_i} \underline{\mathrm{f}}\left(\xi\right) \cdot \underline{\mathrm{d}s} \approx \sum_{j \text{ neighbor of } i} \underline{\mathrm{f}}_{ij} \cdot \underline{s_{ij}} \tag{1.9}$$

where $\underline{s_{ij}}$ is the product of the face area and the unit normal vector directed outwards volume i and $\underline{\mathrm{f}}_{ij} \cdot \underline{s_{ij}}$ is an approximation of the flux going through the face between cells i and j. This approximation is a key element of the Finite Volume method and will be discussed later. The function G from equation (1.5) can now be computed: for each face of the mesh, one computes $\underline{\mathrm{f}}_{ij} \cdot \underline{s_{ij}}$, adds this value to the ith component and removes it from the jth component of the new vector. Then, after adding the source terms, this yields the vector containing the result of $\mathrm{G}\left(\Xi\right)$. As can be seen, every contribution of the flux added in a cell is removed from another, and therefore this spatial discretisation method preserves the conservation of the underlying equation.

1.2.2 The Riemann problem

The last remaining problem with this presentation of the Finite Volume method is how to compute the flux going through cell interfaces. On the interfaces between two cells, the left and right quantities ξ_L and ξ_R are known and are used to compute the corresponding flux. It is possible here to use a reconstruction method to get a better approximation of the quantities at the left and right sides of the interface, and therefore the left and right quantities at the face ξ_L^* and ξ_R^* are used, rather than the quantities at the cell center. The idea is now to compute the flux going through the face as a function of ξ_L^*, ξ_R^* and the surface vector $\underline{s}$. From the interface point of view, there are two possible different states, one from each side: this is what is called a Riemann problem. A Riemann problem is an initial value problem applied to a conservation equation, where the initial solution is piecewise constant with a single possible discontinuity. By working with the equation and deriving the jump condition, it is possible to

compute the quantity from a possibly discontinuous state at the interface. Then it is possible to evaluate the flux associated with this state going through the surface. This approach can be called the exact Riemann solver as it uses the exact solution of the Riemann problem. But the drawback of this approach usually is the computational cost required to find this exact solution and people generally revert to approximate Riemann solvers, compromising between speed and accuracy. Several approximate Riemann solvers are available to the user in our solver, such as the well-known Roe, HLLC or AUSM+ schemes [12, 13].

1.2.3 Gradient reconstruction methods

The standard Finite Volume method represents quantities with the averaged value in each cell. This corresponds to a first-order discretisation method. Simply put, it means that it can represent quantities exactly as 0-order polynomials locally to each cell. There are ways to achieve higher-order representation such as with the MUSCL approach [14]. It consists in handling the surface flux evaluation as explained in the Riemann problem part on one hand and deciding what left and right quantities to feed to this flux computation on the other hand. In our solver, there are two ways to construct high-order states to give to the flux computation method. They are described in the following parts. For both of them, the idea is to use neighbouring data to enhance the order of the local representation.

The k-exact method

The first method used to reconstruct high-order quantities is called the *k-exact* method with successive corrections. The idea is to construct iteratively an order k representation of the quantity using the neighbouring order $k-1$ representation [15]. This method is often employed to achieve a second-order reconstruction, as choose most users, but it can also achieve higher-order reconstructions [16, 17, 18]. At each step, while increasing the order of the representation, it is important not to create a local maximum or minimum. This might happen close to discontinuities in the solution, or near rapidly varying spots. It is indeed common, when interpolating, to create local overshoot or undershoot. One might think here about Gibbs or Runge's phenomena, and despite the problem here being a different one, the idea is the same. Creating local extrema in the solution can be troublesome, and so the k-exact method includes limiters to limit reconstructed data.

The Multislope method

The second method used to reconstruct high-order quantities is called the *Multislope* method. This method is a direct adaptation of the structured directional method to unstructured grids. It consists in defining local directions for the interpolation of the unknown on the face. For a given cell and one of its faces, one can draw the line from their centers. Information is interpolated on this line using neighbouring cells to get the local slope. The mechanism to define the slopes and the geometric treatment is quite complex and it won't be discussed here. The details are provided in [19]. Finally, the Multislope method gives a second-order reconstruction. Once again, this reconstruction might create local maxima or minima, and therefore it uses slope limiters [20, 21] to prevent it.

We briefly explained the spatial discretisation method used in our solver, as it might help the analysis of the time integration part. The Finite Volume method averages the partial differential equation over each cell of the mesh, which transforms the flux divergence into a flux surface average. A reconstruction method, the k-exact method or the Multislope method, is then used

to get a higher-order representation of the solution so that the surface flux can be computed at each face. Slope limiters are used to prevent the formation of local extrema, which can be harmful to the computation. The choices made for the spatial discretisation methods are motivated by the fact that CEDRE and in particular the fluid solver CHARME aim to work with general unstructured meshes. It restricts the choices of algorithms, as handling general unstructured meshes can prove difficult for simple spatial discretisation methods. It means that the methods must be more robust, which often means less precise.

1.3 Introduction to time integration methods

With the help of a spatial discretisation method, the equation to solve is now an ordinary differential equation. The main objective of this thesis is focused on the resolution of steady problems. The steady solution of equation (1.5) is given by $\mathrm{G}(\Xi) = 0$. To get the solution, one might then try to find a root of the function G. Unfortunately, with our typical applications, this function G has got bad mathematical properties, such as its stiffness, arising from the nonlinearities of the underlying equations. Therefore, algorithms that try to find a root of G struggle and usually fail. Another approach is to take an initial value Ξ_0, and to solve the equation (1.5) for this initial value. After a long enough time, one hopes that Ξ will reach the desired steady solution.

The idea is now to solve the temporal evolution of Ξ to get the solution after a long time when it approaches the steady solution. The equation is solved numerically, which means the next solution is computed iteratively after a given time step, knowing the current one. It is also possible to modify the equation, as the interest is in the final state, not in the transient one. It is possible, for example, to use local time-stepping, which consists in having each cell of the mesh move forward in time with its own time step. The resulting transient states do not make sense from a physical point of view, as the equation solved is not the initial one, but it converges to the same steady solution. Therefore, it is alright to change the equation as long as it gives the same steady solution. Finally, this way of finding a converged steady solution is what is called a *Pseudo-Transient Continuation* method [22].

After deciding on an initial value, the equation to solve is:

$$\begin{cases} \dfrac{\mathrm{d}\Xi}{\mathrm{d}t} = \mathrm{G}(\Xi) \\ \Xi(t_0) = \Xi_0. \end{cases} \tag{1.10}$$

A time integration method is going to produce a succession of solutions: starting from Ξ_0, it produces Ξ_1 at t_1, then Ξ_2 at t_2, and Ξ_n at t_n, etc. Let Δt_n be the time step $t_{n+1} - t_n$. The following is related to a single step of the time integration method, so the subscript on the time step can be dropped as it is not meaningful. For a steady problem, the evolution of the solution has no interest and it seems reasonable to want to "go fast" to the steady state, meaning to use as large a time step as possible. Unfortunately, not every time integration method allows large time steps due to the well-known Courant–Friedrichs–Lewy condition [23]. Some tools must be defined in order to help decide on the method to use.

1.3.1 Analysis of time integration methods

Consistency and order

A time integration method must respect some properties to be "well-behaved". For instance, it has to be consistent. To define the consistency, let us look at equation (1.10). After one step, a numerical method gives a value Ξ_1, believed to be near the exact value $\Xi(t_0 + \Delta t)$. A numerical time integration method is said to be consistent if:

$$\lim_{\Delta t \to 0} \frac{\Xi_1 - \Xi(t_0 + \Delta t)}{\Delta t} = 0. \tag{1.11}$$

Also, the method is of order p if the local error is in Δt^{p+1} [24]:

$$\Xi_1 - \Xi(t_0 + \Delta t) = O\left(\Delta t^{p+1}\right). \tag{1.12}$$

This means a p-order method can recover exactly a solution that is a polynomial function of time with an order less or equal to p.

Note: The order of a time integration method reflects its "local" behaviour, meaning on a single given time step, provided it is small enough. In the field of spatial discretisation of partial differential equations, the order p of a method is such as:

$$\|\Xi - \Xi_{exact}\| = O(h^p) \tag{1.13}$$

where h is the spatial discretisation parameter. There is a difference between the two definitions: the error order of magnitude is $p + 1$ for the temporal method and p for the spatial one. This is due to the fact that the error in the spatial case is global: it sums the error over the whole domain. It would correspond to counting the error on each step for the temporal integration. To convince oneself, one could say that when solving the differential equation on an interval $[0, T]$ with a fixed T, the global error of a p-order method would behave as $O(\Delta t^p)$ as it amounts to summing $T/\Delta t$ local errors of $O\left(\Delta t^{p+1}\right)$. The coherency with the definition of the order for spatial discretisation methods is now clear. If this trick can help understand the difference between the two definitions, this is indeed just a mental trick and not a rigorous mathematical proof. To get this proof, more hypotheses on the method are required [24].

Stability

A meaningful criterion in the choice of a time integration method is stability. Depending on the application, different levels of stability are expected in order to avoid a numerically induced divergence of the computation.

The stability of a time integration method is usually analysed on the ordinary differential equation with a linear right-hand side [25]. The reason is that if $\tilde{\Xi}$ is solution of equation (1.10), G can be linearised in $\tilde{\Xi}$. With $y = \Xi - \tilde{\Xi}$ and $J = \frac{\partial \mathrm{G}}{\partial \Xi}\left(\tilde{\Xi}\right)$, assumed constant, it comes:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = Jy. \tag{1.14}$$

This new equation used to analyse the stability of time integration methods is the one called the Dahlquist test equation. This equation is studied in $\mathbb{C}$, so that eigenvalues and eigenvectors of the matrix J exist.

Note: When looking at a method applied to the Dahlquist test equation (1.14), it is assumed that the real parts of the eigenvalues of J are all negative. This choice may seem arbitrary but can be understood with the following example. Let us work in $\mathbb{C}^2$, with:

$$J = \begin{pmatrix} -1 & 0 \\ 0 & 10^3 \end{pmatrix}, \quad y_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{1.15}$$

The solution of the equation is then:

$$y(t) = \begin{pmatrix} e^{-t} \\ 0 \end{pmatrix}. \tag{1.16}$$

As the equation is solved numerically, the floating point representation introduces some roundoff error. The initial condition may then be

$$y_0' = \begin{pmatrix} 1 \\ \epsilon \end{pmatrix} \tag{1.17}$$

instead of the exact one y_0, with a typical $\epsilon = 10^{-15}$ for double precision. Let us suppose that we have an exact time integration method that gives the exact solution at each time step $t_n = n\Delta t$. The solution computed by this method will be:

$$y_n = \begin{pmatrix} e^{-n\Delta t} \\ \epsilon e^{10^3 n\Delta t} \end{pmatrix} \tag{1.18}$$

that gives an error of $\epsilon e^{10^3 n\Delta t}$. For the numerical values suggested here, this amount to an error as large as 10^6 for $n = 5$ and 10^{28} for $n = 10$. The explosion of the error comes from the fact that the positive eigenvalue of J amplifies the roundoff error. This phenomenon has nothing to do with the time integration scheme, but with the equation. Finally, this is why equation (1.14) is studied assuming the eigenvalues of J are negative.

Single-step methods To compute the solution at the next time step, some methods need only to know the current solution. Such methods are called single-step methods. A single-step method applied to the Dahlquist test equation (1.14) leads to the following relationship between the current solution and the next one:

$$y_{n+1} = g(\Delta t J)\, y_n. \tag{1.19}$$

For most time integration methods, g is an analytic function. The initial value y_0 can be decomposed on a basis of eigenvectors of J, $v_1, \ldots, v_N$, associated with the eigenvalues $\alpha_1, \ldots, \alpha_N$: $y_0 = \sum_{i=1}^{N} \lambda_i v_i$. Because g is an analytic function, it comes immediately:

$$y_n = \sum_{i=1}^{N} \lambda_i g(\Delta t \alpha_i)^n v_i. \tag{1.20}$$

It is now straightforward to deduce a stability condition for the single-step method: if for any i there is $|g(\Delta t \alpha_i)| < 1$, then y_n converges to 0. This is how the stability region of a time integration method is defined:

$$\{ z \in \mathbb{C} \mid |g(z)| < 1 \}. \tag{1.21}$$

When each eigenvalue of $\Delta t J$ falls in the stability region, then the method is stable.

If an eigenvalue is not in the stability region, the associated eigendirection will be amplified and a numerical instability will lead to the divergence of the computation. We can see that the argument of the function g is not J but $\Delta t J$. This means that stability can be achieved by choosing wisely the time step: with a small enough Δt one can ensure that each eigenvalue falls into the stability region. Unfortunately, this often forces the user to set a relatively small time step, which is an issue for steady computations.

Multistep methods Some methods do not fall into the previous framework. The multistep methods, in particular, cannot be written under the form of equation (1.19). These methods use not only y_n to find y_{n+1}, but the k previous steps. Applied to equation (1.14), they can be written in the form:

$$y_{n+1} = \sum_{i=1}^{k} g_{k-i}\left(\Delta t J\right) y_{n+1-i}. \tag{1.22}$$

When looking for y_i under the form $y_i \propto \mu^i$, one has:

$$\mu^k = \sum_{i=1}^{k} g_{k-i}\left(\Delta t J\right) \mu^{k-i} \tag{1.23}$$

which leads to identifying the polynomial $g_{\Delta t J}(\mu) = \mu^k - \sum_{i=0}^{k-1} g_i\left(\Delta t J\right) \mu^i$. If each root of this polynomial is of modulus less than 1, the solution converges to 0. This is how the stability region for multistep methods is defined [25]:

$$\left\{ z \in \mathbb{C} \;\middle|\; \begin{array}{l} \text{all roots of } X^k - \sum_{i=0}^{k-1} g_i\left(z\right) X^i \text{ are of modulus less or} \\ \text{equal to 1, strictly less to 1 for roots with multiplicity} \end{array} \right\}. \tag{1.24}$$

Once again, if all eigenvalue of $\Delta t J$ falls in the stability region, then the method is stable.

The key property resulting in the stability analysis of time integration methods is the *A-stability* [26]. A time integration method is A-stable if its stability region contains the left half complex plane. Simply put, a method is A-stable if it converges to 0 when it should, and does not diverge due to numerical errors. The A-stability is interesting, as an A-stable method is also said to be unconditionally stable, whereas a method that is not is conditionally stable. This other characterisation comes from the fact that a non-A-stable method needs to respect some additional criteria to be stable, on the time step it uses for example, whereas an A-stable method is stable no matter the time step. As we said before, we would like to use large time steps to quickly find the steady state of our applications, and that is why we look for A-stability in our methods.

Having defined the concepts of stability and order of time integration methods, we are now equipped to analyse them. We see from the literature that time integration methods are usually split into two groups: explicit and implicit methods. We can now look at methods from this literature, starting with the simpler explicit methods.

1.3.2 Explicit methods

Explicit methods are called this way because, at each step, the computation of the next solution is straightforward: they give it explicitly as a function of currently available data. They are largely used in unsteady computational fluid dynamics simulations. Their strength comes from the fact that they are usually simple and therefore easy to implement in a solver, and computationally inexpensive compared to non-explicit methods.

Explicit Euler method

The explicit Euler method is the most simple time integration method. It consists in integrating equation (1.10) between t_n and t_{n+1} assuming the function G stays constant, equal to $\mathrm{G}(\Xi_n)$. Equivalently, it consists in replacing the time derivative $\frac{\mathrm{d}\Xi}{\mathrm{d}t}$ by a finite difference $\frac{\Xi_{n+1}-\Xi_n}{\Delta t}$ and evaluating G in Ξ_n. Then, the method gives:

$$\Xi_{n+1} = \Xi_n + \Delta t\, \mathrm{G}(\Xi_n)\,. \tag{1.25}$$

After verifying that this is a first-order method, a quick stability analysis gives a stability region equal to the open unity disk centred in -1. Practically, this stability region is often deemed unsatisfactory as it forces the use of small time steps. Yet this method is a classic that must be introduced before talking about more complex methods.

Runge–Kutta methods

Instead of making one step forward in time as the explicit Euler method, a Runge–Kutta method will make a set of intermediate steps, and find the final solution as a combination of those intermediate steps. It starts from the value of Ξ_n known in t_n and given intermediates steps $t_{n,i} = t_n + c_i\Delta t$ for $1 \leq i \leq k$, with a fixed k. An exact integration between t_n and $t_{n,i}$ of equation (1.10) leads to:

$$\Xi(t_{n,i}) = \Xi_n + \Delta t \int_{t_n}^{t_{n,i}} \mathrm{G}(\Xi(t))\,\mathrm{d}t. \tag{1.26}$$

The integral on the right-hand side is then approximated with a quadrature using the previously computed intermediate steps:

$$\int_{t_n}^{tn,i} \mathrm{G}(\Xi(t))\,\mathrm{d}t \approx \sum_{j=1}^{i-1} a_{ij}\,\mathrm{G}(\Xi(t_{n,j}))\,. \tag{1.27}$$

Once each intermediate step is known, equation (1.10) is integrated between t_n and t_{n+1} and the resulting integral is approximated using the intermediate steps.

To sum up, a Runge–Kutta method iterates in the following way:

$$\begin{cases} \Xi_{n+1} = \Xi_n + \Delta t \sum_{i=1}^{k} b_i\,\mathrm{G}(\Xi_{n,i}) \\ \text{with} \quad \Xi_{n,i} = \Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij}\,\mathrm{G}(\Xi_{n,j})\,. \end{cases} \tag{1.28}$$

0	
	1

RK1

0		
1/2	1/2	
	0	1

RK2

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

RK4

Table 1.1: Butcher tableau for the explicit Euler, Midpoint and RK4 methods.

A Runge–Kutta method is characterised by its size k and by the quadrature coefficients: $a_{ij,1\leq j<i\leq k}$, $b_{i,1\leq i\leq k}$ and $c_{i,1\leq i\leq k}$. There are as many Runge–Kutta methods as there are choices in the quadrature coefficients, but not all choices give good methods. There are criteria that the coefficients must follow to ensure the consistency of the method, and then criteria with more complexity as the order increases. The quadrature coefficients are often arranged in the Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \quad = \quad \begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_k & a_{k1} & a_{k2} & \dots & a_{k,k-1} \\ \hline & b_1 & b_2 & \dots & b_{k-1} \quad b_k \end{array} . \tag{1.29}$$

The Butcher tableau of some well-known Runge–Kutta methods are shown in table 1.1. The RK1 method is in fact equivalent to the explicit Euler method. The RK2 method is one of the 2 steps second-order Runge–Kutta method. This one is also called the Midpoint method. The RK4 method is a fourth-order method with 4 steps. This is the most famous Runge–Kutta method, vastly used for explicit time integration of ordinary differential equations.

We note that the coefficients c_i do not appear in the definition from equation (1.28). It is because the differential equation (1.5) is autonomous: the function G does not depend explicitly on the time. If it did, the equation would then be non-autonomous, and G would be a function of two variables: the state vector Ξ and the time t. Equation (1.28) becomes then:

$$\begin{cases} \Xi_{n+1} = \Xi_n + \Delta t \sum_{i=1}^{k} b_i \, \mathrm{G}\left(\Xi_{n,i} \ , \ t_n + c_i \Delta t\right) \\ \text{with} \quad \Xi_{n,i} = \Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \, \mathrm{G}\left(\Xi_{n,j} \ , \ t_n + c_j \Delta t\right). \end{cases} \tag{1.30}$$

It can be shown that the order p of the method is less or equal to the number of steps k. Up to 4 steps, it is possible to choose the quadrature coefficients to have $p = k$. Above that, getting a lower bound on the order depending on the number of steps is still an open problem as of today. For a Runge–Kutta method of order p, the corresponding function used for the stability analysis is [25]:

$$g(z) = 1 + z + \frac{z^2}{2} + \cdots + \frac{z^p}{p!} + O\left(z^{p+1}\right). \tag{1.31}$$

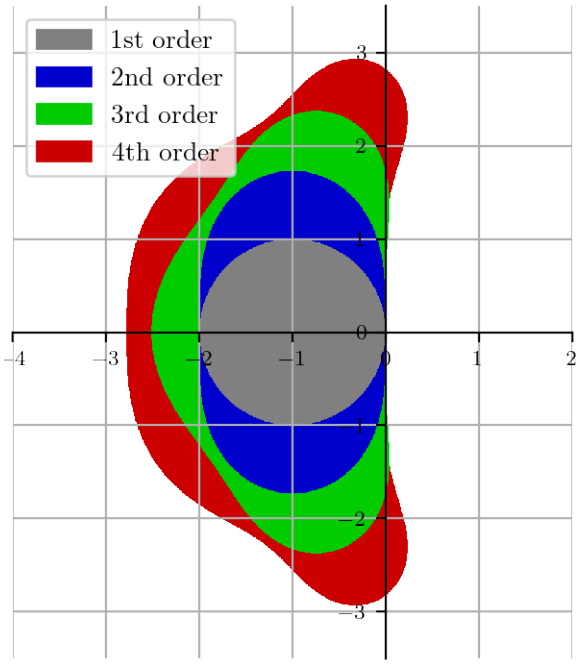


Figure 1.1: Stability regions (in colour) of the first fourth-orders Runge–Kutta methods.

When $p = k$, the last term $O\left(z^{p+1}\right)$ is in fact null. Figure 1.1 shows the stability region of the Runge–Kutta methods of orders up to 4.

We can see that Runge–Kutta methods are not A-stable, and they do not have a large stability region. Increasing the order of the method does increase the stability region, but not enough for our applications. Practically, this imposes the use of small time steps, which is agreeable for unsteady computations but not for our steady ones. If one iteration of the method is inexpensive, the total number of iterations needed to reach the steady solution will make the overall computation too costly.

Adams–Bashforth methods

We could try to use other explicit methods, such as multistep Adams–Bashforth methods. The kth order Adams–Bashforth method uses the last k computed steps to find the next one. One can indeed check that the index k designating the method also corresponds to its order [27].

The idea is to apply a Lagrange interpolation of the function G from equation (1.10) in the k last computed points, and then replace G with the interpolation polynomial when integrating from t_n to t_{n+1}. Contrary to the Runge–Kutta methods, as this method reuses previous information, a single G evaluation is required at each step. Because of that, the cost of one iteration of the Adams–Bashforth method is quite inexpensive. The stability analysis for such methods is a bit more complex [27, 25], and so the result obtained numerically is shown in figure 1.2 without giving the details. The conclusion is even worse than with Runge–Kutta methods, as the stability region decreases as the order increases. The Adams–Bashforth methods can reach

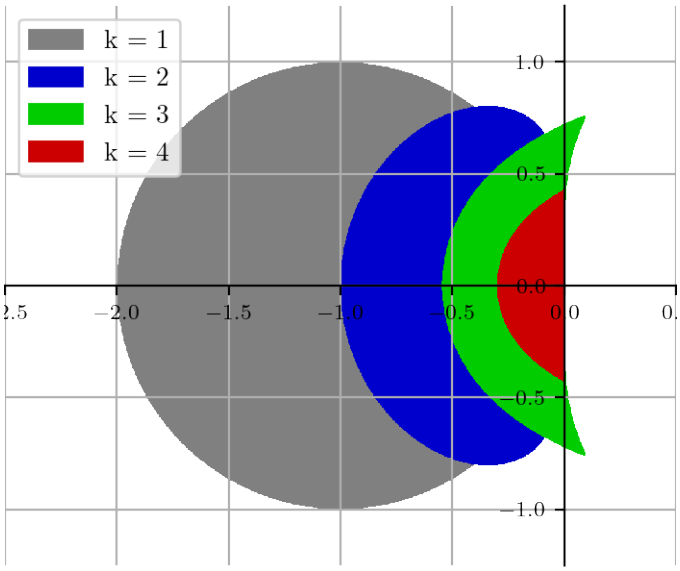


Figure 1.2: Stability regions (in colour) of the first four Adams–Bashforth methods.

a high order of accuracy while staying computationally inexpensive, but they drastically lack stability, and that is why they are often not used in computational fluid dynamics computations. More generally, an explicit multistep method cannot be A-stable [26].

1.3.3 Implicit methods

We explained in the previous section why explicit time integration methods are not suited for our applications. It is then natural to look at implicit methods. Contrary to the explicit methods, implicit methods do not give the looked-for solution right away, but as the solution of a specific equation. The name is appropriate: the next state is not given explicitly but implicitly.

Implicit Euler method

The implicit Euler method is the equivalent of the explicit Euler method but on the implicit side. It is quite similar, as it consists in integrating equation (1.10) assuming the function G is constant, but this time equal to $\mathrm{G}(\Xi_{n+1})$:

$$\begin{aligned} \Xi_{n+1} &= \Xi_n + \Delta t\, \mathrm{G}(\Xi_{n+1}) \\ \Leftrightarrow\ \Xi_{n+1} - \Xi_n - \Delta t\, \mathrm{G}(\Xi_{n+1}) &= 0 \end{aligned} \tag{1.32}$$

and the next state Ξ_{n+1} is given as the solution of a nonlinear problem, the root of a nonlinear function.

After checking that this is a first-order method, the stability analysis gives the corresponding function $g(z) = (1-z)^{-1}$, which gives a stability region equal to the whole complex plane minus the closed unity disk centered in 1. Therefore this method is A-stable.

Implicit Runge–Kutta methods

Runge–Kutta methods can also be implicit methods. This happens when the quadratures use points that have not already been computed. In other words, this corresponds to a full A matrix in the Butcher tableau, where it is strictly lower triangular for explicit Runge–Kutta methods. This also means that any step of the Runge–Kutta method may be used in any other step, and therefore one may have to simultaneously solve an implicit system of equations. This may lead to awfully expensive methods, and therefore users tend to restrict themselves to some particular methods.

Despite their cost, implicit Runge–Kutta methods can be appealing. The main reason is that they can easily achieve a high order of accuracy. For example, the methods based on Gauss–Legendre quadratures achieve an order $2k$ with k steps, and they are all A-stable [24]. Theoretically, this means that an arbitrarily high order can be achieved while keeping the stability quality with these methods. However, users usually stop at the 3-stage 6th order method, as the computational cost tends to be too much for higher order methods.

When applying the stability analysis to a Runge–Kutta method, one can get the function g using the matrix A and the array b:

$$g(z) = 1 + zb^T (\mathrm{Id} - zA)^{-1} (1, \dots, 1)^T. \tag{1.33}$$

We will not try to show the corresponding stability regions as there are too many methods possible. Instead, we will review some of the most frequent from the literature.

Diagonally Implicit Runge–Kutta methods The Diagonally Implicit Runge–Kutta methods [28], or DIRK methods, are Runge–Kutta methods with a lower triangular matrix A. Then, each step is given as an implicit problem using the already known steps and the next one. The difference is that instead of solving a full implicit system, one just needs to solve each implicit step successively. This help drastically reduces the cost of the method. Furthermore, if all quadrature coefficient a_{ii} are all equals, as each step requires the inversion of the matrix

$$\mathrm{Id} - a_{ii} \Delta t \frac{\mathrm{d}\,\mathrm{G}}{\mathrm{d}\Xi} (\Xi_n), \tag{1.34}$$

this can help solve the linear problem. This variant is called Singly Diagonally Implicit Runge–Kutta methods (SDIRK) [25].

Rosenbrock methods Rosenbrock methods are also called linearly implicit Runge–Kutta methods [29]. They start from the recurrence relation of a DIRK method:

$$\Xi_{n,i} = \Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathrm{G}(\Xi_{n,j}) + \Delta t a_{ii} \mathrm{G}(\Xi_{n,i}) \tag{1.35}$$

and then take its image by G. Instead of solving this nonlinear equation with a nonlinear solver as the DIRK method would do, they linearize the result:

$$\mathrm{G}_{n,i} = \mathrm{G}\left(\Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathrm{G}_{n,j}\right) + \Delta t a_{ii} \frac{\mathrm{d}\,\mathrm{G}}{\mathrm{d}\Xi} \left(\Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathrm{G}_{n,j}\right) \mathrm{G}_{n,i} \tag{1.36}$$

by noting $\mathrm{G}_{n,j} = \mathrm{G}\left(\Xi_{n,i}\right)$. The recurrence equation for the integration scheme is no longer on the intermediate states but on the intermediate slopes. Usually, the Jacobian matrix is kept constant throughout the stages, equal to the Jacobian matrix evaluated in Ξ_n. Additional parameters γ_{ij} are often to add more flexibility to the method [30]. It gives the Rosenbrock–Wanner method:

$$\begin{cases} \Xi_{n+1} = \Xi_n + \Delta t \sum_{i=1}^{k} b_i \, \mathrm{G}_{n,i} \\ \text{with} \quad \mathrm{G}_{n,i} = \mathrm{G}\left(\Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \, \mathrm{G}_{n,j}\right) + \Delta t \dfrac{\mathrm{d}\,\mathrm{G}}{\mathrm{d}\Xi}\left(\Xi_n\right) \sum_{j=1}^{i} \gamma_{ij} \, \mathrm{G}_{n,j} \, . \end{cases} \tag{1.37}$$

This corresponds to classical Rosenbrock methods when all γ_{ij} are null, except for γ_{ii} that correspond to the a_{ii}. However, the distinction is not made very often, and Rosenbrock–Wanner methods are simply called Rosenbrock methods. The method is defined by the coefficients $a_{ij,1\leq j<i\leq k}$, $b_{i,1\leq i\leq k}$ and $\gamma_{ij,1\leq j\leq i\leq k}$. If the problem is non-autonomous, the method is then:

$$\begin{cases} \Xi_{n+1} = \Xi_n + \Delta t \sum_{i=1}^{k} b_i \, \mathrm{G}_{n,i} \\ \text{with} \quad \mathrm{G}_{n,i} = \mathrm{G}\left(\Xi_n + \Delta t \sum_{j=1}^{i-1} a_{ij} \, \mathrm{G}_{n,j} \ , \ t_n + c_i \Delta t\right) \\ \qquad\qquad + \Delta t \dfrac{\partial\,\mathrm{G}}{\partial \Xi}\left(\Xi_n, t_n\right) \sum_{j=1}^{i} \gamma_{ij} \, \mathrm{G}_{n,j} \\ \qquad\qquad + \Delta t \dfrac{\partial\,\mathrm{G}}{\partial t}\left(\Xi_n, t_n\right) \sum_{j=1}^{i} \gamma_{ij} \end{cases} \tag{1.38}$$

with $c_i = \sum_{j=1}^{i-1} a_{ij}$ as for other Runge–Kutta methods. If one takes all the γ_{ii} equals to one another, then the matrix to invert at each step of the Rosenbrock method is the same. This can help reduce the cost of the method by doing a single matrix inversion, or by recycling information between Krylov subspace methods [31]. Overall, even if they can achieve high-order and A-stability, Rosenbrock methods are still computationally expensive.

Backward differentiation formula

As explicit Runge–Kutta methods are extended to implicit methods, Adams–Bashforth methods also have their implicit counterparts. The idea is to still compute the Lagrange interpolation of the function G but with one additional point: the point at the next time step. This new method is called the Adams–Moulton method. However, the stability region of Adams–Moulton is quite narrow, as they were originally not made for stiff equations [24]. This is why the Backward Differentiation Formula methods, or BDF methods, were introduced. Contrary to Adams–Bashforth and Adams–Moulton methods, they use the Lagrange interpolation of the solution Ξ instead of the interpolation of the function G. Then, the time derivative of the solution can be replaced by the time derivative of the interpolating polynomial, and the resulting equality is evaluated at the next time step t_{n+1}. This gives an implicit equation that is solved to get Ξ_{n+1}. The name of those methods comes from the fact that if it uses a constant time step between iterations, this equation can be written using the differentiating operator defined by $\nabla^0 \square_i = \square_i$ and

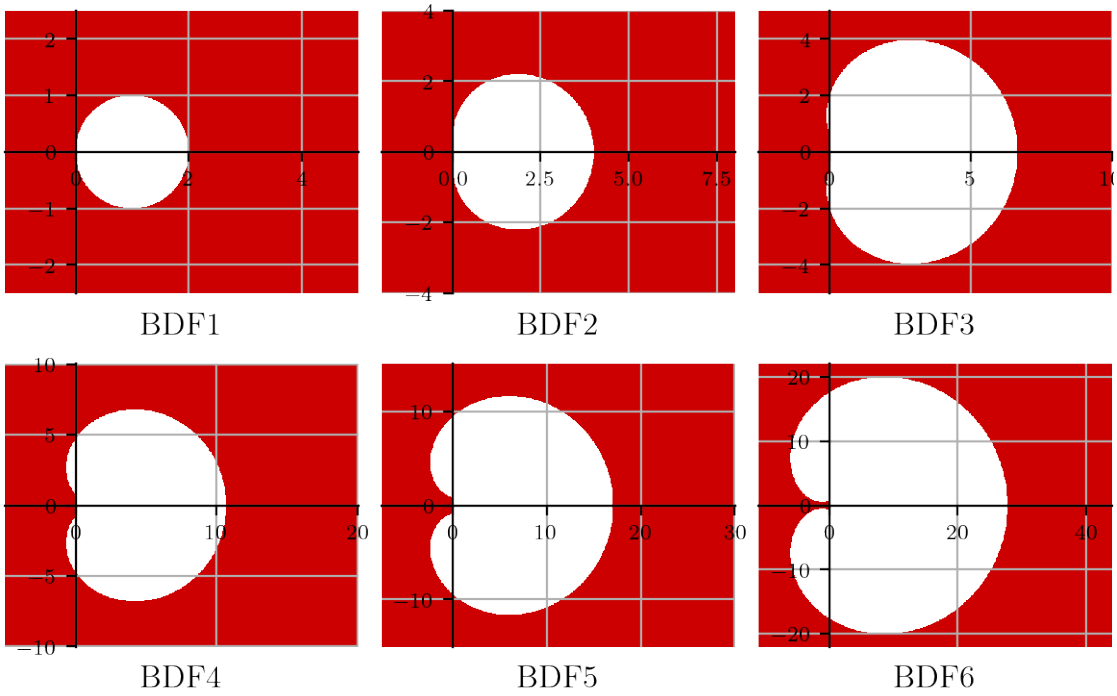


Figure 1.3: Stability regions (in colour) of the BDF methods. Zoom is not constant between plots.

$\nabla^{j+1}\square_i = \nabla^j\square_i - \nabla^j\square_{i-1}$:

$$\sum_{i=1}^{k} \frac{1}{i}\nabla^i \Xi_{n+1} = \Delta t\, \mathrm{G}\left(\Xi_{n+1}\right). \tag{1.39}$$

The order of the method is equal to the index of the method, corresponding to the number of previous states needed to compute the next one. These methods allow for an arbitrarily high order without increasing the cost, as the implicit equation is not harder to solve as the order increases. However, the stability analysis limits the higher order achievable. The stability analysis of the BDF methods can be done numerically. At this stage, one can note that the first-order BDF method is in fact the implicit Euler method. Also, methods of order 7 or higher are unstable, so we can limit our analysis to methods with orders from 1 to 6. The corresponding stability regions are satisfying, as can be seen in figure 1.3. Particularly, the first and second methods are A-stable. We see when looking at the scales in figure 1.3 that the complement of the stability region grows with the index of the method. More generally, there are no A-stable multistep methods with orders higher than 2 [26, 25].

This introduction to the classic time integration methods helps us to decide what to do for our solver. As we said earlier, we want to solve an ordinary differential equation in order to recover the steady solution, reached after a long time. Then, explicit methods that constrain the time step are not well fitted. Even if they cost more, algorithmically speaking, implicit methods are indeed the best choice for stiff equations. It is often better to do a single expensive iteration over a large time step with an implicit method than make a lot of inexpensive iterations over

small time steps with an explicit method. This is finally why we will continue working with the A-stable implicit Euler method. This method is already used in our solver as the base implicit method, for the same reasons we want to use it. Furthermore, it is at the base of all other implicit methods, as they can be seen as small variations of the implicit Euler methods. Choosing it is also smart as updating it into another implicit time integration method will not require too much work.

1.4 Implicit methods framework

1.4.1 Methodology of the implicit time integration

As we explained in the previous section, implicit time integration methods give the next state as the solution of a nonlinear equation or a system of nonlinear equations. If we want to use implicit methods, we then need to be able to solve a nonlinear problem. We will continue our discussion using the implicit Euler method, but everything can easily be adapted for any other implicit method. For Diagonally Implicit Runge–Kutta methods, we apply the nonlinear solver successively for each step. For BDF methods, we apply the nonlinear solver to a slightly different equation.

From a nonlinear problem ...

Newton's method can be used to solve a nonlinear problem of the form:

$$\tilde{f}\left(\Xi_{n+1}\right) = 0. \tag{1.40}$$

Equivalently, as it works for a single n at a time, this equation can be expressed in terms of the increment $x = \Xi_{n+1} - \Xi_n$ without writing the subscript:

$$f\left(x\right) = 0. \tag{1.41}$$

Newton's method starts from an initial guess x_0. This initial guess is often equal to zero, as it is equivalent to taking Ξ_n as an initial guess for Ξ_{n+1}. The method will then iterate to approximate the solution of equation (1.41).

At each step of Newton's method, the problem is linearised in the current estimation x_i and the new equation is evaluated at the next iteration x_{i+1}:

$$f\left(x_{i+1}\right) \approx f\left(x_i\right) + f'\left(x_i\right)\left(x_{i+1} - x_i\right) = 0 \tag{1.42}$$

This gives a linear problem in which x_{i+1} is the solution.

... to a linear one

Equation (1.42) can be rewritten into the classic linear problem:

$$Ax = b \tag{1.43}$$

where $A = f'\left(x_i\right)$, $b = -f\left(x_i\right)$ and $x = \left(x_{i+1} - x_i\right)$. The x notation is reused as it is often used in the literature to name the unknown in linear problems. Such linear problems are found during the iterations of Newton's method, but each is solved for a given iteration number so there is no ambiguity in this notation. A lot of methods were conceived to solve such linear problems, and we will explain later how to handle them.

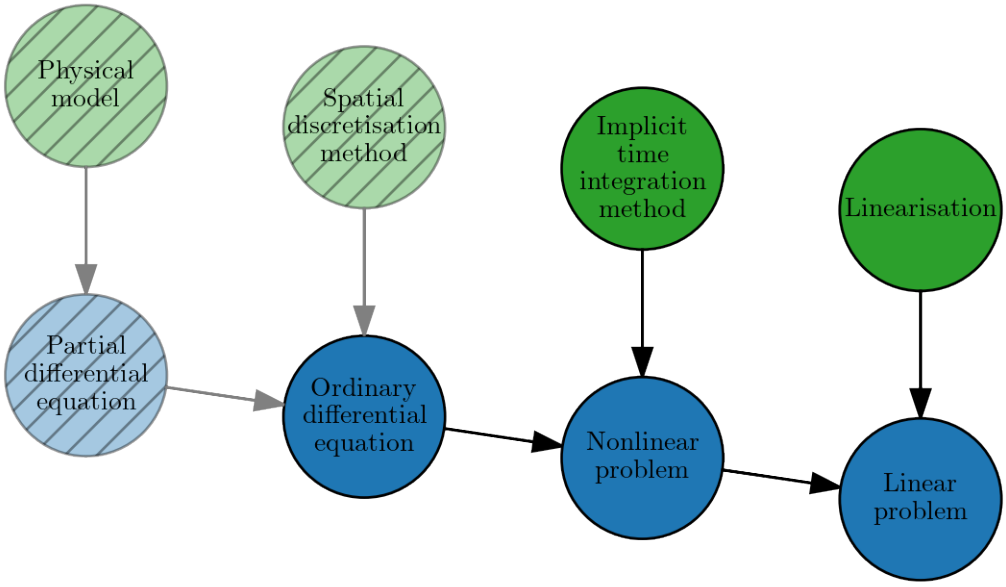


Figure 1.4: Procedure used to find the solution of steady problems.

To sum up, we started from the partial differential equation (1.1) arising from the physical model. With a spatial discretisation method, and after choosing an initial value, this transforms into an ordinary differential equation (1.10). For stability reasons, we decided to use implicit time integration methods. Iteratively, such methods are going to produce one or several nonlinear problems in the form of equation (1.41). Newton's method used to solve such problems is going to produce a succession of linear problems in the form of equation (1.43). This complicated sequence of operations describes the time integration procedure used in our solver to find the solution to steady problems. It is schematised in figure 1.4, where the blue circles correspond to the problems being solved, while green circles correspond to the methods used to solve them. In this thesis, we are interested in the non-hatched parts.

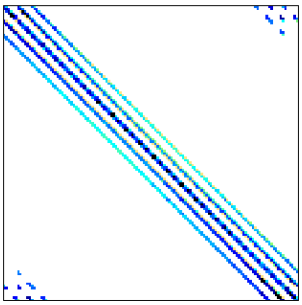
1.4.2 Nonlinear solver

As we said, we solve the nonlinear problem with Newton's method. What was done in our solver was in fact a single step of Newton's method, which means a single linearization of the nonlinear problem. Since we could benefit from an actual Newton's method, it was implemented. Contrary to the single linearisation that was done before, several linear problems need to be solved during one single time step, so several Jacobian matrices are needed.

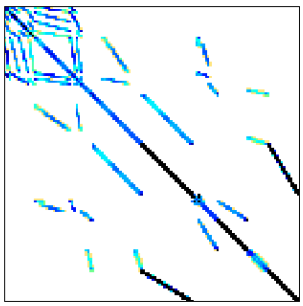
The issue when using Newton's method is that it should be accompanied by a Line Search algorithm to determine the step size α so that the next iterate of the method is:

$$x_{i+1} = x_i - \alpha f'(x_i)^{-1} f(x_i). \tag{1.44}$$

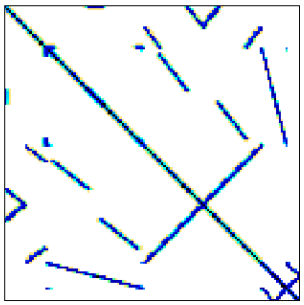
During this thesis, we did some work towards using a complete Newton's method, but we did not have time to develop a Line Search algorithm, so we ended up using the standard method



(a) GT01R: 2D inviscid flow in the inter-blade channel of a linear cascade turbine.



(b) HV15R: 3D RANS simulation of an engine fan.



(c) RM07R: 3D viscous flow in a jet engine compressor.

Figure 1.5: Matrices from various CFD simulations [33] using a Finite Volume method. Coloured points correspond to nonzero values.

from our solver: a single linearization of the nonlinear problem.

1.4.3 Linear solver

We want to be able to solve efficiently linear systems like (1.43). We additionally assume that A is an invertible matrix. Let us note the size of the linear system N. This size is quite large in our typical applications, but the linear system is sparse. This means that the coefficients of the matrix A are mostly zeros. This is because a coefficient in the matrix A corresponds to a link between two degrees of freedom. For the considered spatial discretisation methods, a cell only depends on a small number of neighbouring cells, and therefore a degree of freedom is not linked with most of the others. In practice, the stencil is generally reduced to the current cell and those that share a face with the current cell. This transcribes into lots of zeros in the matrix, and therefore its sparsity. Such sparse matrices can be seen in figure 1.5. Using the sparsity of the matrix is essential, as it would not be possible to store it as a dense matrix in the memory of a computer. Instead, some clever formats are used, such as the Compressed Sparse Row format [32]. Some operations such as matrix-vector products are more efficiently done using such formats.

Many methods exist to solve linear problems. Some are even taught in school, such as the Gaussian elimination. Such methods are called direct methods, as they first do some work and then find directly the exact solution to the problem. But for problems with huge sizes such as the ones encountered in computational fluid dynamics, the amount of work is too much to be computed with today's means. It would take too much time as well as too much memory. Instead, one can use iterative methods [34]. An iterative method starts from an initial guess x_0 and produces as it iterates a supposedly better estimation of the solution x_n. The subscript n has nothing to do with the subscript used to identify the iterates of the time integration method: this happens at a given fixed time step. We can then decide when to stop the method, whether the solution estimate is good enough or the resolution is taking too much time. Iterative methods are the most well-fitted to solve large sparse linear problems, and they are the preferred solution in computational fluid dynamics.

Among the iterative methods are what could be called the "classic" methods, or relaxation. They are the Jacobi, Gauss–Seidel or Successive Over Relaxation methods. They decompose the matrix into $A = M - N$ with M a matrix easily invertible. The choice of M depends on the method. Then, starting from a given x_0, each step computes $x_{n+1} = M^{-1}(Nx_n + b)$. Those methods are often deemed not efficient enough for computational fluid dynamics applications. They are often used, however, as preconditioning methods, as we will see later.

Another class of iterative methods is becoming the standard for computational fluid dynamics: the Krylov subspace methods. A Krylov subspace method projects the linear problem (1.43) on a smaller linear subspace called Krylov subspace. The obtained smaller system is then solved, much more easily. The cleverness resides in the fact that those Krylov subspaces are nested, and each iteration reuses the information obtained in the previous ones.

In the following, we note at iteration n the residual $r_n = b - Ax_n$. Practically, we keep $n \ll N$ so that the cost of the method stays reasonable, but this does not change the following. The corresponding Krylov subspaces are defined as:

$$\mathcal{K}_n(A, r_0) = \text{Vect}\left(r_0, Ar_0, \ldots, A^{n-1}r_0\right). \tag{1.45}$$

Then, the next iterate in $\mathcal{K}_n$ must satisfy a Petrov–Galerkin condition [35]:

$$x_n \in x_0 + \mathcal{K}_n(A, r_0) \quad \text{such as} \quad r_n \perp \mathcal{L}_n \tag{1.46}$$

where $\mathcal{L}_n$ is a linear subspace with dimension n. For example, $\mathcal{L}_n = \mathcal{K}_n$ corresponds to a Galerkin condition, and $\mathcal{L}_n = A\mathcal{K}_n$ is a minimum residual condition.

To construct the growing Krylov subspace, one can use the Arnoldi iteration [36]. As a result, the matrix A is only used through matrix-vector products. Indeed, Krylov subspace methods do not require the matrix A to solve the linear system (1.43), just to know how to compute matrix-vector products. We will use this property later on.

There are many Krylov subspace methods that can solve a linear problem, characterised by the choice of $\mathcal{L}_n$. The minimal residual condition gives the Generalized Minimal Residual method or GMRES [37], vastly used in computational fluid dynamics [38] and other fields of numerical simulations [39, 40]. Its main advantage is that even if its iterations may be slightly more expensive than other methods such as the Bi-CGSTAB method [41, 36], the residual norm is minimised and the error is then decreasing as the method iterates. This allows for some control of the residual norm. With the Bi-CGSTAB method for example, also available in our solver, the residual norms do not have to be decreasing, which can lead to chaotic convergence. The GMRES method already exists in our solver and is often used. Because it is discussed a lot in the literature, many variants and enhancements were developed [42, 43, 44]. For those reasons, we decided to keep the GMRES method as the base of our linear solver.

The convergence of Krylov subspace methods, and more generally of iterative methods, depends on the linear system matrix. It is common knowledge that the convergence is linked to the condition number of the matrix [45]. The condition number of an invertible matrix A is defined as:

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| \tag{1.47}$$

and so it depends on the chosen norm. A problem is said to be well-conditioned when $\kappa = O(1)$, and ill-conditioned when $\kappa \gg 1$. For the Euclidean norm $\|\cdot\|_2$,

$$\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}} \tag{1.48}$$

where $\sigma_{\min}$ and $\sigma_{\max}$ are the smallest and largest singular values of the matrix A. As a reminder, the singular values of A are the eigenvalues of A^*A where A^* notes the conjugate transpose of A. The relation (1.48) is often simplified as

$$\kappa(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|} \tag{1.49}$$

with $\lambda_{\min}$ and $\lambda_{\max}$ the eigenvalue of A with the smallest and largest modulus. If this helps picture what the condition number stands for, this is only true for normal matrices: matrices that commute with their conjugate transpose. However, the matrices we face in our field have no reasons to be normal matrices. An intriguing result is found in [46]. For a given decreasing convergence curve and a given spectrum, it is possible to construct a linear problem such as the convergence of GMRES follows the given curve and the matrix has the given spectrum. In other words, there are ill-conditioned matrices for which GMRES converges quickly and well-conditioned matrices for which GMRES will be slow. In particular, one can find a matrix with the best condition number possible, meaning 1, on which GMRES will make no progress until the last iteration. This is because of the nonnormality of the matrix [47, 46]. To handle it in the analysis of the convergence, one must not look at the spectrum but the pseudospectrum [48, 49]. This analysis is quite complex and is mostly done in an analytical context, not an industrial one. Using Krylov subspace methods with nonnormal matrices and the study of their convergence is no simple task [50, 51]. As the matrices we encounter in our computational fluid dynamics problems are arbitrary, meaning mostly nonnormal, we decided not to study finely the spectrum of the operator.

Even if we will not look deeply into the spectrum of the matrix, we still help the linear solver with preconditioning. Preconditioning consists in multiplying the equation (1.43) with a preconditioning operator P to the left for left preconditioning:

$$PAx = Pb \tag{1.50}$$

and to the right for right preconditioning:

$$\begin{cases} APx' = b \\ Px' = x. \end{cases} \tag{1.51}$$

The idea is to transform A into a matrix that is more easily invertible. The matrix to invert is now PA for left preconditioning and AP for right preconditioning. If the preconditioning operator is close to A^{-1} and is cheap to compute, this new matrix is close to Id and so is easily invertible. If P is too close to A^{-1}, computing the preconditioning will be as expensive as solving the original linear problem. If P is too easy to compute, meaning close to Id, the preconditioning will be useless. Choosing a preconditioner means making a compromise between those two extrema.

Just as Krylov subspace methods only need to compute matrix-vector products, and not to know the matrix, we do not need to know the preconditioning matrix, but only to know how to apply it on any given vector.

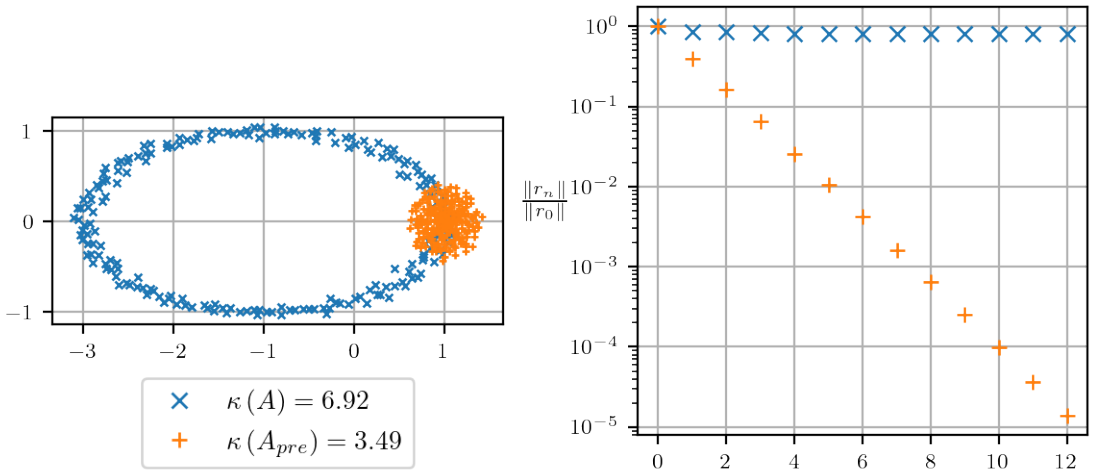


Figure 1.6: Spectrum of A in blue and $A_{pre} = AD^{-1}$ in orange (left). GMRES convergence for those matrices (right).

To better understand how preconditioning works, let us take an example, inspired by example 35.2 of [36]. We take a square matrix of size 200 by 200, which is the sum of a diagonal part and a random perturbations part:

$$\begin{aligned} A &= A_{diag} + \frac{1}{2\sqrt{N}} A_{rand} \\ \text{where} \quad A_{diag,k} &= 2\sin\left(\frac{2k\pi}{N-1}\right) - 1 + i\cos\left(\frac{2k\pi}{N-1}\right) \\ \text{and} \quad A_{rand} &\hookrightarrow \mathcal{N}(0,1)\,. \end{aligned} \tag{1.52}$$

The vector b is a vector of ones. The spectrum of A is displayed in figure 1.6 in blue. It is spread out around the origin. When we apply the Jacobi preconditioning, meaning the preconditioning matrix is the invert of the diagonal part, the spectrum is gathered around 1. This helps a lot GMRES, as can be seen in figure 1.6: it struggles to reduce the residual norm on the standard problem but does it easily on the preconditioned problem. This is only a dummy example, but it helps visualise the importance of preconditioning.

We need to use preconditioners when solving the linear system (1.43) with GMRES. One of the less sophisticated preconditioners is the Jacobi one. It is the one used in the example above: the preconditioning matrix is the invert of the diagonal part of A. Its advantage resides in its simplicity. It is inexpensive and easily scalable. On the other hand, it lacks efficiency, and that is why many preconditioners were developed in the literature. A slight variation consists in taking the diagonal blocs instead of the diagonal elements. This makes sense for our matrices, as they can be divided into blocs, each block corresponding to the degrees of freedom in a cell. The resulting preconditioner, called Block Jacobi, already exists in our solver and is often used as the standard preconditioner.

Others preconditioners exist in our solver. One consists in using the invert of the cell volumes as a diagonal matrix: the preconditioning matrix is a diagonal matrix, where each diagonal

element is the inverse of the corresponding mesh cell volume. This was originally conceived to preserve the conservative properties through the GMRES solver. Indeed, as GMRES produces an approximation of the linear problem solution, the overall scheme may not be conservative. This preconditioner makes sure it is. More explanations on this statement can be found in appendix A. As it is even simpler than the Jacobi preconditioner, it shows poor performance. Some work was done towards Polynomial preconditioning: the preconditioning matrix P is equal to the partial Neumann series of $\mathrm{Id} - A$, as it approximates A^{-1} [52]. Its major drawback is its computational cost, as it requires multiple matrix-vector products. For this reason, we decided not to look into it.

We were at first interested in some other preconditioners found in the literature, such as the Incomplete LU preconditioner. When taking the LU factorisation of the sparse matrix A, the triangular matrices L and U lose the sparsity pattern of A. This is troublesome as it is not possible to store dense matrices of the size of A. The Incomplete LU preconditioner, characterised by its index k, computes only the coefficients of the triangular factors that belong to the sparsity pattern of A^{k+1}. This way, the factorisation is incomplete in the sense that it does not recreate A, but it is still sparse and can be used as a preconditioner. The Incomplete LU factorisation is often used in computational fluid dynamics [53, 54], but it was not a good fit for our solver, due to the difficulty in developing the method in an industrial-size solver, and because it relies directly on the matrix. Indeed, the matrix available to us is not precise, as we will explain later, so using it as a preconditioner may not work as well as expected.

As the preconditioning matrix P is not explicitly needed, and its effect should be near A^{-1}, one could take as a preconditioning procedure another Krylov subspace method: to apply the preconditioning to any vector v would mean solving $Ax = v$ with a Krylov subspace method. However, applying a Krylov subspace method is not a linear operator. To handle this case, the GMRES algorithm must be modified into the Flexible Generalized minimal residual method, or FGMRES [55, 56]. GMRES can be preconditioned with an inner GMRES, and as GMRES was already written in our solver this idea was simple to implement. This preconditioning does not need more information on the matrix A than the outer GMRES does: in particular it does not require knowing the matrix coefficients. This is interesting for some reasons that are discussed later. Finally, this method interests many scientists [42, 43] and has shown promising results in numerical simulation [57]. For all those reasons, we decided to add this method to our solver.

The GMRES method is often used with restarting: instead of keeping on iterating the method, one could stop it and start again. This is the Restarted GMRES method. The cost of one iteration increases as the method iterates. Restarting allows for some control of the Krylov subspace dimension, and therefore on computational cost and memory usage. The issue with restarting is that it can be harmful to the convergence. When the Krylov subspace generated on the restart is too close to the previously generated Krylov subspace, the residual norm may stagnate [58]. A solution to this problem would be to reuse information between restarts. This is done with the Loose GMRES method [59] or with Augmentation or Deflation techniques [60, 61, 62]. Augmentation and Deflation give in fact equivalent results [42]. Their idea is to recycle the spectral information acquired at the end of the cycle onto the next cycle. Recycling spectral information could even be done throughout multiple linear solve, during a nonlinear solve for example [31]. Those techniques look promising but we did not have time to try them in our solver, as we focused on other points.

1.4.4 Evaluating the Jacobian matrices

If we can solve precisely linear problems, but the problem is not the one the nonlinear solver wants, it may hurt the nonlinear solver convergence. We then need to be able to get the right linear problem from the nonlinear one. At this point, we know how to evaluate the function f from equation (1.41), which is often expressed as a linear combination of previous states and G evaluations, where G is the function introduced in equation (1.10), the function from the starting ordinary differential equation. On the other hand, computing its derivative with regard to x is a different story. As this derivative is the matrix used when solving the linear problem, it is crucial to have a good representation of it. This derivative $f'(x)$ uses the Jacobian matrix of the function G. As the function G comes from the spatial discretisation method applied to the original partial differential equations (1.1), it is quite complex to evaluate this Jacobian. The underlying algorithm is hard to fully understand and write, and the numerical evaluation is usually expensive. Furthermore, our software keeps evolving, and models are constantly added and modified. It would require constant work to maintain the Jacobian matrix computation. Computing by hand the exact Jacobian matrix would amount to too much work in our industrial software. We must use other alternatives to get the Jacobian matrix we need for Newton's method.

An idea would be to use *Automatic Differentiation* [63]. This means to give the source code of the G function to some software [64], which gives in return a way to compute its Jacobian matrix. One advantage of this method is that the cost of derivating the function is done only once, at software compilation. After that, computing the Jacobian matrix amount to calling a function. Another advantage is that the given Jacobian is supposedly exact, contrary to some alternatives we will discuss later. For those reasons, Automatic Differentiation is today being used in actual computational fluid dynamics software [65, 66]. In our software, using Automatic Differentiation did seem too hard and therefore we looked at other methods instead.

A possible idea is to use an approximation of the Jacobian matrix that is inexpensive to compute. We said that the function G comes from the second-order or higher-order Finite Volume method used as the spatial discretisation method. One can then take the function G_1 given by the first-order corresponding method, and use its Jacobian matrix instead. Indeed, computing G_1 is inexpensive in comparison to G, and the same goes for the corresponding Jacobian matrices. Going further, one could also approximate the Jacobian matrix of G_1: when G uses some complex models such as turbulence models, it is often decided to not include those models' contributions to the Jacobian matrix, for complexity and stability reasons [7]. This is what is done originally in our in-house solver: using a cheap low-order approximation of the Jacobian matrix.

When we decided to use Krylov subspace methods as our linear solver, we highlighted the fact that they only need to know how to compute matrix-vector products. Using this, and the fact that the matrix is a Jacobian matrix, one could approximate the matrix-vector product of $f'(x)$ with a vector v by:

$$f'(x)\, v \approx \frac{f(x + \varepsilon v) - f(x)}{\varepsilon} \tag{1.53}$$

with a scalar parameter ε discussed below. This is the finite difference approximation of the Jacobian matrix-vector product. A quick analysis shows that the error on this approximation is $o(\varepsilon)$. Usually, the value of $f(x)$ is previously computed, so one evaluation of f is required for

one matrix-vector product. One could also use the centered finite difference approximation:

$$f'(x)\,v \approx \frac{f(x+\varepsilon v) - f(x-\varepsilon v)}{2\varepsilon} \tag{1.54}$$

that gives a smaller error of $o\left(\varepsilon^2\right)$, but it requires two f evaluations, so it is twice as expensive as the first-order approximation. Therefore, we will use the first-order approximation.

Using this approximation, it is possible to recover the full Jacobian matrix. If the approximation (1.53) is applied to each vector of the canonical basis as the vector v, one can gather each column of the Jacobian matrix. This gives what is often called the finite difference Jacobian matrix. Instead of computing a matrix-vector product for each direction, a technique consists of computing independent directions beforehand to get a colouring of the matrix: two directions are of the same colour if they are independent through the matrix. Then, a smaller number of matrix-vector product approximations are required: as many as there are colours. This is often called the finite difference Jacobian matrix with colouring [67]. Those techniques are still not well fitted for our solver. As we work with large dimensions, computing the Jacobian matrix takes time. Furthermore, we explained that we do not need to know explicitly the matrix. That is why we will use the approximation (1.53) each time we need a matrix-vector product instead of computing the Jacobian matrix first.

We still have not talked about the parameter ε introduced in equation (1.53). As it represents the size of the step made to approximate a derivative, it should be small. But taking it too small leads to roundoff errors so it must be chosen carefully [68]. We decided to look at some strategies in the choice of ε, and this work will be presented in a later part.

Now the choices we made for our nonlinear and linear solve strategies are starting to make sense. Working with different Jacobian matrices is not an issue, as they are not computed. The Krylov subspace method GMRES does not need the Jacobian matrix, only to compute matrix-vector products, contrary to other iterative methods such as the Gauss–Seidel method. The flexible preconditioning also does not need the Jacobian matrix for the same reason, contrary to other types of preconditioners such as ILU preconditioners.

Overall, using Newton's method to solve the nonlinear problem and a Krylov subspace method to solve the linear problem while using a matrix-free method is known as the Jacobian-Free Newton–Krylov method, or JFNK. This methods and its variants are as of today still discussed [69, 8] and used on actual computational fluid dynamics solvers [53, 38]. We are interested in the JFNK method as we hope it will give a better representation of the Jacobian matrix than the low-order one already in use in our solver. We hope that a better Jacobian matrix will lead to a better convergence, in particular when some models are ignored in the classic Jacobian matrix, such as turbulence models.

Furthermore, CEDRE works with multiple distinct solvers. When a computation uses multiple solvers, two, for example, the function f from equation (1.41) is in fact (f_1, f_2) and x is (x_1, x_2). The Jacobian matrix is then:

$$f'(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix}. \tag{1.55}$$

The cross-term Jacobian matrices, $\frac{\partial f_i}{\partial x_j}$ with $i \neq j$, are usually hard to compute, both analytically and computationally, and are the main obstacle to a fully implicit solver. Instead, in CEDRE, each solver iterates independently to the others, with some exchanges between iterations. Using the matrix-free approximation and the JFNK method, those cross-solver terms are taken into account. This choice is then perfectly aligned with the direction our solver is aiming: towards a better implicit cross-solver mechanism.

Some specific situations are known to be troublesome for the JFNK method, such as shocks, reaction fronts or discontinuities arising from high-order advection schemes [68]. Unfortunately, those features are precisely at the center of our solver. We are hoping that the difficulties of the method will be overbalanced by its advantages: getting a better convergence, allowing a fully coupled implicit solve and allowing an implicit integration method for new models. It means that we do not expect the method to enhance the robustness of the solver, but we hope that because it uses a better Jacobian matrix than the one already in use it will help the convergence. And once again, the second advantage is that the matrix-free method would allow us to do coupled implicit time integration. We will come back to the third advantage later in the application's part.

In this section, we described how an implicit method operates: the method itself produces one or several nonlinear problems at each time step. Those problems are solved with a nonlinear solver that uses linearisation to transform them into linear systems, which are then solved with a linear solver like a preconditioned GMRES method in the case of our solver. We explained that this linearisation in CEDRE is done by using Jacobian matrices that are poorly approximated. It means that we do not solve the correct linear systems, which hinders the nonlinear resolution and the time integration method ends up finding an incorrect increment. The Jacobian-Free Newton–Krylov method, which is compatible with other algorithms already available in CEDRE such as the GMRES method, should rectify this issue. For this reason, we will continue this work with this method for our time integration method.

Chapter 2

Description of the development workflow in CEDRE

Résumé du chapitre : Description des développements dans CEDRE

Le but de ce chapitre est de décrire les développements réalisés dans CEDRE afin d'implémenter le mécanisme "sans matrice" de la méthode JFNK. Il s'agit de développer astucieusement ce mécanisme afin d'obtenir la méthode souhaitée tout en garantissant l'efficacité et la portabilité du code produit.

Dans ce chapitre, nous décrivons un peu plus en détail les différentes briques qui composent CEDRE. Nous donnons ensuite les idées utilisées afin de réaliser les développements de la manière la plus générique possible. La nécessité de donner ces détails vient du fait que CEDRE est un code particulièrement massif, qui se prête mal aux développements. Afin d'obtenir le résultat souhaité, nous avons donc utilisé des notions modernes du langage Fortran que nous détaillons dans ce chapitre. Nous précisons ensuite un point précis de l'implémentation de la méthode JFNK, qui est le choix du paramètre de différenciation ε, en apportant une correction par rapport aux choix principalement utilisés par le reste de la communauté. Nous montrons enfin comment ces nouveaux développements s'articulent autour des mécanismes déjà existants, pour vérifier la compatibilité de l'ensemble des méthodes.

In this brief chapter, we will discuss the details of how we implemented selected methods in CEDRE. This does not constitute research work, but it ended up being a large part of the work done during this thesis. It is also not without interest, as we sometimes used advanced features in order to implement what we set out to do.

2.1 General description of CEDRE

The software system CEDRE gather several solvers to solve problems in the field of multiphysics [1]. Each solver is dedicated to a given model. As of today, there are seven solvers embedded in CEDRE:

- CHARME, the fluid solver, for compressible multifluid and reactive flow, with RANS or LES turbulence models
- SPIREE, the dispersed phase solver using an Eulerian framework
- SPARTE, the dispersed phase solver using a Lagrangian framework
- ASTRE, the radiation solver using a Monte Carlo method
- REA, the radiation solver using a discrete ordinates method
- FILM, for shallow water equations, used to model ice accretion
- ACACIA, the conduction solver, for heat transfer in solids.

Combining different solvers, CEDRE is able to simulate multiphysics phenomena numerically. Using those solvers, CEDRE applications go from aerodynamics to aeroacoustics, aerothermodynamics, combustion, icing, etc. The solver is coupled either through boundary conditions for example in a thermal interaction at a fluid-structure contact or inside the computational domain for example in the case of mass and energy transfer between dispersed phases and the main flow. The coupling can either be one-way or two-way, depending on the user's choice. Each solver is integrated in time separately, and the coupling consists of some data exchange between iterations: it is an explicit coupling.

Some functionalities common to multiple solvers exist outside the solver in helper libraries. For instance:

- ASSEMBLAGE acts as the conductor by handling the overall simulation, telling the solvers what to do and when to do it, when to exchange data and with which other coupled solver
- BIBCEDRE contains tools for geometrical operations, linear algebra methods, mesh handling, parallel communications and other general functionalities
- THERMOLIB is used to compute the different thermophysical properties such as heat capacities or to compute the closure relations that can be complex in our applications.

Those libraries are the main engines of the solver. CEDRE does a computation with CHARME, it is in fact ASSEMBLAGE that tells CHARME when to initialise itself, to do a step of the time integration method, etc. When CHARME needs to compute the ordinary differential equation function, it is BIBCEDRE that does the higher-order polynomial reconstruction of the solution so that the Riemann solver can compute the interface flux. When CHARME needs to compute the primitive variables such as the pressure and temperature from the conservative variables, it is THERMOLIB that solves a nonlinear problem with Newton's method to compute the result.

Now that we defined the existing framework for the software system CEDRE, we can develop new functionalities. Despite allowing some flexibility in the programming language, most of CEDRE is written in Fortran. We decided to keep working with Fortran to help with the integration of our work. As CEDRE is used by industrial clients, and as they rely on their own supercomputers, we need to limit ourselves to Fortran 2003 standards, so as to ensure compatibility.

2.2 Implementation details

In this thesis, we focused on the most used solver: CHARME. Indeed, not only is it the most used, but other solvers use it as a base for many applications. For example, when simulating ice accretion around a wing profile, a standard methodology with CEDRE is to first get the base aerodynamic flow with CHARME and then compute the ice particles with SPIREE or SPARTE. Working with the solver CHARME was the way to benefit the most from our work. Even if during this thesis we only worked on CHARME, we always kept in mind that the finality was multiphysics simulations using multiple solvers. That is why we tried to develop generic functionalities so that they could be easily imported to other solvers, provided the developers of said solvers wanted to use them. The same reason was also used as a criterium in our choices, as was explained previously. Choosing the Jacobian-Free Newton–Krylov method goes towards fully implicit coupling between solvers, instead of the explicit coupling existing today.

2.2.1 FGMRES

In order for our work to be usable in every other solver, we had to work on the common library BIBCEDRE. When the implicit Euler method of CHARME needs to solve a linear problem, it uses BIBCEDRE. It contains everything needed to solve linear problems, such as GMRES and preconditioners. A linear problem is stored in BIBCEDRE as the Fortran derived type `type_sys`. In order to add Flexible preconditioning to the existing GMRES, we added a pointer to an inner instance of `type_sys` inside of `type_sys`, so that the linear system and its corresponding solver may use an inner solver for an inner problem:

```
type type_sys
  ! Inner linear system and solver
  type(type_sys), pointer :: sys_int => null()

  ... ! Additional data
end type type_sys
```

This way, when we need to apply the preconditioner during a GMRES iteration, we can use the inner `type_sys` instance to call the inner GMRES. Furthermore, having a pointer to an inner instance allows for more freedom for the inner solver. One could for instance use multiple depths of preconditioning and have the inner GMRES also be an FGMRES method, preconditioned by another GMRES, etc.

2.2.2 Matrix-free

Sparse matrices are stored in an in-house format, using an array for the diagonal blocks, another one for the extra-diagonal blocks and a third one to index the extra-diagonal blocks. Matrix-vector products are made inside BIBCEDRE to handle this matrix format, with the routine:

```
subroutine gmvec(sys, i_p, i_ap)
  type(type_sys), intent(inout) :: sys
  integer,        intent(in)    :: i_p
  integer,        intent(in)    :: i_ap
```

that takes three arguments: the `type_sys` instance, an index identifying the vector to multiply and an index identifying the vector where to put the result. As we explained when we introduced

Krylov subspace methods, GMRES uses the linear system matrix through this matrix-vector product routine. Classically, a client solver such as CHARME fills the matrix coefficients and then lets BIBCEDRE solve the linear system. In order to use the matrix-free approximation from equation (1.53), one only needs to replace this routine with a new one that computes the approximation. Unfortunately, the approximation uses a function that belongs to the client solver. The library BIBCEDRE does not know this function and how to compute it, as it is part of CHARME or any other client solver. As we said, we want to write generic solutions, so merging the library BIBCEDRE with the solver CHARME is not a good way to proceed. What we need here is to allow BIBCEDRE to use a callback from the client solver. We did that using the Fortran 2003 feature: polymorphism. The Fortran 2003 standard is almost twenty years old today, and many may consider that the functionalities added to the previous Fortran 95 standard are therefore not that innovative. But in the context of a project as large as CEDRE, developed by researchers and used by industrial partners, even the Fortran 2003 standard and polymorphism in particular are innovative. Without going into too much detail, we added a member to the type `type_sys` that contains the context to evaluate a matrix-vector product:

```
type type_sys
  ! Matrix-vector product context
  class(type_gmvec_ctx), pointer :: gmvec_ctx => null()

  ... ! Additional data
end type type_sys
```

with:

```
type type_gmvec_ctx
  procedure(interface_gmvec), pointer, nopass :: gmvec

  ... ! Additional data
end type type_gmvec_ctx
```

This way, when the client solver creates an instance `sys` of `type_sys`, it can choose how to evaluate matrix-vector products by setting the procedure pointer `sys%gmvec_ctx%gmvec`. It can for example point to the already existing routine to use the classical matrix-vector product, but it also can use a custom routine that implements the approximation (1.53). Furthermore, the client solver can use polymorphism and create an extended type of `type_gmvec_ctx` in order to store additional data in the context. This is what is done by the solver CHARME, as it does need additional data to approximate the Jacobian matrix-vector product. Finally, with this implementation, any solver that wants to use the approximation (1.53) just needs to write the corresponding routine and set the context accordingly. Then, a user can choose at execution time whether to use the standard Jacobian matrix or the matrix-free method.

The computational cost of the matrix-vector approximation is approximately the cost of one function evaluation. It requires some additional operations, such as vector subtractions, and vector scaling, but those operations are way less expensive. Using a matrix-free algorithm is clearly less expensive in terms of memory, as the matrix does not need to be stored at all. Some even claim that it can lead to a less expensive method in terms of time [69]. This happens when the alternative, which is computing the Jacobian matrix, is an expensive task. In CHARME, the Jacobian matrix that is traditionally used is inexpensive to compute, with a cost similar to a function evaluation. The issue is that the function evaluation itself is expensive in our solver. This is troublesome, and the issue is currently addressed by the team of developers in charge

of the solver. In the meantime, it means that the cost of the matrix-free approximation will suffer even more. Practically, using it increases the computational time of one iteration. We now hope that as it will not improve the iteration speed, it will improve the convergence of our solver. Also, as we said the team responsible for CEDRE is working to improve the speed of the function evaluation. If they succeed, our method will then benefit greatly from their work.

2.2.3 Strategy for the choice of ε

When we introduced the approximation (1.53) it introduced a new parameter ε. It is easy to check that the truncation error on this approximation decreases linearly with regard to ε. It is natural to take a small value for ε. But unfortunately, when working with floating-point arithmetic, dividing by a small ε introduces roundoff errors [70]. This parameter needs to balance truncation and roundoff errors. We need to decide on a strategy for the choice of epsilon. One could take for example $\varepsilon = \sqrt{\varepsilon_{\text{mach}}}$ where $\varepsilon_{\text{mach}}$ is the machine epsilon: around 10^{-6} for single precision and 10^{-15} for double precision. This choice is often discarded as it is deemed too simplistic. Instead, works from the literature tend to use the same few options [6, 53, 71] that come from [72] and [73]. Those options are well described in [68]. In particular, the one that we encounter the most is the one from [72]:

$$\varepsilon_{\text{wp}} = \frac{\sqrt{\varepsilon_0 \left(1 + \|x\|_2\right)}}{\|v\|_2} \tag{2.1}$$

using the same x and v as in equation (1.53). Here ε_0 is the estimated relative error in function evaluation. A reason this choice is so popular is that apart from this ε_0 value, it does not require any user input. Furthermore, ε_0 is often simply set to machine epsilon $\varepsilon_{\text{mach}}$.

To analyse this strategy in the choice of ε, we do the following numerical experiment. We consider the one-dimensional Burgers' equation over a regular periodic mesh made of 10 cells (or segments in the one-dimensional space). The function f is taken as the right-hand side of equation (1.5) when solving the Burgers' equation, using a first-order Finite Volume method as the spatial discretisation method, itself using an exact Riemann solver (Godunov's scheme). This function was chosen because it is a nonlinear conservative equation, often viewed in computational fluid dynamics as a simplified version of Euler equations and it contains most aspects of nonlinear hyperbolic equations. The vectors x and v are $10 + r_1$ and $0.1\,(2r_2 - 1)$ where r_1 and r_2 are random vectors following a uniform distribution on $[0, 1[$, obtained with the Python package NumPy: `numpy.random.random`.

The experiment result can be seen in figure 2.1. The relative error in the approximation is shown as a function of ε with small grey crosses. On the right part of the figure, the error decreases linearly with regard to ε. This corresponds to a truncation error-dominated region. On the left part, the error increases as ε decreases. This corresponds to a roundoff error-dominated region. Also, this increase is not smooth as the linear decrease, because roundoff errors tend to produce more chaotic results. The choice of ε from [72] is also shown in the figure, as the large blue dot. As can be seen, it falls into a low error region, not too much on the left, not too much on the right.

The issue we noticed is the following. Some references in the literature do not specify the norm used in equation (2.1). As it is most of the time the Euclidian norm $\|\cdot\|_2$, or 2-norm, we can assume that it is also the case when it is not specified. However, this means that in our

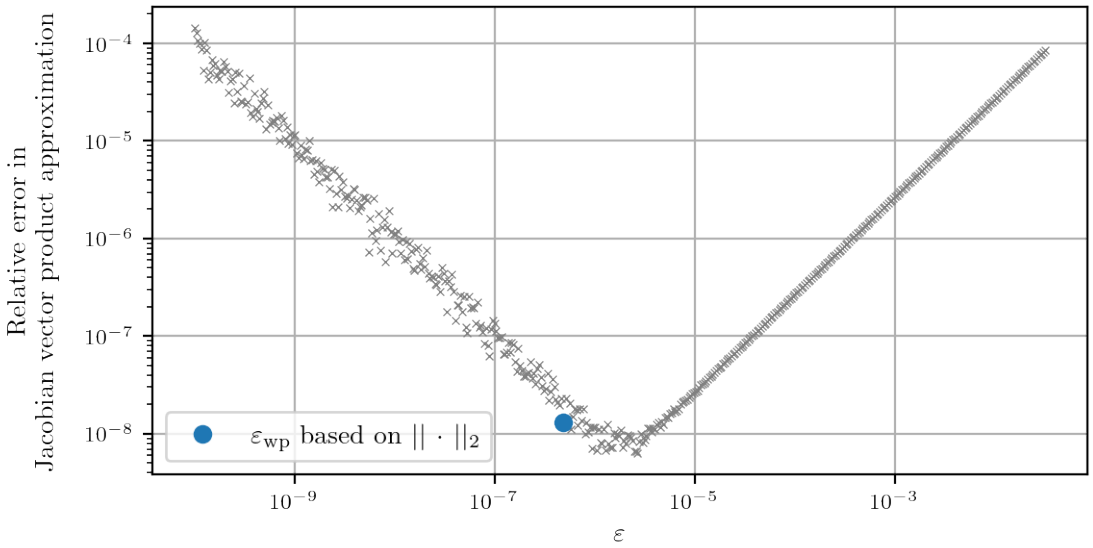


Figure 2.1: Error in the Jacobian matrix-vector product approximation as a function of ε, and in particular for the most popular choice ε_{wp}. The function is the right-hand side of equation (1.5) from a Finite Volume method using an exact Riemann solver for Burgers' equation, over a 10-cell one-dimensional regular mesh.

example the value of ε_{wp} will depend on the vector size. With our example, if one increases the number of cells in the mesh the typical size of the vector components stays roughly the same, so the shape of the error as a function of ε should not change much from the one in figure 2.1. But if the vector components size stays the same, the 2-norm does increases with the vector dimension. One can even see that when the dimension is $N \gg 1$, $\varepsilon_{\text{wp}} \sim N^{-1/4}$. Having ε to depend on N did not make sense to us, as we are working with possibly large vectors in our industrial applications. That is why we decided to use a variation from what is currently found in the literature: we will use the norm $\|\cdot\|_2/\sqrt{N}$ that can be seen as a scaled 2-norm. We then get a new strategy for the choice of ε that we note $\varepsilon_{\text{wp},N}$:

$$\varepsilon_{\text{wp},N} = \frac{\sqrt{\varepsilon_0 \left(1 + \|x\|_2/\sqrt{N}\right)}}{\|v\|_2/\sqrt{N}}. \tag{2.2}$$

We compared the two strategies on the same test case while increasing the number of cells from 10 to 100, 1000, ..., up to 10^7. Figure 2.2 shows the result of this experiment. On the left, we see that the shape of the relative error as a function of ε when $N = 10^7$ is similar to when $N = 10$, albeit the roundoff error-dominated region is more regular. In particular, the ideal trade-off between the two error types did not move a lot. The value of ε_{wp} decreases as expected: this translates as the fact that the blue dot moved to the left. When looking at the error as a function of the dimension N, one can even recover the $-1/4$ expected slope. In the right figure, there is in fact a $1/4$ slope but the error is inversely proportional to the value of ε so if the error behaves as $N^{1/4}$, then ε_{wp} behaves as $N^{-1/4}$. Because of the shape of the error as a function of ε, if ε_{wp} changes with N it will eventually end up increasing the error. This is

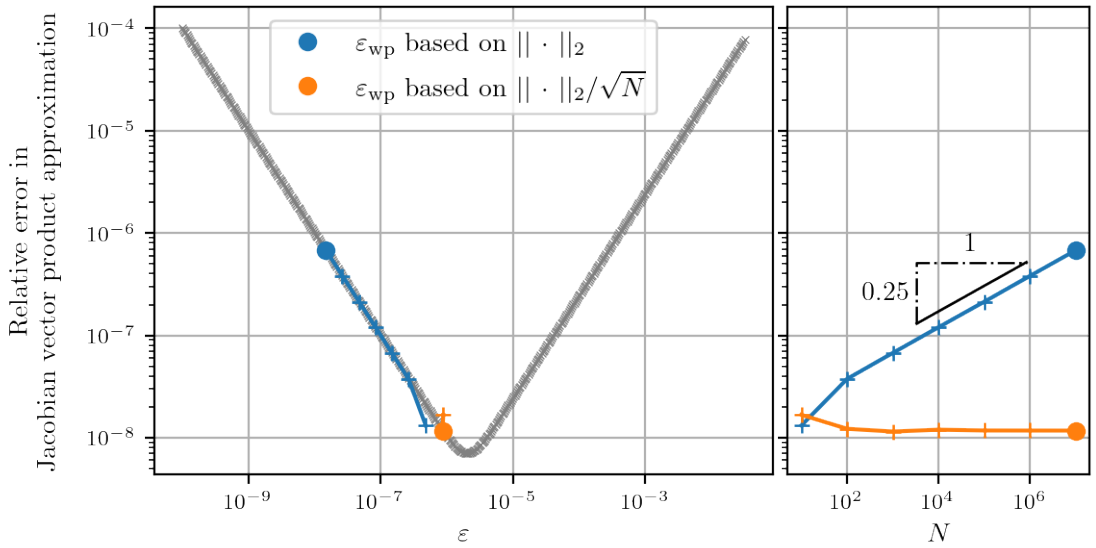


Figure 2.2: Error in the Jacobian matrix-vector product approximation as a function of ε (left) and of the dimension N (right). On the left figure, coloured cross markers correspond to the values computed on a mesh of 10^1, 10^2, ... cells, and circle markers to the last value with 10^7 cells. Grey cross markers on the left figure correspond to a mesh of 10^7 cells.

an undesired feature. Our new choice $\varepsilon_{\text{wp},N}$ on the other hand does not change a lot when N increases. Therefore, the error level stays the same no matter the dimension.

The same numerical experiment was made on a more complex case: the one-dimensional Euler equations. Here, the function f corresponds to the right-hand side of equation (1.5) given by a centred Finite Volume method used as the spatial discretisation method, in which the interface flux is defined as an average of left and right fluxes. This means the Riemann solver just averages the left and right fluxes. This spatial method is known for its instability when used in an actual solver, but it is useful in this experiment as the analytic Jacobian matrix is easy to derive, contrary to methods using more complex Riemann solvers. This physical model assigns 3 degrees of freedom in each cell. We will use primitive variables. The vector x corresponds to a uniform density of $1\text{kg}\,\text{m}^{-3}$, a velocity equal to a sine making one period over the mesh and of amplitude $10\text{m}\,\text{s}^{-1}$, and uniform pressure of 10^5Pa. The vector v is a random vector in $[0, 1]$ as before, where the first component is scaled by 10^{-3}, the second by 10^{-2}, and the third by 10^2, in order to impose 10^{-3} relative perturbations.

Figure 2.3 shows the corresponding results. As for the previous experiment, ε_{wp} depends on the dimension N, and then the associated error ends up increasing as N grows. Our correction $\varepsilon_{\text{wp},N}$ does not exhibit the same drawback. Even if it does not fall at the bottom of the error curve, it at least does not lead to a larger error.

In this last case, we see that at the beginning the two errors were close with $\varepsilon_{\text{wp},N}$ being better than ε_{wp}. In the first one, their starting positions were a bit different and this time ε_{wp} was the better one. This is largely due to the randomness of this analysis: the one used to construct the x and v vectors. In fact, it would be unwise to draw a conclusion from the position of the

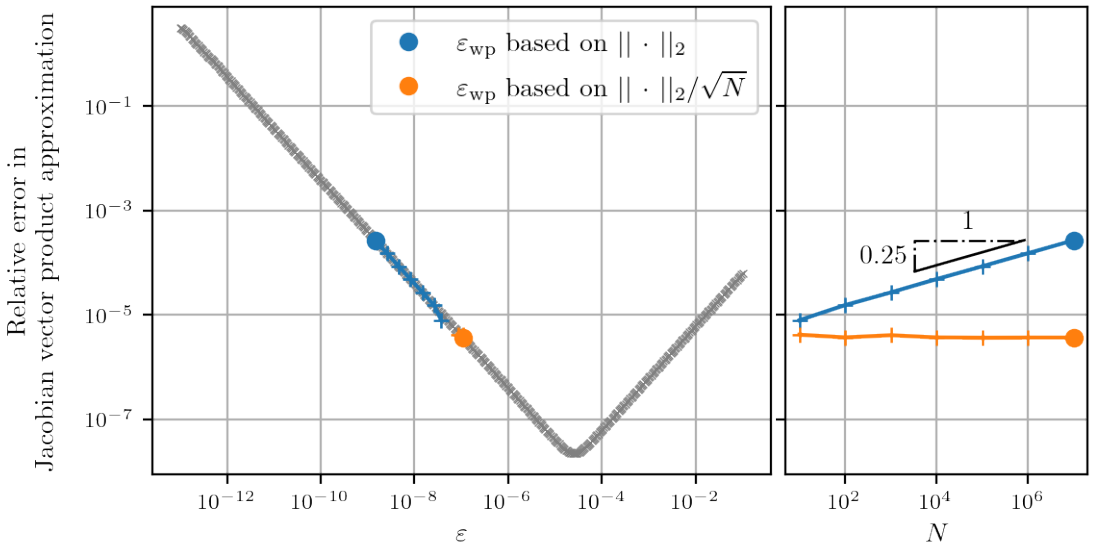


Figure 2.3: Error in the Jacobian matrix-vector product approximation as a function on ε (left) and of the dimension N (right). The function is the right-hand side of equation (1.5) from a Finite Volume method using a centred scheme for the Euler equations over a regular one-dimensional mesh. On the left figure, coloured cross markers correspond to the values computed on a mesh of 10^1, 10^2, ... cells, and circle markers to the last value with 10^7 cells. Grey cross markers on the left figure correspond to a mesh of 10^7 cells.

points in figures 2.2 and 2.3. Changing the choice of vectors, of the function f, or even the randomness in the vectors is enough to modify their position. For example, we can not conclude anything from the fact that $\varepsilon_{\text{wp},N}$ is almost at the bottom of the error curve in figure 2.2. It may as well be a bit more on the side like in figure 2.3. What we can use from this analysis, however, is the tendencies that choices of ε show. The main conclusion is then that with our strategy we removed a dependency of the relative error on the dimension, which makes sense as this dependency is not expected *a priori*.

The issue with this analysis is that it relies on extremely simple examples. The function f in our actual applications is in fact much more complicated than the two we used here. But in order to do this analysis we need to be able to compute the relative error, and this means we need to be able to compute a Jacobian matrix-vector product analytically. Unfortunately, this is not possible for our applications. If it was, we would not even need to introduce the approximation (1.53) and therefore do this analysis. Using one-dimensional Burgers' equation and one-dimensional Euler equations with a simple Riemann solver was then a good compromise. Those equations share similarities with the ones we work with in our solver, as they are nonlinear hyperbolic equations, but they are still simple enough so that they can be used here.

2.2.4 Linear system normalisation

An available option in BIBCEDRE for the linear solver is normalisation. It consists in taking an invertible diagonal matrix $\mathcal{N}$ and rewriting the linear system (1.43) as:

$$\tilde{A}\tilde{x} = \tilde{b} \quad \text{with} \quad \tilde{A} = \mathcal{N}^{-1}A\mathcal{N}, \quad \tilde{x} = \mathcal{N}^{-1}x \quad \text{and} \quad \tilde{b} = \mathcal{N}^{-1}b \tag{2.3}$$

One can check that it is equivalent to the original linear system. The diagonal coefficients of the matrix $\mathcal{N}$ are estimates of the order of magnitude of the corresponding degrees of freedom. This estimation can be computed at each time step depending on the current state, or can simply be constant, equal to some reference values, depending on the choice of the user. When working with Euler equations, for example, the numerical values corresponding to the energy degrees of freedom are usually many orders of magnitudes over other degrees of freedom. The goal of this normalisation is to bring back each vector component at the same level in the linear system. It can also be seen as a combination of a left preconditioning by $\mathcal{N}^{-1}$ and a right preconditioning by $\mathcal{N}$. This normalisation is often used by our solver's users, so we need to ensure compatibility with our matrix-free method. We can adapt the approximation (1.53) to account for the normalisation as:

$$\left(\mathcal{N}^{-1}f'(x)\mathcal{N}\right)v \approx \mathcal{N}^{-1}\frac{f(x+\varepsilon\mathcal{N}v) - f(x)}{\varepsilon}. \tag{2.4}$$

We also need to adapt our strategy for the choice of ε. With the normalisation, we see that $\varepsilon\mathcal{N}$ sort of plays the role of ε. It then makes sense to take $\|\varepsilon\mathcal{N}\|$ equal to the original choice of ε, and therefore we take:

$$\varepsilon = \frac{\sqrt{\varepsilon_0\left(1 + \|x\|\right)}}{\|\mathcal{N}\|\,\|v\|}. \tag{2.5}$$

We take as a matrix norm the one induced by the vector norm. This means that:

$$\|\mathcal{N}\| = \sup_{v\neq 0}\frac{\|\mathcal{N}v\|}{\|v\|}. \tag{2.6}$$

When using the 2-norm, the corresponding norm is $\|\mathcal{N}\| = \max\mathcal{N}_{ii}$ as $\mathcal{N}$ is diagonal with strictly positive coefficients. When using the scaled 2-norm discussed in the previous part, the corresponding norm is also the same. This is nice because whatever strategy we use in the choice of ε, it does not interfere with the handling of normalisation.

2.2.5 Preconditioning

Our matrix-free implementation also needs to be compatible with the already existing preconditioning from BIBCEDRE. There are two ways to precondition the linear system in BIBCEDRE.

- The *outer* left preconditioning is applied once before the linear solve. It changes the numerical values of the matrix coefficients and the right-hand side. Once it is applied, the resolution can continue as if nothing had happened.
- The *inner* right preconditioning is applied before each matrix-vector product, and once more at the end of the solve.

Handling the inner preconditioning with our matrix-free implementation is easy as those two operations are independent of each other. The outer preconditioning relies on the fact that the preconditioners available are simple, and therefore it is inexpensive to apply it to the whole matrix, and less expensive than applying after each matrix-vector product. This advantage is lost with the matrix-free method. The left preconditioning is still applied once to the right-hand side before the solve, but it is also applied after each matrix-vector approximation.

2.2.6 Local time-stepping

We said earlier that local time-stepping is used in CEDRE to accelerate the convergence on steady problems. What it does is that each cell of the mesh moves forward in time with its own time step, independently of other cells. This local time step is expressed as a fraction of the global time step. There are several options a user can choose from to set the time-stepping strategy. It can use a CFL number based option and compute the corresponding local time step so that the CFL number local to the cell is not too large. The preferred and default option is based on the explicit increment and some reference values. It first estimates what the solution increment will be by using the explicit increment and possibly the reference values, depending on the choice of method, and compute a local time step so that the increment local to the cell is not too large. This local time-stepping strategy adds various parameters that need to be tuned, in the form of scalar coefficients, whether or not to apply the local time-stepping to the boundaries, whether to use conservative or primitive variables, etc. In the end, what local time-stepping methods do is artificially modify the volume of the cells: increasing a cell volume corresponds to increasing its inertia, and therefore scaling the time step down. The volumes of the cells appear in the linear system matrix, in the form of a diagonal "mass" matrix. By using the scaled mass matrix, the matrix-free approximation does account for local time-stepping. With this last feature, we ensured that the matrix-free approximation is compatible with the already existing methods.

This chapter explained in a simplified manner some implementation detail of our solver. It is important to remind oneself that CEDRE is an industrial-sized software system, compared to smaller solvers developed for academic purposes. The scale of the solver was a limiting factor in this thesis. We often wanted to try some new scientific idea but were faced with many troubles that are inherent to the solver size. At some point at the early stage of this thesis, for example, we wanted to improve the quality of the linear solve by using advanced solvers and preconditioners, with the help of the PETSc library [74, 75, 76]. Using the library in CEDRE ended up being impractical, or at least would have taken too much time that we could not afford, and so we had to change direction. In the end, a solver of this size is not an ideal sandbox to try methods that are too innovative. A good amount of experience is required to master the various models and methods, an amount I did not have at the beginning of this thesis. For example, the spatial discretisation methods come with a lot of parameters: the kind of method, the slope limiter, the Riemann solver, and many more that are specific to CEDRE. The same goes for the time integration with for instance the local time-stepping or the linear system normalisation. A single parameter may be deterministic in the success of a computation. This forces us to allocate a quite large amount of time to every computation. Moreover, because of the large number of developers that are working on the solver, and the fact that many developments are small tweaks from occasional developers such as interns and PhD students, fixing bugs also took a decent amount of time. Overall, this explains why we decided to implement the solutions previously discussed. They are already well known from the literature and feel like a natural step up from the already existing methods of CEDRE. It is good to mention that many difficulties arise from the solver, and are inherent to it, mostly due to its overall size. We would probably have taken a different route if we had worked with a different solver.

Without getting into too much detail, this chapter explained how the selected methods and modifications were implemented into the solver CEDRE. Besides some minor developments, the main feature is the matrix-free implementation. As it was explained, this implementation is generic so that it can be used with any solver that wants to solve a linear problem with the library BIBCEDRE. The idea was also that if multiple solvers use this implementation, it becomes easy to do a coupled implicit solve. Without it, because of the structure of CEDRE with the library ASSEMBLAGE conducting the solvers, a coupled implicit solve is not possible without reworking the basic structure of CEDRE. During this thesis, however, because we were limited in the time allowed to us, the implementation was only used for the reactive Navier–Stokes solver CHARME. This is why we had to compromise and develop only for the CHARME solver but in a generic way.

Chapter 3

Analysis of the JFNK method in CEDRE

Résumé du chapitre : Analyse de la méthode JFNK dans CEDRE

Le but de ce chapitre est de vérifier les gains apportés par l'ajout de la méthode JFNK dans le solveur CEDRE. Il s'agit de comparer les performances de cette nouvelle méthode avec les méthodes préexistantes en matière de stabilité, robustesse et rapidité. Nous adaptons pour cela le point de vue d'un utilisateur, en considérant ce qui l'intéresse lorsqu'il réalise une simulation stationnaire.

Pour cela, nous comparons tout d'abord la méthode JFNK avec la méthode utilisant explicitement la Jacobienne. Cette comparaison se traduit principalement par la comparaison de la convergence des deux méthodes à travers la norme des résidus. Ainsi, nous réalisons cette comparaison sur une succession de cas tests, de complexité croissante, sélectionnés car ils représentent différentes facettes du solveur.

Nous commençons par un calcul purement aérodynamique d'un profil d'aile en deux dimensions dans un écoulement turbulent, qui utilise une modélisation des effets turbulents de la couche limite. Puis, nous utilisons le même profil mais cette fois en incidence, avec un maillage bien plus fin qui capture les effets de couche limite turbulente, plutôt que de la modéliser.

Nous regardons ensuite un cas de rentrée atmosphérique avec une sphère placée dans un écoulement à haute énergie. Cela se traduit par un très fort choc, ainsi qu'une zone de déséquilibre thermodynamique où ont lieu d'intenses réactions chimiques. Dans un premier temps, nous regardons comme précédemment les deux méthodes pour comparer leur convergence. Pour finir, nous utilisons un modèle de fluide récemment ajouté au solveur afin de représenter plus finement le déséquilibre thermodynamique en caractérisant l'écoulement par plusieurs températures distinctes. Avec les développements réalisés actuellement sur ce nouveau modèle, il n'est pas possible d'utiliser la méthode implicite préexistante, et les utilisateurs sont forcés d'employer des méthodes explicites. Nous montrons donc que la méthode JFNK peut elle être utilisée en l'état, sans nécessiter de développements supplémentaires, et permet d'accélérer les calculs ainsi que d'obtenir une meilleure convergence.

In a previous chapter, we identified some methods from the literature that we wanted to use in our solver CEDRE, and some others from CEDRE that we wanted to improve. In another chapter, we discussed the practical implementation of said methods in the solver. The goal of this chapter is now to test those methods on several applications to comment on the choices we made. We need to define test cases that represent well enough target applications so we can comment on the performances of our choices.

3.1 Comparison between matrix-free formulation and Jacobian matrix approximation

From the previous analysis and implementation, the main addition to the solver is the Jacobian-Free Newton–Krylov method, and in particular the matrix-free approach. In this part, the new method that uses the matrix-free approach will be compared with the implicit Euler method as the interest is focused on implicit time integration schemes. The traditional method linearises the equation that the implicit Euler method produces, approximates the Jacobian matrix using the Jacobian matrix of the corresponding first-order scheme, and then solves the linear system with the Krylov subspace method GMRES. In order to understand the impact of a better Jacobian matrix, the only difference is how the Jacobian matrix is handled. The new method will then work in a similar way, except the matrix used in the linear solver is not actually computed, but the matrix-vector products are approximated using equation (1.53).

3.1.1 Turbulent transonic airfoil

Definition of the test case

The first application is a typical aerodynamics test case. It is a two-dimensional simulation of the flow around an RAE 2822 wing profile. The fluid is standard air assumed to be a perfect gas. The Mach number is taken equal to 0.75, the chord is equal to 1m, the angle of attack is 0° and we use the atmospheric conditions at 10km. This gives a laminar Reynolds number of 6.5×10^6.

This first test case is chosen for multiple reasons. Firstly, it is a simple case in the field of computational fluid dynamics. It is a standard aerodynamics case, with a small mesh in comparison to many other three-dimensional cases. This allows testing our methods inexpensively. Secondly, this case belongs to the tutorial suite of our solver. It means that it is already well mastered by the team. Thirdly, even if it is only a standard aerodynamics case it still has some stiff features, such as turbulence modelling and a shock. Finally, it is a standard test case for turbulence modelling validation. Therefore there are many references in the literature using this case.

The mesh used for this simulation is an unstructured hybrid mesh made of triangles and quadrangles. Parts of it can be seen in figure 3.1. Cell sizes range from 2.5m far from the airfoil and 100µm at the wall. At the wall, there is a C-shaped layer of regular cells. This helps better capture boundary layer effects near the profile and the wake. Also, under those conditions, a shock is expected to develop on the upper part. Special treatment such as refinement was applied to the mesh at the expected shock location.

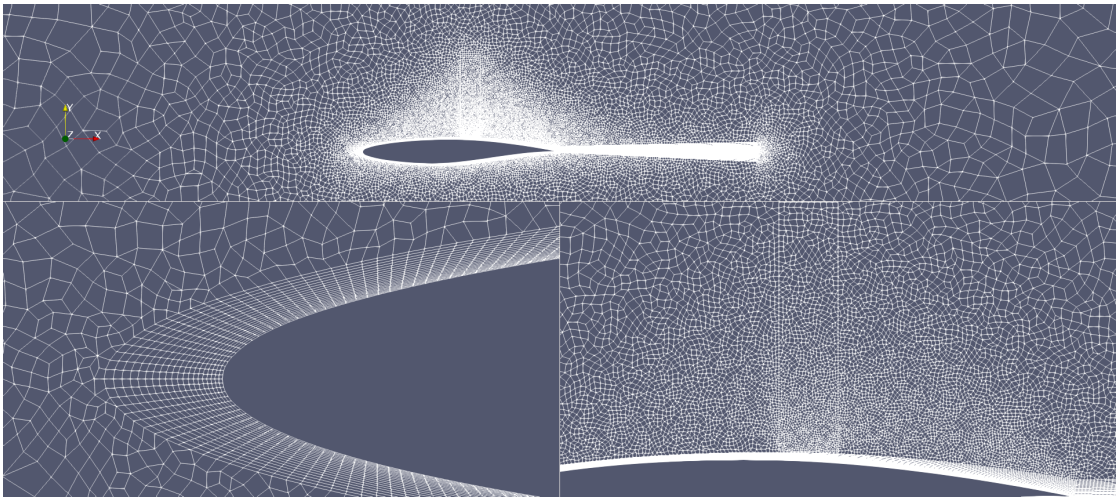


Figure 3.1: Mesh for the RAE 2822 test case. Close up on the leading edge and on the expected shock location.

The model used is the Reynolds Averaged Navier–Stokes equations, or RANS. Simply put, every scale of the turbulence is modelled. The Spalart–Allmaras turbulence model is chosen to close the RANS equation. It is well known for being one of the simplest turbulence models, which is fine to set up a simple first test case. Figure 3.1 shows that the mesh near the wall boundary is not particularly fine, and therefore it is not fine enough to compute the boundary layer. It is indeed because this computation uses a turbulence wall model that replaces the standard no-slip boundary condition with a more complex relationship between the variables and their derivatives. The physics of the boundary layer is introduced in the model, which limits the stiffness of the system. Such models are known to be troublesome with some more complex turbulence models such as the $k - \epsilon$ model but are used with others such as the one we use here. Turbulence wall modelling may be nonconventional in aerodynamic simulations but it is a commonly used feature of our solver, so it is interesting to see how our new method behaves on such test cases. The spatial discretisation method is a second-order Finite Volume method, using the HLLC Riemann solver and Multislope method [19] with a Van–Leer slope limiter. Local time-stepping is used to speed up the convergence.

Analysis of the results

We are now going to compare the results from the two different simulations. It is first worth noting that before trying the JFNK method, the computation needs to start using the traditional method. As explained before, the matrix-free approximation is not ideal to handle discontinuities such as shocks, and a shock is indeed present in this computation. Only some iterations are needed, just to start the development of the boundary layer and approximately place the shock. After that, the computation can be continued, with the traditional method on one side and with the new method on the other.

The global flow is shown in figure 3.2 through the pressure field near the airfoil. The shock is clearly seen on the upper part. The two computations give similar results: they are indistinguishable by just looking at the pressure field. This is to be expected as the traditional method

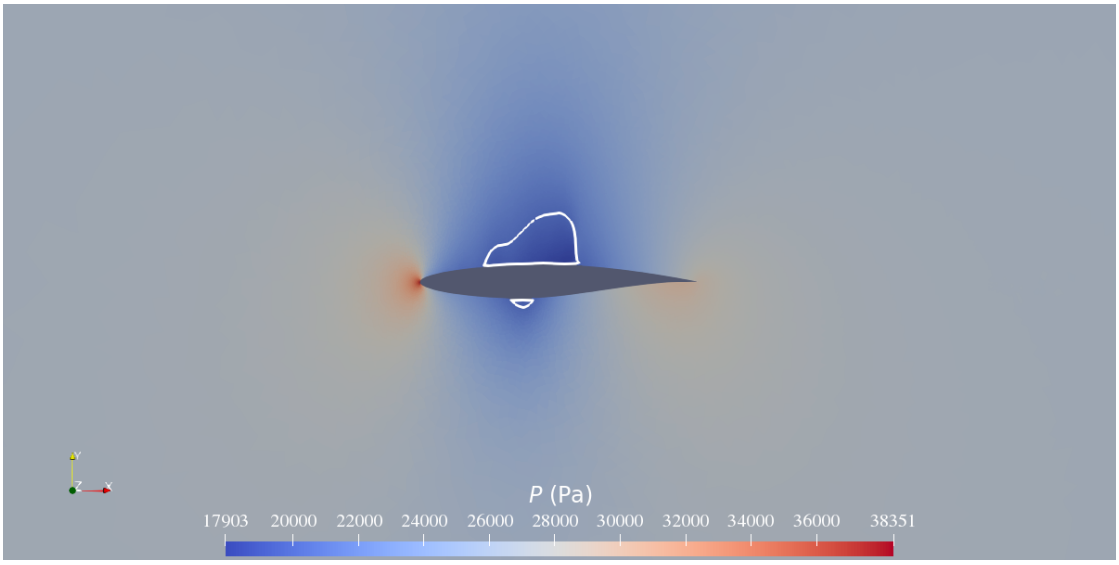


Figure 3.2: Pressure around the RAE 2822 airfoil and sonic Ma = 1 contour.

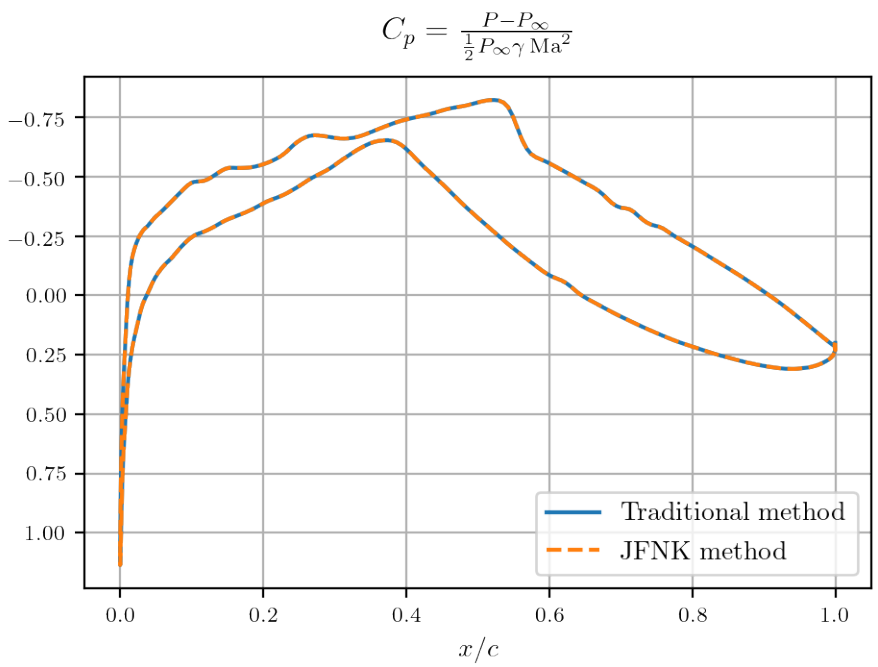


Figure 3.3: Pressure coefficient around the airfoil for the traditional method (blue) and the JFNK method (orange).

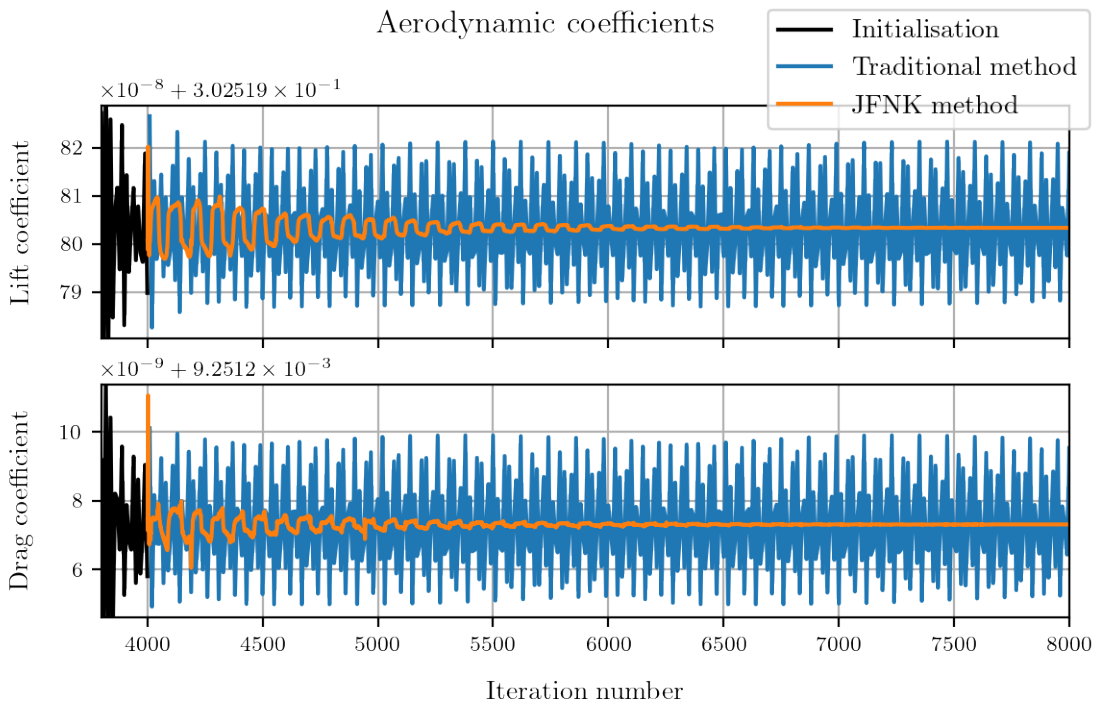


Figure 3.4: Aerodynamic coefficients (lift and drag) for the profile throughout the computation.

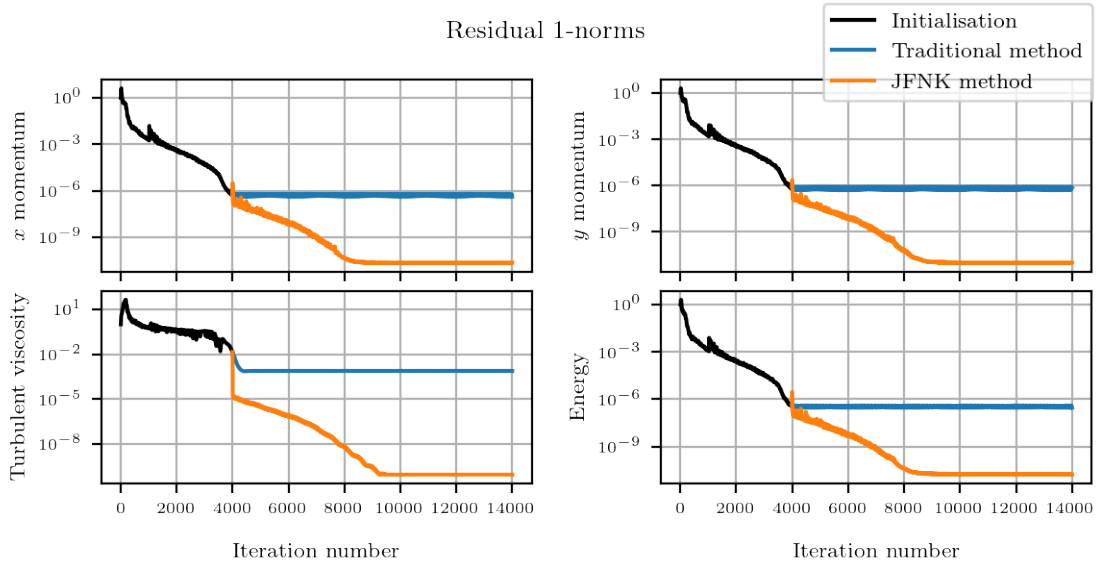


Figure 3.5: Residual 1-norms for the two momentum components, turbulent viscosity and energy throughout the computation for the turbulent transonic airfoil case. Values are normalised by the initial residual.

gives already satisfying results on such applications. The pressure coefficient around the airfoil is given in figure 3.3. Once again, the two curves are indistinguishable. Finally, the aerodynamic coefficients, the drag and lift coefficients, are given throughout the simulation in figure 3.4. This time, the JFNK method gives an improvement: the coefficients are much more stable, which means the convergence is better. The blue curve corresponding to the traditional method struggles to converge and seems to oscillate periodically. In comparison, the oscillation of the orange curve is dampened: this corresponds to convergence. The scale of the oscillation seen in figure 3.4 is not significant physically speaking, as it is negligible. It shows an issue with the traditional implicit method of the solver CEDRE: it fails to reach proper convergence but it oscillates around the solution. The new JFNK method however achieves proper convergence. Even if the figure shows an advantage of the new method, the result is almost meaningless as the oscillation of the blue curve is numerically insignificant. Looking at other data is necessary to properly conclude on the convergence of the two methods.

The residual is the best way to measure if a method can help the convergence of the solver. To define the residuals, let us use the partial differential equation (1.1). The residual is in fact the value of the function F from this equation. The name is quite adequate: for steady problems, the residual is what is left and still needs to be removed to find a steady solution. To decide on the convergence of a steady simulation, this residual norm is measured throughout the computational domain $\mathcal{D}$. More precisely, this residual is analysed component by component. For example, the residual 1-norm associated with the ith component is:

$$\|\mathrm{F}_i\|_1 = \int_{\mathcal{D}} |\mathrm{F}_i| \, \mathrm{d}v \tag{3.1}$$

and the ∞-norm is:

$$\|\mathrm{F}_i\|_\infty = \max_{\mathcal{D}} |\mathrm{F}_i| \, . \tag{3.2}$$

In this fluid dynamics application, the residual norm is associated with the conservative variables: the density, the momentum components, the energy and the turbulent eddy viscosity ν_t. With the Spalart–Allmaras compressible model, the conservative variable is in fact the product of the density and the Spalart–Allmaras variable $\rho\tilde{\nu}$.

Figure 3.5 shows the residual norms from both computations. The gain from using the JFNK method appears clearly. The final residual norm is much smaller for each and every conservative component. This means that the Jacobian-Free Newton–Krylov method is able to reach a better convergence level than the traditional method. In other words, the better Jacobian matrix approximation leads to a better convergence than the older poor approximation.

In this first test case, we looked at the residual norms as the fields computed by both methods were almost identical. Moreover, our method was even able to reach better convergence levels. This validates our method on a first simple case, albeit with turbulence modelling and a shock.

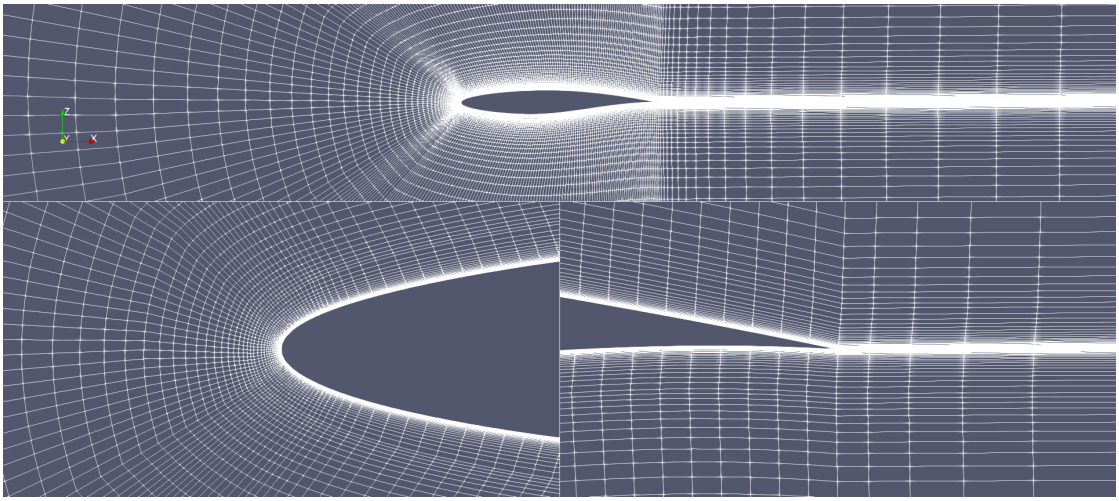


Figure 3.6: Finer mesh for the RAE 2822 test case. Close up on the leading and trailing edges.

3.1.2 Turbulent transonic airfoil with boundary layer resolution

Definition of the test case

The previous case uses a mesh that is too coarse to resolve the boundary layer but instead uses a turbulent wall model. It is representative of CEDRE applications, but not of actual applications from the aerodynamic community. The RAE 2822 airfoil is often used for turbulence model validations, but computations in the field prefer to use a finer mesh to resolve the boundary layer. This is what we present next. The new mesh, shown in figure 3.6 is clearly different. It is a three-dimensional mesh, which is in fact a two-dimensional mesh that was extruded on 6 cells in the y direction. Periodicity is set between the two parallel sides of the domain. The mesh is made of 198 144 cells whereas the previous RAE 2822 mesh was only made of 36 541 cells. With these additional cells, this mesh is able to be fine on the wall boundary condition as seen in figure 3.6. This means that the use of the turbulent wall model is no longer necessary, unlike in the previous test case. Turbulence closure is still provided by the Spalart–Allmaras one equation turbulence model. The scale of the mesh is the same: the chord is still equal to 1m.

The flight conditions are also modified to match some existing experimental conditions: the AGARD-AR-138 experimental database. The Mach number is then 0.734 and the angle of attack is 2.54°, with a Reynolds number of 6.5×10^6. This corresponds to *Case 9* from the experimental conditions. The parameters are not exactly the same as they are corrected to account for the effects of the wind tunnel [77].

Analysis of the results

As before, the computation starts from an initial state constant in the domain. Then the computation is continued for a few more iterations with the traditional implicit Euler method. This is to reach a state close enough to the steady solution more quickly before analysing the convergence of our methods. After that, the computations are restarted, with the traditional method on one side and the matrix-free method on the other. The residuals are given in figures 3.7 and 3.8. Overall, it takes much more iterations to decrease residual norms for this test case

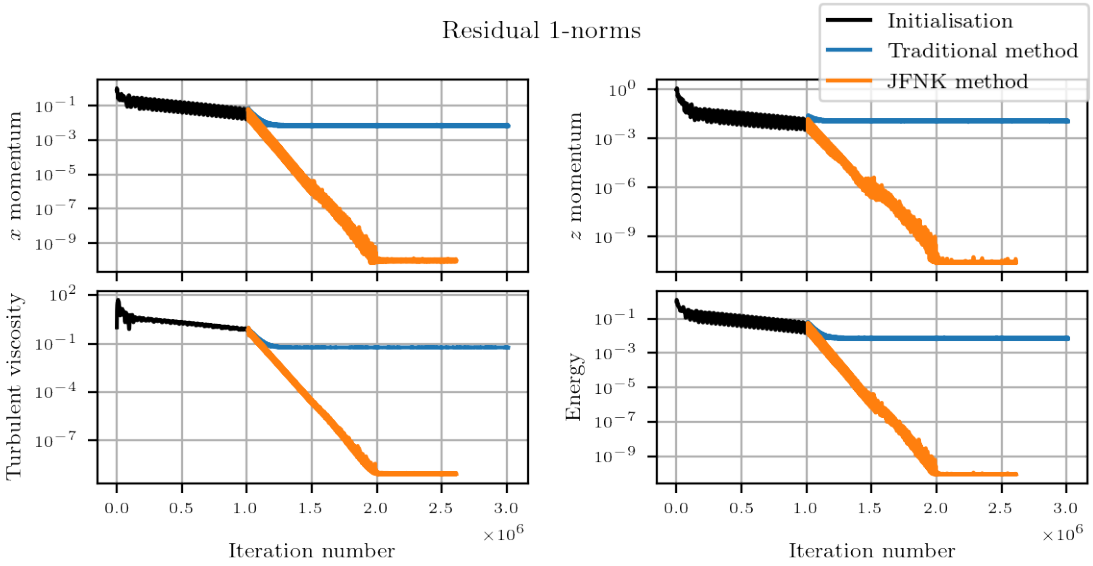


Figure 3.7: Residual 1-norms for the horizontal and vertical momentum components, turbulent viscosity and energy throughout the computation for the turbulent transonic airfoil with boundary layer resolution case. Values are normalised by the initial residual.

compared to the previous one. It is due to the larger number of cells, and the presence of small flattened cells in the boundary layer.

Figure 3.7 shows that the reduction in residual 1-norms with the standard method is not satisfying. Users usually expect the solver to reduce the order of magnitude of the residuals by 4 or 5, which is more than what our solver does. In contrast, the matrix-free method reduces residual norms much more efficiently and converges to a more precise solution. The quality of the solution computed by the matrix-free method is then better.

The ∞-norm of the residual is given in figure 3.8 for the same simulations. This time, the traditional method is not able at all to reduce residual norms. It means there are some cells in the computational domain for which the method is not able to reduce the local residual. The JFNK method in orange does not have this issue and converges to a better solution. The width of the curve is only due to data noise, as all lines use the same width.

The value of the turbulent variable $\tilde{\nu}$ in the first cell above the wing profile is provided for both computations in figure 3.9. The top views show the upper part of the airfoil, and the discontinuity due to the shock is visible, as expected. The bottom views show the lower part of the airfoil. The views on the right correspond to the JFNK computation, whereas the ones on the left to the computation with the traditional method. This traditional method using an approximated Jacobian matrix produces a solution of lower quality, as the pattern is not physical. In particular, there are cells for which $\tilde{\nu}$ is null in the computational domain, which should not happen with the standard Spalart–Allmaras model. Figure 3.8 shows that the ∞-norm of the residual on the conservative variable $\rho\tilde{\nu}$ normalised by the initial residual is near 10^0, so figure 3.9 highlights cells for which the residual is higher than $10^{-0.1}$. Those cells correspond to the ones

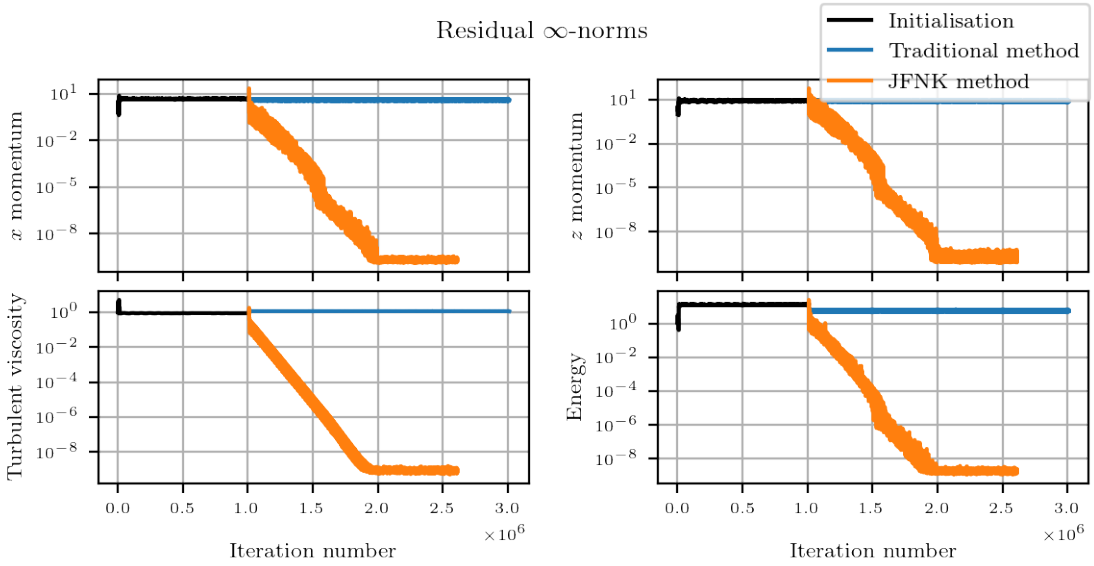


Figure 3.8: Residual ∞-norms for the horizontal and vertical momentum components, turbulent viscosity and energy throughout the computation for the turbulent transonic airfoil with boundary layer resolution case. Values are normalised by the initial residual.

for which $\tilde{\nu} = 0$. It shows a flaw in the standard time integration method, as it has non-physical features in the solution that prevent residual convergence. In contrast, it shows the quality of the JFNK method on this application.

3.1.3 Hypersonic reactive sphere

The second application we selected to compare the Jacobian-Free Newton–Krylov with the traditional one is the computation of the flow around a hypersonic solid sphere. Because of the high energy of the surrounding flow, the air molecules can separate and even form a plasma. The hypersonic reactive sphere is a well-known case, for both experimental [78] and numerical studies [79]. This is a simple yet representative test case of CEDRE applications. It then makes a lot of sense to analyse our new method on it.

Definition of the test case

The two-dimensional mesh is shown in figure 3.10. It is a regular mesh made of quadrangles, with refinement in the radial direction at the wall boundary and the expected shock location. The refinement at the wall boundary helps to compute more precisely the physical phenomena that happen at said boundary. The refinement at the shock is a requirement of the spatial discretisation methods. In order to get a clean and slim shock, the cells near the shock must have a high aspect ratio. The mesh takes this into account and is made according to CEDRE best practices. This expected shock location is obtained from a previous computation, made with a coarser mesh. This mesh is used in a two-dimensional axisymmetric computation to get the flow around a three-dimensional solid sphere.

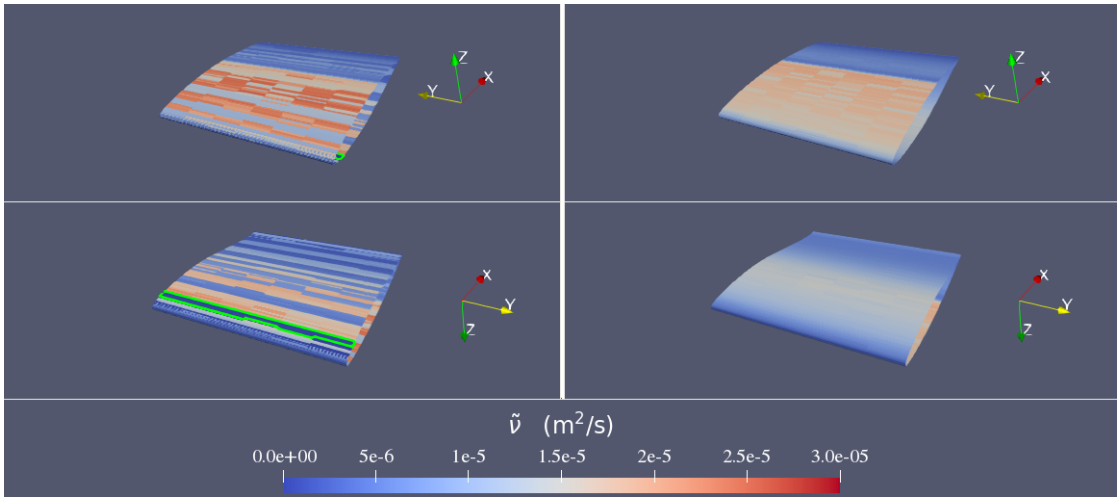


Figure 3.9: Spalart–Allmaras variable $\tilde{\nu}$ on the RAE 2822 wing profile for the traditional method (left) and the JFNK method (right). The top views show the profile from above, and the bottom views from below. The highlighted region in green corresponds to cells where the absolute value of the residual on $\rho\tilde{\nu}$ is higher than $10^{-0.1}$.

The solid sphere is modelled by an isothermal wall boundary condition. This is a representative choice as well, as the heat flux going through the wall is one of the main interests of such computation. Moreover, as it depends on the derivatives of the flow variables, it is usually harder to get a correct value for the heat transfer. A better convergence will lead to better derivatives, which will lead to better physical results for such case users. No turbulence model is used, as the flow is mostly laminar.

The main feature of this test case is that it simulates a hypersonic flow. At the left, the input boundary condition feeds air at Mach 15. This will induce a strong shock, meaning a strong discontinuity in the flow. It is indeed seen in figure 3.10 on the pressure and temperature. As this is a typical application of our solver, it is of importance to us that the new method behaves well with such flow features. When going through a strong shock, the temperature of the flow will increase a lot. The flow is made of air, or a mixture of 77% N_2 and 23% O_2. At the high temperature they reach after the shock, the molecules can decompose into N, O, and NO. They can even get ionised. The choosen model uses 11 possible species: N_2, O_2, N, O and NO, the corresponding cations N_2^+, O_2^+, N^+, O^+ and NO^+ and the electrons e^-.

We argued previously that our solver is quite complex and that it makes it hard to do correct computations for non-experimented users. This hypersonic reactive sphere is a perfect example of this statement. Even if this case looks simplistic, it should not be taken lightly. When using unfit spatial discretisation methods, we ended up with a well-known problem of a such simulation called carbuncle [80]. This phenomenon does not come from physics but is solely due to numerical methods. It is shown in figure 3.11 from some earlier computations that were using a different mesh. Simply put, it creates a recirculation bubble downstream of the shock, near the stagnation point. This bubble tends to push the shock farther from the sphere, and modify greatly the heat flux at this location. The choice of the Riemann solver ended up being the key element to getting rid of this undesirable effect: there is no recirculation with the AUSM+ scheme. There

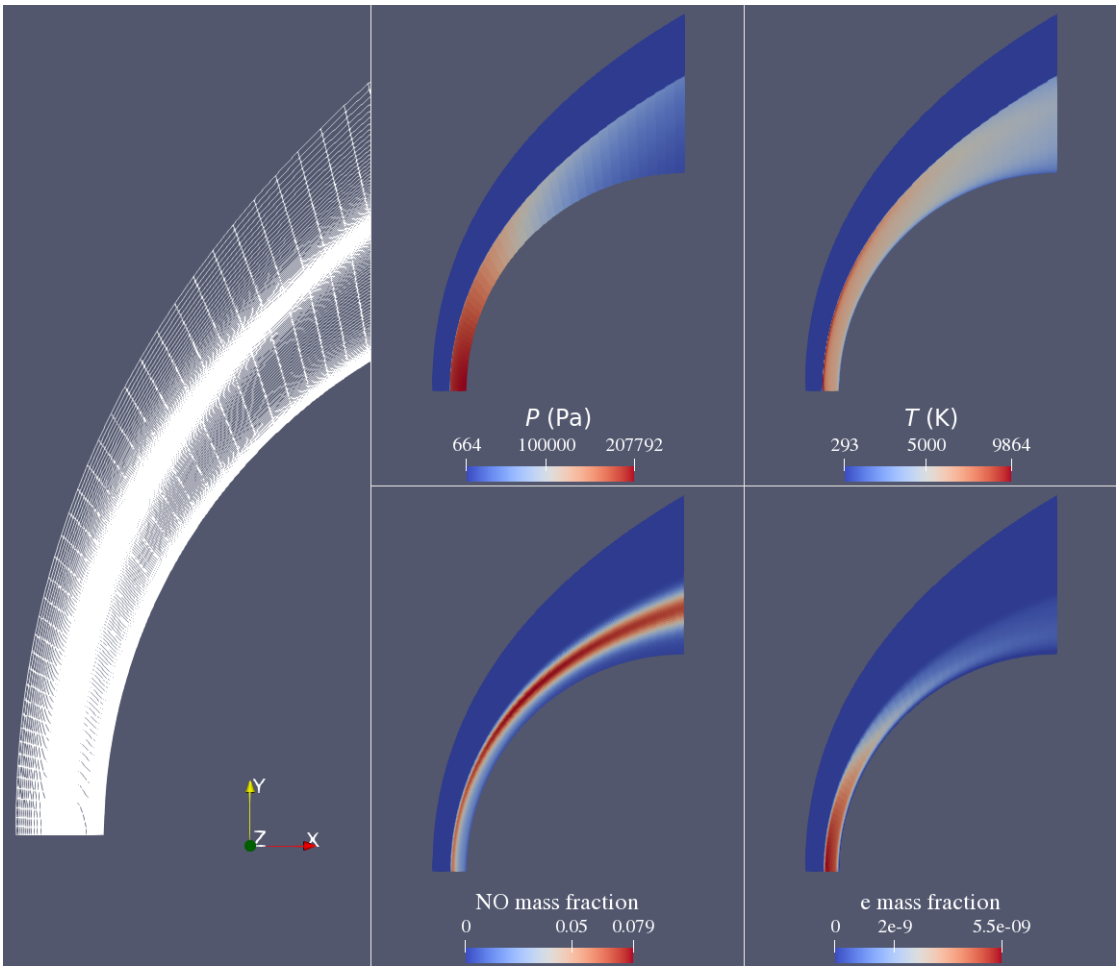


Figure 3.10: Mesh along with final pressure, temperature, and NO and e mass fractions for the hypersonic reactive sphere test case.

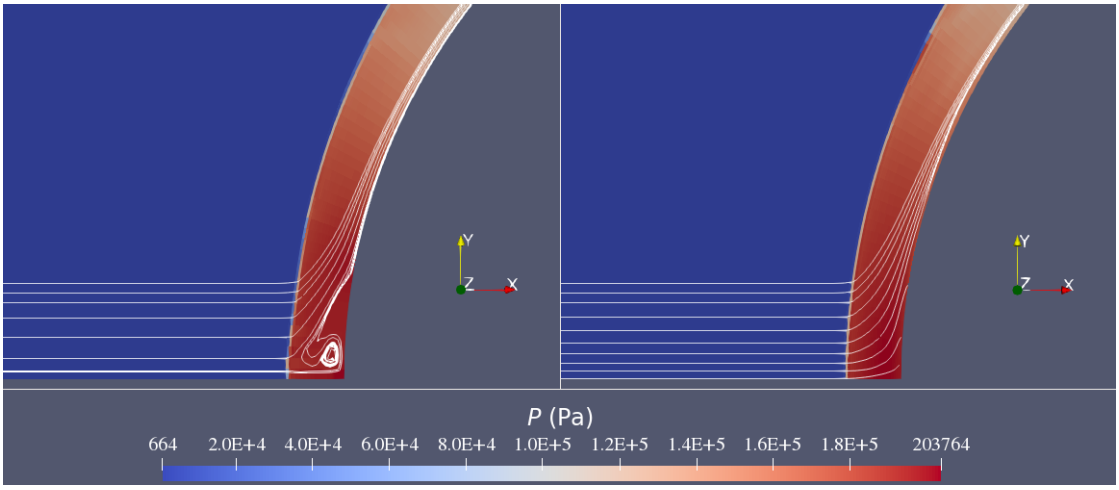


Figure 3.11: Streamlines near the stagnation point for two Riemann solvers.

are other pitfalls regarding this test case in CEDRE. For example, the user can choose whether to interpolate the mass fraction y_i for the species at the cell interfaces for the MUSCL scheme or the mass concentration ρy_i. Choosing the default option leads to nonconverging residuals no matter the time integration method. Indeed, this default option is recommended when running simulations with multiple phases, with significant variations in the density. Because CEDRE is made to solve a large variety of problems, it has a lot of methods that come with a lot of parameters, so even a simple simulation such as this one requires a lot of knowledge if a good convergence is required.

This test case is a typical application of CEDRE, as opposed to the previous one which focuses more on the aerodynamic properties. Indeed CEDRE does not try to be a pure aerodynamic solver but a multiphysics one, equipped to solve high-energy problems. Reentry phenomena are therefore in the scope of our solver. Any improvements coming from our method on such applications would benefit a lot of CEDRE users on their applications.

Analysis of the results

Once again, a first computation is done using the more robust traditional method. Indeed, the flow in the first cell against the sphere and the symmetry axis is initialised with a Mach number of 15 going straight into the wall. The matrix-free approximation struggles with such stiff initialisations, which is why the computation must start with another method, just until the shock has started to detach from the wall. Practically this amount to just a few iterations, compared to the number required to achieve convergence.

We first look at the different fields from the two computations. Once again they are similar and the residual norms are used to compare them, and this shows the simulation accounts for the features of interest. In figure 3.10, we see that the shock is present and that it falls as expected in the refined region. We see in the same figure the mass fraction of the various species downstream of the shock, which means the simulation is actively computing the chemical features of the flow, as expected.

To analyse the result from this case, one might once again look at the residuals. The first result corresponds to a computation without a reactive model. This simplification makes sense as the case to better understand the behaviour of the methods on an easier problem in terms of numerical complexity. It also means a different mesh is used, as not accounting for ionisation means the shock location is slightly different, but the two meshes are built the same way. The residual 1-norms are shown in figure 3.12. It shows that the new method gives a better convergence. The difference between the traditional method and the new one is that the traditional method uses the Jacobian matrix of the first-order spatial discretisation method, whereas the new one approximates the true Jacobian matrix using the approximation (1.53). This can indeed be verified. Because this simulation does not use complex models, the poor approximation used in the traditional model should use the Jacobian matrix of the first-order scheme. If the approximation (1.53) uses the first-order evaluation of f, then it should approximate this same Jacobian matrix. The evolutions of the residual norms from this method match the ones from the traditional method in figure 3.12. It first validates the development of the matrix-free approximation, and that the unbridle method should then use the actual second-order Jacobian matrix. It also confirms that the traditional method uses the Jacobian matrix of the first-order spatial discretisation method. Finally, it validates the choice of ε and the matrix-free approximation, as it is able to give the same result as when we use the Jacobian matrix.

With the new method, using the true function f with a second-order lead to much smaller residual norms. The only difference between this method and the previously existing one is that the latter uses the Jacobian matrix of the first-order spatial discretisation method, but the JFNK method takes into account the MUSCL reconstruction when using the Jacobian matrix. Because it uses a better Jacobian matrix in the linear problem, it gives a better solution to the nonlinear solver, which gives in turn a better time integration. It confirms that using a better Jacobian matrix helps the overall convergence when solving steady problems and validates the choice of the Jacobian-Free Newton–Krylov method.

When adding the reactive model, the conclusion is not in favour of the JFNK method anymore. The evolution of the residual norms is shown in figure 3.13. This time, the JFNK method does not longer find smaller residuals than the traditional method. In fact, when comparing figures 3.12 and 3.13, it appears that the traditional method converges better, enough so that it converges better than the JFNK method. This appears to happen because the mesh quality is better in this computation, compared to the mesh used without a reactive model. However, understanding the mesh quality is hard even on this simple test case for the reconstruction methods used in CEDRE. This is under current investigation by the responsible team. Still, the difference in the convergence of both methods is relatively small. A possible interpretation of this result is that when the mesh is not ideal for the spatial discretisation methods, using the JFNK method leads to a better convergence as it uses more precise Jacobian matrices for the linear system. When the mesh quality is better, the JFNK method is slightly less converged. Overall, it is still an improvement in the convergence on such test cases.

In a previous part, we suggested that the poor quality of the Jacobian matrix used in our solver was an obstacle to achieving good convergence. We then proposed an improvement: the Jacobian-Free Newton–Krylov method using the matrix-free approximation (1.53). We compared this new method with the traditional one on simple yet representative applications of our solver. This comparison showed that indeed, using a better Jacobian matrix leads to a lower residual norm. The new method can be useful when looking at quantities that require precise convergence, for instance, the derivatives of the flow variables. However, the main drawback of this new method

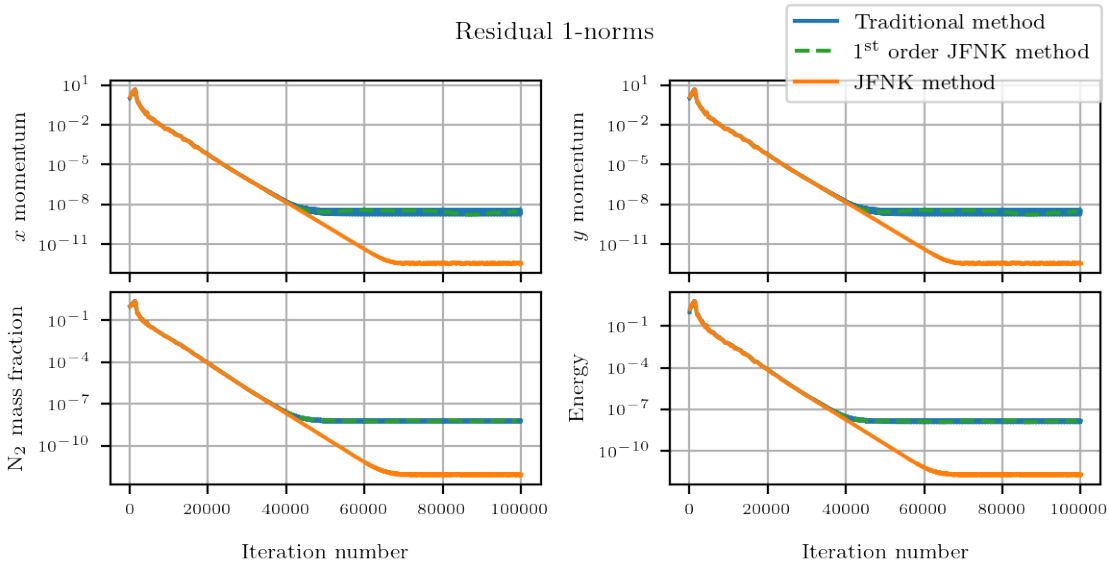


Figure 3.12: Residual 1-norms for the horizontal and vertical momentum components, N_2 volumic mass and energy throughout the computation for the hypersonic non-reactive sphere case. Values are normalised by the initial residual.

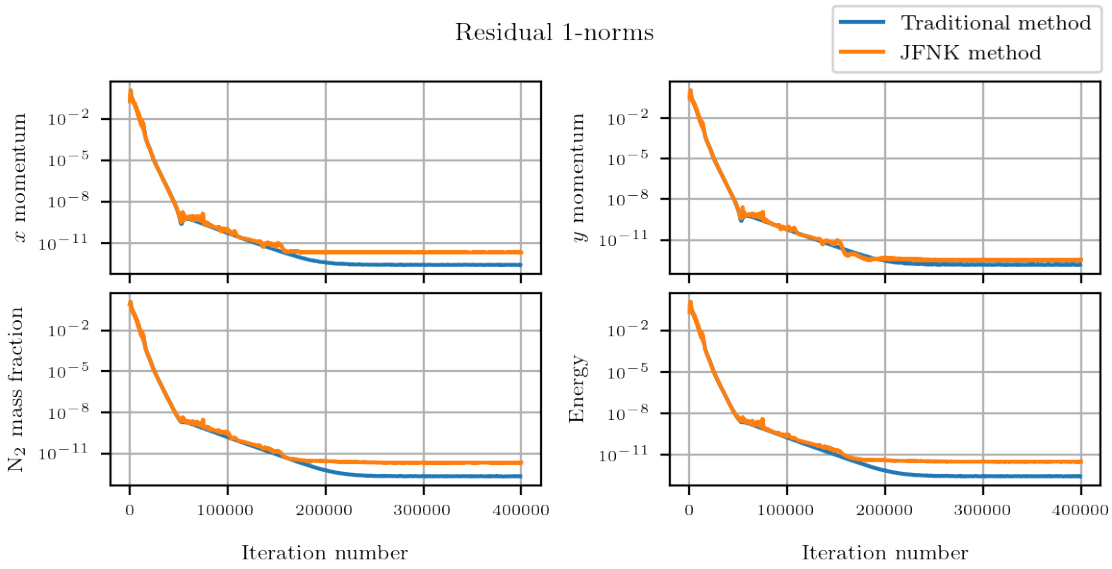


Figure 3.13: Residual 1-norms for the horizontal and vertical momentum components, N_2 volumic mass and energy throughout the computation for the hypersonic reactive sphere case. Values are normalised by the initial residual.

is its computational cost. This cost is not inherent to the method but comes from our solver properties as was discussed previously. The recommended usage is then to start a computation using the inexpensive traditional method, and then achieve better convergence using the more precise yet more expensive new method.

3.2 Using the matrix-free formulation with a new fluid model

In the previous section, the new Jacobian-Free Newton–Krylov method was compared to the traditional method that uses the first-order Jacobian matrix. This first-order matrix is easily computed for the usual Navier–Stokes equations. But as CEDRE is under constant development, new models are added in order to handle more finely various multiphysics phenomena. For example, a new model was recently added to better handle multiphasic flows [81]. Another model that we will investigate in the following was added to better account for thermodynamic disequilibriums. Adding a new model amount to writing the function F from equation (1.1), or equivalently G from equation (1.5). Usually, the development of this function is straightforward as it is given explicitly from the equations. However, the development of the Jacobian matrix is much more difficult. This is why the new models can not yet compute Jacobian matrices, and therefore can not use implicit methods as of today. The Jacobian-Free Newton–Krylov method is then a nice way to use implicit methods as it does not require the Jacobian matrix. Because the matrix-free approximation is written in a generic fashion, it can be used on any fluid model. In the following, we will use the JFNK method on a new fluid model that does not have an available Jacobian matrix and therefore can not use traditional implicit methods.

3.2.1 Multi-TEmperature model

The new model we will use is called the *Multi-TEmperature model*, or MTE. Its difference from the traditional Navier–Stokes model is that it allows for a thermodynamic non-equilibrium of some flow components. A particle has multiple degrees of freedom: translation, rotation and vibration. In more traditional models, it is assumed that all modes are at equilibrium, and they are grouped in what we call energy. One can define a time constant for each energy mode that corresponds to the number of collisions that are required to get to the equilibrium. A few collisions are required for translation modes, and about ten for rotation modes. This gives a small time constant of order 10^{-9}s. For vibration modes, however, up to 20 000 collisions are required. The corresponding time constant is significantly larger, and there may be regions where there can be a disequilibrium between vibrational energy on one side and translation and rotation energy on the other. The energy of such components can no longer be described with a single temperature. As electrons are much lighter, they move more than the other heavier flow components. Their transitional energy and the transitional energy of other components can not be described with the same temperature. With the new Multi-TEmperature model, the flow components may be divided into three classes:

- the ones that always are at the equilibrium
- the ones that may be at vibrational disequilibrium
- the electrons that are handled separately from the other heavy component.

To account for the disequilibrium, the Navier–Stokes equations (1.3) are modified into the following:

$$\begin{cases} \partial_t(\rho_s) + \nabla\cdot(\rho_s\underline{u}) = \nabla\cdot\left(\rho D_s\nabla y_s\right) + \dot{\omega}_s \\ \partial_t(\rho\underline{u}) + \nabla\cdot\left(\rho\underline{u}\otimes\underline{u} + (p+p_e)\,\underline{\underline{\mathrm{Id}}}\right) = \nabla\cdot\left(\mu\left(\nabla\underline{u} + \nabla\underline{u}^T\right) - \frac{2}{3}\mu\left(\nabla\cdot\underline{u}\right)\underline{\underline{\mathrm{Id}}}\right) \\ \partial_t(\rho_m e_{v,m}) + \nabla\cdot(\rho_m e_{v,m}\underline{u}) = \nabla\cdot\left(\lambda_{v,m}\nabla T_{v,m} + \rho e_{v,m} D_m\nabla y_m\right) \\ \qquad\qquad + S_m^{v-t} + S_m^{v-v} + S_m^{v-e} + \dot{\omega}_m e_{v,m} \\ \partial_t(\rho_e e_e) + \nabla\cdot((\rho_e e_e + p_e)\,\underline{u}) = \nabla\cdot\left(\lambda_e\nabla T_e + \rho h_e D_e\nabla y_e\right) + \underline{u}\cdot\nabla p_e \\ \qquad\qquad + S^{e-t} + S^{e-r} + S^{e-v} + \dot{\omega}_e e_e \\ \partial_t(\rho E) + \nabla\cdot((\rho E + p + p_e)\,\underline{u}) = \nabla\cdot\Bigg(\lambda_{eq}\nabla T + \sum_m \lambda_m\nabla T_m + \lambda_e\nabla T_e \\ \qquad\qquad + \mu\left(\nabla\underline{u} + \nabla\underline{u}^T\right)\underline{u} - \frac{2}{3}\mu\left(\nabla\cdot\underline{u}\right)\underline{u} \\ \qquad\qquad + \rho\sum_s h_s D_s\nabla y_s\Bigg) \end{cases} \tag{3.3}$$

The first noticeable feature is that the equations of this physical model are more complex compared to the standard Navier–Stokes equations. Furthermore, there are multiple energies corresponding to the multiple temperatures: e_e the energy of the electronic gas, $e_{v,m}$ the vibrational energy for each flow component that may be at disequilibrium, and E the total energy. It increases the number of conservative variables. The variables y correspond to the mass fraction of the different flow components. The source terms ω correspond to chemical production or decay. The source terms S correspond to the energy transfers between the different energy modes. The details of those terms can be complex and are not discussed here as it is not the subject of this thesis, but can be found in [82]. A noticeable feature of this model is the presence of a nonconservative term in the conservation equation of the electrons' energy: $\underline{u}\cdot\nabla p_e$. This term is due to the effect of the electric field. This is a known issue of this model, as the Finite Volume method handles better conservative terms. There are two ways of handling this issue. The first one uses electronic entropy instead of energy as a conservative variable [83]. To do so, it makes some simplifications concerning electronic dissipation terms. However, this simplification can lead to inaccuracy in the physical result, and it appears that using the nonconservative formulation is necessary to get physically accurate results [82, 84]. The other approach is the one used in our model, which handles the nonconservative term from equation (3.3). Not to go into too much detail, such models that account for thermodynamic disequilibrium are still under discussion as of today [85].

The Multi-TEmperature model allows for non-equilibrium between the vibrational mode of the flow components, the electronic energy and the total energy. Taking this non-equilibrium into account helps to compute more precisely physical quantities such as the electronic density, which can be useful for magnetohydrodynamics applications for example. It gives a better prediction of shock layer radiation with a better representation of the population of energy states which

helps the simulation of radiation cooling or the computation of wall heat fluxes. This model is most interesting on reentry problems, as there is indeed some non-equilibrium downstream of the strong shocks that appear on such problems. Taking the non-equilibrium into account can change the flow downstream of the shock and on the solid wall, and give more precise temperatures and chemical compositions. Generally speaking, the Multi-TEmperature model gives more precise results than the standard Navier–Stokes model on fast flow with low relative density where there may be some thermodynamic non-equilibrium, such as reentry problems or problems with the strong expansion of a plume.

3.2.2 Hypersonic reactive sphere

The hypersonic sphere test case is a perfect fit for the Multi-TEmperature model. There is some thermodynamic non-equilibrium in the region downstream of the shock, because of its intensity. Unfortunately, the Jacobian matrix for the new MTE model is not currently available. This limits the MTE model users to explicit time integration methods, and therefore small time steps for stability reasons. As they deemed explicit time integration too slow, they wanted to use the newly implemented Jacobian-Free Newton–Krylov method for their simulations. We will show in the following another interest of the matrix-free approximation: it allows for implicit time integration methods despite having no Jacobian matrix. The goal here is to be able to get a steady solution to the problem as fast as possible for the user. Initially, the MTE model users were using a second-order Runge–Kutta method: the Midpoint method. We will compare the JFNK method to this reference method, and we will look in particular at how much time is required from the user to get to a physically satisfying result.

Full reactive model

For the first step of our analysis, we will use the most complete physical model. We use the same 11 flow components as before: N_2, O_2, N, O and NO, the corresponding cations N_2^+, O_2^+, N^+, O^+ and NO^+ and the electrons e^-. We of course use the Multi-TEmperature model, and we consider that N_2 and O_2 may be at vibrational disequilibrium. Other components may in fact also be at disequilibrium and we could take them into account but the computational cost increase is not worth it. As they are less present, and they quickly get to the equilibrium relatively to N_2 and O_2, accounting for their disequilibrium would not be significant in the results [86]. This amount to having four temperatures to describe the flow: T_e for the electrons, T_{v,N_2} and T_{v,O_2} for the vibration modes of N_2 and O_2, and T for the total energy.

As before, the Jacobian-Free Newton–Krylov method needs some help at the beginning. This is why the computation starts with some iterations of the Midpoint method and uses the first-order Finite Volume method. Then, it continues with the JFNK method, still using the first-order Finite Volume method, to quickly get a good enough approximation of the expected solution. Finally, to get a finer result, the simulation switches the second-order spatial discretisation method. Local time-stepping based on the CFL number is used with the implicit simulation using the JFNK method. The GMRES method can not use preconditioners that require the matrix, so the only one available is the diagonal preconditioner based on the cell volumes. It is not ideal as it is extremely simple, and some physics-based preconditioner would be preferable [68] but none were available to us at the time. The spatial discretisation method used is the Multislope method, as it appears more robust than the k-exact method with the Multi-TEmperature model.

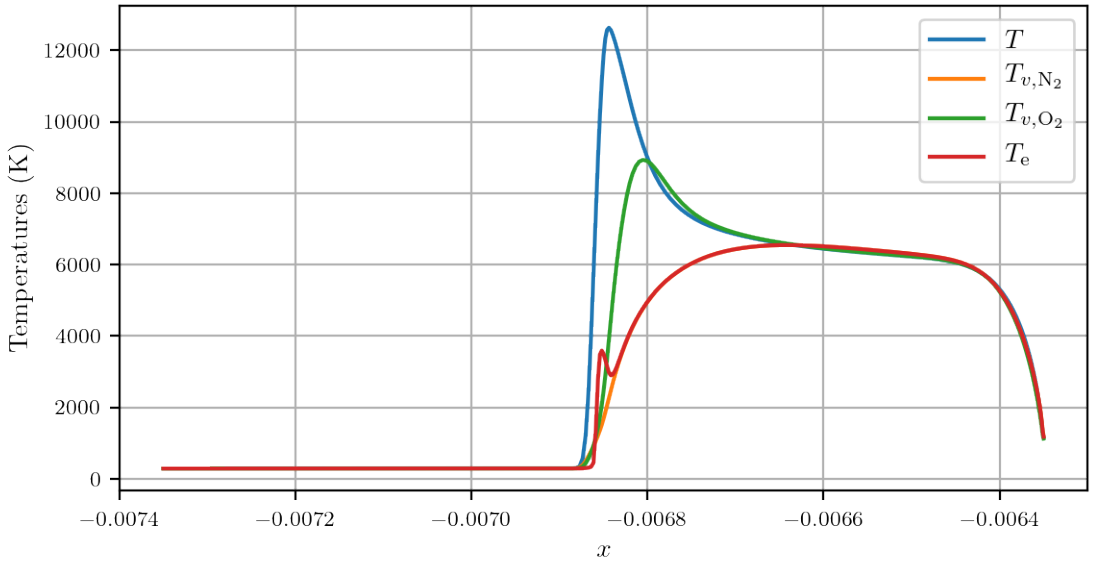


Figure 3.14: Temperature on the symmetry axis for the Multi-TEmperature model.

The goal of this thesis is not the physical analysis of this test case so it will be left out, but according to the MTE model users, the results are in agreement with the literature. Figure 3.14 the various temperatures along the symmetry axis. This justifies the use of the Multi-Temperature model: the region downstream of the shock is indeed in a thermodynamic non-equilibrium state.

We will also look at the residuals for this analysis. But this time, using the number of iterations as the x-axis would be unfair to the explicit method. Indeed the cost of one iteration is a lot smaller for the explicit method than for the implicit JFNK one. What matters to us and a typical user is not the number of iterations but the time spent waiting for the result. It is called the elapsed real time or wall-clock time. As the computations run each time in the same parallel environment, with a fixed number of CPU cores, the elapsed time is proportional to the CPU time. The x-axis will then be the elapsed real time: the actual time it took to get to the current residual.

Figure 3.15 shows the residual norms for the two computations. The leftmost black curve corresponds to the initialisation: a few iterations of the explicit time integration method. From then the computation can continue with the Midpoint method on one side and the Jacobian-Free Newton–Krylov method on the other. We see that the residual norm obtained with the JFNK method is lower than its explicit counterpart. Indeed, the explicit method is limited to small time steps because of the shock and the stiff reactive model. The implicit method however is not. The fact that the implicit method can use larger time steps is not new, but with this specific model, it was impossible to use implicit methods until the matrix-free approximation was added. The JFNK method even ends up being cheaper in terms of CPU time. This means that for a user, it is faster to use the newly implemented Jacobian-Free Newton–Krylov method than to use explicit methods. Our method gives a better alternative to users working on problems with the Multi-TEmperature model.

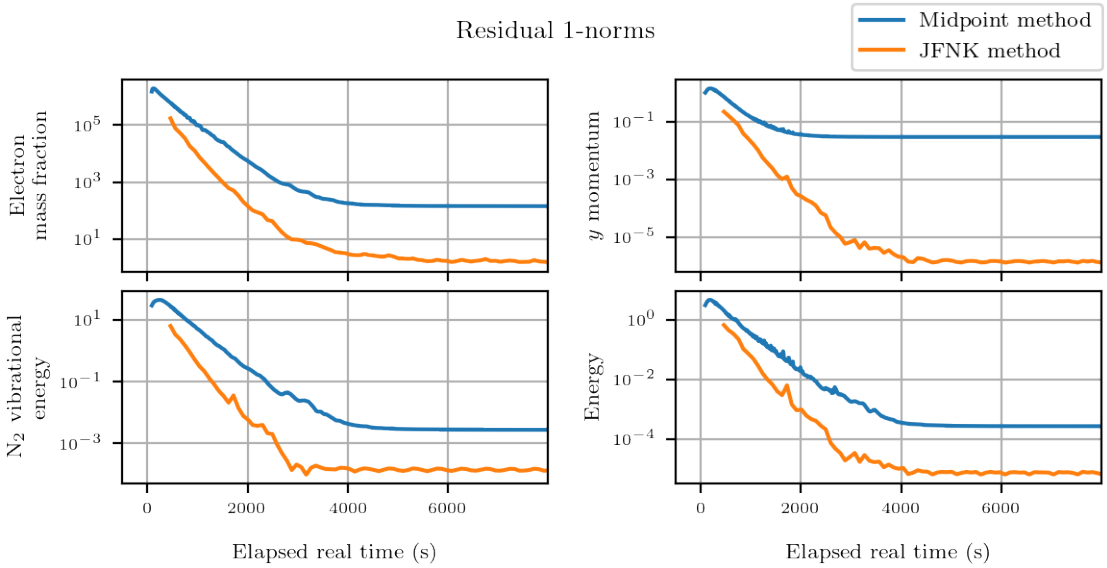


Figure 3.15: Residual 1-norms for the electron mass fraction, vertical momentum, N_2 vibrational energy and total energy throughout the computation for the hypersonic reactive sphere case using the Multi-TEmperature model. Values are normalised by the initial residual.

Simplified reactive model

The equations concerning electrons in the Multi-TEmperature model bring a lot of issues. It is because the quantity of free electrons is usually relatively small when compared to other flow components. It can lead to numerical instabilities or cause other problems. As the equations hold everywhere in the numerical domain, they hold in particular in areas without free electrons, upstream of the shock in our hypersonic sphere for example. The question one must ask is then how to compute quantities such as the electronic temperature T_e where there are no free electrons. We will not go into such implementation details as it goes beyond the scope of this work, but the point is that having electrons is quite troublesome for numerical methods. According to users that work on such reentry applications, it is not always useful to compute fluid ionisation. We explained why it might be crucial for some simulations, but the impact of ionisation on other quantities is almost negligible when the interest is not on the electrons. Sometimes, disregarding degrees of freedom corresponding to the electrons leads to simpler yet meaningful computations. The corresponding computation is drastically simplified as the contribution of the electrons is difficult to handle. The result however still gives interesting information, when one is not interested in the effects of ionisation. It is done by considering only the chemical components N_2, O_2, N, O and NO.

We decided to do the same comparison as before but this time with the simplified reactive model. The computation starts once again with a few iterations of the Midpoint method. Then, we compare the explicit method with the Jacobian-Free Newton–Krylov method. We show the corresponding residual norms in figure 3.16, while still using the elapsed real time as the x-axis. The result is similar to the previous one, except it is even more in favour of the JFNK method. The implicit method reaches much lower residual norms than the explicit one. With the full reactive model, the limiting factor for the implicit method came from the stiffness due

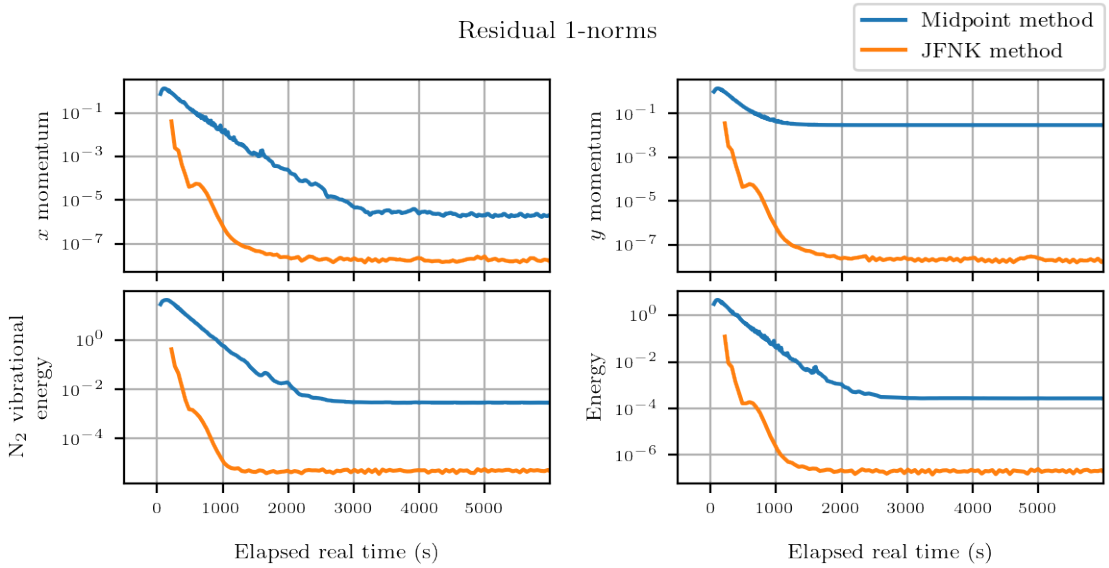


Figure 3.16: Residual 1-norms for the electron mass fraction, vertical momentum, N_2 vibrational energy and total energy throughout the computation for the hypersonic reactive sphere case using the Multi-TEmperature model without ionisation. Values are normalised by the initial residual.

to the electrons. Here, the term stiffness corresponds to both stiffness from the equation and difficulties arising from the implementation, that uses tools such as *min/max* functions that add non-differentiability. Without them, the convergence is significantly faster. However, the time step for the explicit method is under the same limitations with both reactive models. The gain in CPU time is then huge, in favour of the JFNK method.

Conclusion

This chapter compared the performances of the Jacobian-Free Newton–Krylov method with already existing methods of CEDRE on typical applications. When compared to the traditional implicit method on turbulent RANS computations, it showed that the JFNK achieves better convergence in the residual norms, when the convergence of the traditional method is unsatisfactory. The conclusion is similar when working with typical reentry applications. When the traditional method can converge well, however, the result is no longer in favour of the JFNK method, although the difference between the two methods is not significant. Finally, the matrix-free method was used with a newly implemented fluid model that does not give access to a Jacobian matrix yet. For this reason, using the JFNK method is the only way of using an implicit method. It was compared to the explicit Midpoint method that is currently used for computations with this fluid model. The results are once again in favour of the JFNK method, both in terms of convergence and speed. The superiority of the matrix-free method depends on the model's complexity level: the required time to reach the same convergence in residual norms is divided by 2 with the full model, and by 4 without ionisation.

Part II

Unsteady time integration using large time steps

Chapter 4

Introduction to exponential integration methods

Résumé du chapitre : Introduction aux méthodes d'intégration exponentielles

Le but de ce chapitre est de d'introduire un nouveau type de méthodes d'intégration temporelle.

Les méthodes classiquement utilisées dans la résolution des problèmes stationnaires sont souvent réutilisées après être adaptées dans la résolution de problèmes instationnaires intégrés avec de grands pas de temps pour s'affranchir des critères de stabilité de l'intégration explicite. Nous présentons dans ce chapitre le principe des méthodes d'intégration exponentielles, qui sortent de la dichotomie explicite/implicite. Nous décrivons ensuite plus en détail la famille des méthodes Rosenbrock exponentielles, qui seront utilisées par la suite. Nous réalisons également une brève analyse théorique de ces méthodes.

Comme leur nom l'indique, ces méthodes reposent sur l'exponentielle. Leur difficulté vient du fait qu'elles nécessitent de calculer des exponentielles de matrices, de très grande taille dans nos applications. Nous détaillons donc comment ce calcul est réalisé en utilisant des sous-espaces de Krylov.

Finalement, nous essayons une méthode exponentielle développée dans CEDRE sur un cas analytique simple. Nous montrons alors la pertinence de cette nouvelle méthode au vu de ses performances sur ce cas.

In this thesis, we were interested in finding solutions to steady problems. As we explained in the previous part, we use implicit time integration methods to efficiently get solutions to such problems. This is a pretty standard choice: most computational fluid dynamics solvers use implicit time integration methods to solve steady problems. The reason is that they can use larger time steps than their explicit counterparts. Because of this advantage, they are also often used when solving unsteady problems but with large time steps. Indeed, solving an unsteady problem with large time steps is similar to solving a steady problem. Most of what is done towards solving the steady problem can therefore be reused for unsteady problems with large time steps.

Even if implicit time integration methods are quite standard when solving problems with large time steps, there are other less conventional methods we could choose from. We could step out from the explicit implicit dichotomy, and decide to use IMplicit-EXplicit, or IMEX, methods. They split the function from the ordinary differential equation (1.5) into two parts: the stiff part that is integrated by an implicit method, because of its stiffness, and the other part that can just be integrated with an explicit method. The Additive Semi-Implicit Runge–Kutta methods, or ASIRK methods, are such methods [87]. They are already in use in computational fluid dynamics problems, such as fluid-structure interaction problems [88]. Previous work already implemented ASIRK methods in our solver CEDRE for specific multiphysics applications. Some methods are even less common and correspond to a total paradigm shift: the parallel time integration methods [89, 90]. Just as we classically split the computational domain over processes and compute the spatial discretisation method in parallel, parallel time integration methods decompose the time integration interval into subintervals. They then solve the ordinary differential equation on each interval concurrently then ensure the continuity between the subintervals. They then iterate with Newton's method to find the solution over the whole time integration interval. As a result, they can approximate accurately the solution at a later time without knowing accurately the solution at a previous time. Despite being nontraditional, they were successfully used to solve fluid-structure interaction problems or Navier–Stokes problems [91]. They were even more recently used to solve simple turbulent flow problems [92]. They can even be used in a more convoluted way, with for instance exponential methods [93]. Just as parallel time integration is inspired by parallel spatial discretisation, some other time integration methods are inspired by spectral discretisation methods. Time spectral methods, that were originally used for fluid dynamic time-periodic problems [94, 95], are now used on non-periodic problems [96]. But as both time parallel integration methods and time spectral methods are considered highly unconventional they require a major refactoring of the solver, which should be avoided for large-scale industrial solvers. In this part, however, we prefer to focus on another class of time integration methods.

4.1 Exponential integration methods

Let us focus on methods known as exponential methods. Despite being known for a long time [97], they were not widely used in computational fluid dynamics, because of some difficulties that are explained later, but still interested some scientists [98]. They started to come back in the literature [99, 100] and are now being used on applications similar to ours [101, 102]. They are still actively studied [103, 104].

The ordinary differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f\left(y\right) \tag{4.1}$$

is the same as the ordinary differential equation (1.5) from the previous part but with different notations. The main idea of exponential integration methods is to start from the ordinary differential equation (4.1) and split the function f into a linear part and a nonlinear part:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = Ly + N\left(y\right). \tag{4.2}$$

This decomposition is sometimes natural for some particular equations, but in the more general case it is always possible: it consists in choosing a linear part L and then setting the nonlinear part $N\left(y\right) = f(y) - Ly$. There are then an infinite number of decompositions, but we will see later that some are more interesting.

To define the time integration step, we start from the current estimate of the solution y_n that we assume to be exact: $y_n = y(t_n)$. We note $\Delta t = t_{n+1} - t_n$ as we work here with a fixed n. We can integrate equation (4.1) using the variation of constants formula to get:

$$y(t_{n+1}) = e^{\Delta t L} y_n + \int_{t_n}^{t_{n+1}} e^{(t_{n+1}-t)L} N(y(t))\,\mathrm{d}t. \tag{4.3}$$

The exponential integration methods then approximate the integral to compute the next value y_{n+1}. What defines the method is how it approximates this integral. For instances, methods that use the Taylor expansion of the nonlinear term are called exponential Taylor methods [105]. Here we focus on methods that use Runge-Kutta like quadratures. As we can see in equation (4.3), the linear part is treated exactly and the nonlinear one is approximated. If there is no nonlinear part, with $N = 0$, then the solution y_{n+1} is exact. If there is no linear part, with $L = 0$, then equation (4.3) transforms into

$$y(t_{n+1}) = y_n + \int_{t_n}^{t_{n+1}} N(y(t))\,\mathrm{d}t \tag{4.4}$$

and the exponential integration method behaves like a standard time integration method. This is the advantage of exponential integration methods: at best they are exact methods, and at worst, they are equivalent to traditional methods.

Similarly to classic methods, there are explicit [102] and implicit [101] exponential integration methods. It depends on whether it uses y_{n+1} or not to compute the integral from equation (4.3).

Before continuing, let us define some functions that are going to be convenient later on. Let the functions φ_k be defined by:

$$\begin{cases} \varphi_0 : z \mapsto e^z \\ \forall k \in \mathbb{N}^*, \varphi_k : z \mapsto \displaystyle\int_0^1 e^{(1-\theta)z} \frac{\theta^{k-1}}{(k-1)!} \mathrm{d}\theta. \end{cases} \tag{4.5}$$

They could also be defined using the recurrence relation:

$$\begin{cases} \varphi_0 : z \mapsto e^z \\ \forall k \in \mathbb{N}, \varphi_{k+1} : z \mapsto \dfrac{\varphi_k(z) - \varphi_k(0)}{z}, \end{cases} \tag{4.6}$$

or directly using their analytic formula:

$$\forall k \in \mathbb{N}, \varphi_k : z \mapsto \sum_{i=0}^{+\infty} \frac{z^i}{(i+k)!}. \tag{4.7}$$

This analytic formula ensures it is well-defined for squared matrices.

4.2 Exponential Rosenbrock–Euler method

4.2.1 Definition of the exponential Rosenbrock–Euler method

The exponential Euler method is the most basic one, and it can be used to understand exponential integration methods. Just as the Euler method assumes that $N(y)$ is constant and equal to $N(y_n)$ in equation (4.4), its exponential counterpart makes the same assumption but in equation (4.3). It then gives:

$$y_{n+1} = e^{\Delta t L} y_n + \Delta t \varphi_1(\Delta t L) N(y_n) \tag{4.8}$$

using the φ_1 function defined above, and finally:

$$y_{n+1} = y_n + \Delta t \varphi_1(\Delta t L) f(y_n). \tag{4.9}$$

Note the difference with the corresponding standard Euler method, for which the φ_1 function is replaced with the identity function. If there is no linear part in the decomposition, with $L = 0$, then nothing is treated differently by the exponential method, and the standard explicit Euler method is recovered.

In this method, the linearisation $L = f'(y_n)$ is assumed. Because of this choice, the method is called the exponential Rosenbrock–Euler method. We will discuss the nomenclature below. This choice feels natural and minimises the error of the method as is shown in the following. This choice is frequent among exponential integration methods, despite some methods using a fixed linearisation $L = f'(y_0)$. The difference is that a new Jacobian matrix is required at each iteration of the time integration method, but it is something we are used to with our traditional implicit methods. It is worth noting that we might want to try the implicit equivalent of this method by taking $N(y) = N(y_{n+1})$ in equation (4.3). However, because we took $L = f'(y_n)$, $N'(y) = 0$ and after a linearisation the two variants are equivalent.

4.2.2 Analysis of the exponential Rosenbrock–Euler method

From the definition of the method and equation (4.3), we have that the error made after one step is:

$$\begin{aligned} y(t_{n+1}) - y_{n+1} &= \int_{t_n}^{t_{n+1}} e^{(t_{n+1}-t)L} N(y(t)) \,\mathrm{d}t - \Delta t \varphi_1(\Delta t L) N(y_n) \\ &= \int_{t_n}^{t_{n+1}} e^{(t_{n+1}-t)L} \left(N(y(t)) - N(y_n)\right) \end{aligned} \tag{4.10}$$

We take the Taylor series of the nonlinear part:

$$N(y) = \sum_{i=0}^{+\infty} \frac{N^{(i)}(y_n)}{i!} (y - y_n)^i \tag{4.11}$$

and in particular, using the fact that

$$\begin{aligned} y(t) &= y_n + y'(t_n)(t - t_n) + O\left((t - t_n)^2\right) \\ &= y_n + f(y_n)(t - t_n) + O\left((t - t_n)^2\right) \end{aligned} \tag{4.12}$$

the partial series of order 2 is:

$$N\left(y\left(t\right)\right) = N\left(y_n\right) + N'\left(y_n\right) f\left(y_n\right)\left(t - t_n\right) + O\left(\left(t - t_n\right)^2\right). \tag{4.13}$$

The error of the method is then:

$$\begin{aligned} y\left(t_{n+1}\right) - y_{n+1} &= \int_{t_n}^{t_{n+1}} e^{(t_{n+1}-t)L}\left(N'\left(y_n\right) f\left(y_n\right)\left(t - t_n\right) + O\left(\left(t - t_n\right)^2\right)\right) \mathrm{d}t \\ &= \Delta t^2 \varphi_2\left(\Delta t L\right) N'\left(y_n\right) f\left(y_n\right) + O\left(\Delta t^3\right). \end{aligned} \tag{4.14}$$

The decomposition chosen earlier is now justified: if $L = f'\left(y_n\right)$ and therefore $N'\left(y_n\right) = 0$ then $y\left(t_{n+1}\right) - y_{n+1} = O\left(\Delta t^3\right)$ and the method is of order 2. This is a noticeable property of this method: despite being a single-step method it has a second-order of accuracy. A more rigorous analysis can be found in [106].

It is much harder to analyse the stability of this method, and more generally of exponential methods. Indeed, the stability analysis is done on the Dahlquist test equation (1.14). But as this equation is linear, exponential methods can solve them exactly no matter the time step size. We can still write the single step equation (1.19) $y_{n+1} = g\left(\Delta t J\right) y_n$ with $g\left(z\right) = e^z$, and get the corresponding stability region which is the left half complex plane. We could then conclude that the method is A-stable. The issue with exponential methods is that the standard stability analysis is no longer pertinent. We tried to do a better stability analysis for simple exponential methods such as the exponential Rosenbrock–Euler method, but we did not succeed. Some work in the literature does define some stability notions for exponential methods [107], but usually takes a linear N. It goes against the idea that all the linear part of f is in L and N is purely nonlinear. We did not find a stability analysis that was satisfying to us.

4.3 Exponential Runge–Kutta and Rosenbrock methods

When we introduced explicit methods, we used more convoluted approximations than the explicit Euler methods for the integral in equation (4.4). This gave the Runge–Kutta methods. The same can be done for the integral in equation (4.3) to get exponential Runge–Kutta methods. The difference with a standard Runge–Kutta method is that the quadrature coefficients are now functions of the matrix L:

$$\begin{cases} y_{n+1} = e^{\Delta t L} y_n + \Delta t \sum_{i=1}^{k} b_i\left(\Delta t L\right) N\left(y_{n,i}\right) \\ \text{with} \quad y_{n,i} = e^{c_i \Delta t L} y_n + \Delta t \sum_{j=1}^{i-1} a_{ij}\left(\Delta t L\right) N\left(y_{n,j}\right). \end{cases} \tag{4.15}$$

The Butcher tableau that we used to represent the Runge–Kutta methods is also used to represent the exponential Runge–Kutta methods. For instance, the Butcher tableau of the exponential Rosenbrock–Euler method, which is a special simple case of exponential Runge–Kutta methods, is shown in table 4.1.

Until now, we have not been exact when naming the methods we constructed here. From the literature, exponential Runge–Kutta methods are defined just as we did above, except they use

a fixed linearisation: $L = f'(y_0)$ [99]. It is because they were developed for semilinear parabolic problems:

$$\frac{\mathrm{d}y}{\mathrm{d}t} + Ay = r(y). \tag{4.16}$$

They are used when the remainder r is small or at least bounded in terms of A. This is not the case in our applications, and in many others so that is why another type of exponential method was introduced: exponential Rosenbrock methods [108]. They can be seen as a variation of the exponential Runge–Kutta methods where the linearisation is done at each time step. The issue is that a new Jacobian matrix is needed at each iteration, but once again we are used to this with implicit methods.

The nomenclature is a bit blurry to us, as the distinction between exponential Runge–Kutta and exponential Rosenbrock methods is not the same as the one between Runge–Kutta methods and Rosenbrock methods. First of all, the exponential Rosenbrock method is not implicit contrary to its standard counterpart. Furthermore, it would make sense to us to name an exponential integration method base on the underlying method used to approximate the integral from equation (4.3), and exponential Rosenbrock methods do not use Rosenbrock methods to do it. The only parallel we found was that the Rosenbrock methods do a linearisation at each inner stage, and the exponential Rosenbrock methods do a linearisation at each step of the time integration. It just seems that in the literature, the term *Rosenbrock* indicates that the linearisation is computed at each step instead of once at the beginning, with no apparent link with Rosenbrock methods.

Just as the Runge–Kutta method has order conditions for its quadrature coefficients, the exponential Rosenbrock method quadrature functions must follow some rules to ensure the expected order. To be consistent, it must have:

$$\sum_{i=1}^{k} b_i = \varphi_1. \tag{4.17}$$

It is reasonable to want multiple-stage methods to have a higher order than the single-stage exponential Euler method, and therefore it must also have

$$\sum_{j=1}^{i-1} a_{ij} = z \mapsto c_i \varphi_1(c_i z), \quad 1 \leq i \leq k \tag{4.18}$$

to achieve a second order. Using these two conditions, exponential Rosenbrock methods can be written in the form:

$$\begin{cases} y_{n+1} = y_n + \Delta t \varphi_1(\Delta t L) f(y_n) + \Delta t \sum_{i=2}^{k} b_i(\Delta t L) D_{n,i} \\ \text{with} \quad y_{n,i} = y_n + c_i \Delta t \varphi_1(c_i \Delta t L) f(y_n) + \Delta t \sum_{j=2}^{i-1} a_{ij}(\Delta t L) D_{n,j} \end{cases} \tag{4.19}$$

using $D_{n,i} = N(y_{n,i}) - N(y_n)$. This form shows that exponential Rosenbrock methods are variations of the exponential Euler method. The defects $D_{n,i}$ are small in size, so less computational effort can be spent when computing their contribution.

0	
	φ_1

Exponential Rosenbrock–Euler

0		
1	φ_1	
	$\varphi_1 - 2\varphi_3$	$2\varphi_3$

ExpRB32

0		
$\frac{3}{4}$	$\frac{3}{4}\varphi_1\left(\frac{3}{4}\cdot\right)$	
	$\varphi_1 - \frac{32}{9}\varphi_3$	$\frac{32}{9}\varphi_3$

ExpRB42

Table 4.1: Butcher tableau for some exponential Rosenbrock integration methods

Among exponential Rosenbrock methods, we looked at the ExpRB32 method from [106] and ExpRB42 methods from [109]. They were developed as adaptive methods, but we will use them without the error estimate as standard methods. They are both 2-stage exponential Rosenbrock methods and achieve respectively a third and fourth order of accuracy. This shows the quality of exponential Rosenbrock methods. Their Butcher tableaux are in table 4.1. For both methods, the coefficient that is the hardest to get is b_1, however equation (4.19) tells us that it will not be computed. We decided to work with those two methods, along with the Rosenbrock–Euler method, as they are simple among exponential methods and are supposedly well suited for our problems. They give us methods with orders of accuracy from 2 to 4. But what is interesting to us is that on linear problems those methods are exact, or with an infinite order of accuracy. As our problems are not purely linear, it means that the linear part will be solved exactly while the nonlinear part will be solved with a still good order of accuracy.

4.4 Evaluating matrix functions

At first glance, exponential methods may seem too good to be true. Indeed, they give explicit methods with few stages but high order of accuracy, and they have the formidable quality that they solve exactly the linear part of the ordinary differential equation. In other words, the exponential Euler method should be able to get the solution of Poisson's equation after any time with a single step. Indeed, Poisson's equation gives often a linear ordinary differential equation after the spatial discretisation, with the Finite Volume method on a regular mesh for example. The exponential Rosenbrock–Euler method can solve this linear differential equation exactly, for any time step, as large as it may be.

The reason why exponential methods are not that good is the reason why they were not used for a long time after being introduced. It is because evaluating matrix exponential, or more generally evaluating the φ-functions on matrices, is not trivial. Indeed, the power series definition of the matrix exponential is not well suited for numerical evaluations. In particular, in our context, computing the powers of the matrix is completely out of the question. The dimension of our matrices can get large, and the powers of the matrices would lose the sparsity. In fact, any approximation working directly with the matrix would not be satisfying. The question of how to evaluate φ-functions in the context of exponential methods is a subject of active research, as in this extremely recent reference [104]. We will however use more standard well-established algorithms.

The other issue with exponential methods is the question of the Jacobian matrix. Exponential Runge–Kutta reuse the same matrix throughout the computation but it is not a good idea in our applications since the nonlinear part can change. Exponential Rosenbrock methods were

introduced for this reason but they require the computation of a new Jacobian matrix at each iteration.

4.4.1 Evaluating matrix functions with Krylov subspace methods

The solution for both issues is the same: Krylov subspace methods. The Cayley–Hamilton theorem ensure that the exponential of a matrix is a polynomial of degree less or equal to N where N is the dimension of the matrix. Instead of looking for the matrix exponential as an infinite series, we can look for an N-order polynomial. Moreover, exponential methods do not need to evaluate the exponential or the φ-functions of the matrix, but their effect when applied to a vector. For instance, the exponential Rosenbrock–Euler method from equation (4.9) does not need to compute $\varphi_1(\Delta t L)$ but $\varphi_1(\Delta t L) f(y_n)$. It means we do not need to evaluate a given function h on the matrix A, but to compute the effect of $h(A)$ on the vector b. For exponential methods, h is a linear combination of φ-functions, and is therefore analytic. The idea, originally developed to approximate eigenvalues of a large sparse matrix, is to use a Krylov subspace method to approximate $h(A) b$ [110, 111]. In fact, it is similar to using a Krylov subspace method to solve the linear system $Ax = b$: it means taking $h = z \mapsto z^{-1}$. With the Arnoldi iteration, the Krylov subspace method produces at the m-th iteration the relation:

$$AV_m = V_{m+1}\tilde{H}_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T. \tag{4.20}$$

The columns of V_m, $v_1, \dots, v_m$, form an orthonormal basis of the Krylov subspace:

$$\mathcal{K}_m(A, b) = \text{Vect}\left(b, Ab, \dots, A^{m-1}b\right) \tag{4.21}$$

with $v_1 = b/\|b\|_2$. The matrix $\tilde{H}_m$ is a Hessenberg matrix and H_m is the square matrix equal to $\tilde{H}_m$ without its last line:

$$\tilde{H}_m = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & h_{m,m-1} & h_{m,m} \\ 0 & \cdots & 0 & h_{m+1,m} \end{pmatrix}, \quad H_m = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & h_{m,m-1} & h_{m,m} \end{pmatrix} \tag{4.22}$$

The vector v_{m+1} is the last column vector of V_{m+1}, added to the column vectors of V_m to get an orthonormal basis of $\mathcal{K}_{m+1}(A, b)$, and e_m is the m-th canonical vector $(0, \dots, 0, 1)^T$. Since H_m represent the compression of A on $\mathcal{K}_m(A, b)$ with respect to basis V_m and $b = \|b\|_2 V_m e_1$ with $e_1 = (1, 0, \dots, 0)$, we make the approximation [112]:

$$h(A) b \approx \|b\|_2 V_m h(H_m) e_1. \tag{4.23}$$

With this approximation, we can evaluate the function h by applying it to a smaller matrix. We reduced the original function evaluation to the much smaller Krylov subspace. Since the method does not need to know the matrix but only to be able to apply it as a linear operator, we can use the matrix-free approximation (1.53) to approximate the Jacobian matrix $L = f'(y_n)$. This way, having to compute a new Jacobian matrix at each iteration of the exponential Rosenbrock method is not an issue. Just as Krylov subspace methods for solving linear problems, this method can be restarted to limit the size of the Krylov subspace and therefore control the memory consumption of the algorithm. It is even possible to define an error estimate in the approximation to dynamically control the number of iterations needed to reach a given tolerance, albeit it is more convoluted than the residual used for solving linear systems with Krylov subspace methods [112].

Note: Since the matrix used to evaluate the quadrature functions of the exponential Rosenbrock methods is always the same, L, only one Arnoldi iteration is required for one step of the method. This could lead to a great cost reduction, however, we did not try it in our developments. Indeed, our work on exponential integration methods was more to demonstrate their feasibility in our applications, but it would be interesting to use this idea for future performance enhancements.

4.4.2 Evaluating matrix exponentials

With the Krylov subspace method just described, the next step is the computation of the exponential of the smaller Hessenberg matrix H_m. Attention is now paid to how to compute the exponential of relatively small dense matrices. Many methods use special properties of the matrices but our matrices do not share the same properties. Instead of using a truncated Taylor series, using the Padé approximant seems to be the best option [113]. The exponential function is approximated by the $[k/m]$ Padé approximant:

$$e^z \approx r_{k,m}(z) = \frac{p_{k,m}(z)}{q_{k,m}(z)} \quad \text{with} \begin{cases} p_{k,m} = \displaystyle\sum_{i=0}^{k} \frac{(k+m-i)!k!}{(k+m)!\,(k-i)!} \frac{z^i}{i!} \\ q_{k,m} = \displaystyle\sum_{i=0}^{m} \frac{(k+m-i)!m!}{(k+m)!\,(m-i)!} \frac{(-z)^i}{i!}. \end{cases} \tag{4.24}$$

It is best to use diagonal approximants, with $k = m$, as they are more accurate [113].

The final method used to compute the exponential of a dense matrix is the Scaling and Squaring method [114]. It computes an optimal parameter s and takes:

$$e^z \approx r_{m,m}(z/2^s)^{2^s}. \tag{4.25}$$

The goal is to choose s such as the error on the computation of the exponential is bounded, with minimal cost. It then decides on an optimal order m for the Padé approximant. The matrix is scaled by 2^s, the Padé approximant is used to compute the exponential, and the result is then squared back to the original scale. It is what gives its name to the method.

4.4.3 Evaluating the φ-functions

The exponential time integration methods do not only use matrix exponential, the φ_0 function, but may use any φ_k function. There is a nice relation that helps compute those functions. We first define the *augmented* squared Hessenberg matrix:

$$\bar{H}_{m+p} = \begin{pmatrix} H_m & c & 0 & \cdots & 0 \\ & 0 & 1 & \ddots & \vdots \\ & & 0 & \ddots & 0 \\ & & & \ddots & 1 \\ 0 & & & & 0 \end{pmatrix} \tag{4.26}$$

with a vector c of dimension m. Then, the following relation holds [111]:

$$\exp\left(\tau \bar{H}_{m+p}\right) = \begin{pmatrix} \exp\left(\tau H_m\right) & \tau\varphi_1\left(\tau H_m\right) c & \tau^2\varphi_2\left(\tau H_m\right) c & \cdots & \tau^p\varphi_p\left(\tau H_m\right) c \\ & 1 & \frac{\tau}{1!} & \ddots & \frac{\tau^{p-1}}{(p-1)!} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & \frac{\tau}{1!} \\ 0 & & & & 1 \end{pmatrix}. \tag{4.27}$$

The additional cost of using this augmented matrix is negligible in comparison with the cost of the rest of the methods. Furthermore, the methods we introduced before only need to augment with $p \leq 3$. By adjusting the scalar coefficient τ and reading the result in the appropriate column, we can finally compute the quadrature coefficients of our exponential Rosenbrock methods.

We explained how to compute one step of an exponential integration method. From what we have seen, the computational cost of one step of an exponential method is roughly the same as the cost of one step of an equivalent implicit method. For example, the computational cost of the exponential Rosenbrock–Euler method is the cost of one Jacobian matrix computation, one Arnoldi decomposition and then one function evaluation on the Hessenberg matrix. The implicit Euler method, which is also a one-stage method, also computes a Jacobian matrix and does an Arnoldi decomposition. It then inverts the Hessenberg matrix with a QR decomposition which is equivalent to applying a function to the Hessenberg matrix. This is why the computational costs of the implicit Euler method and the exponential Rosenbrock–Euler method are similar.

When we wrote the general formula for exponential Rosenbrock methods in equation (4.19), we used the defects $D_{n,i}$. The point of this formulation is to write the exponential Rosenbrock methods as corrections of the Rosenbrock–Euler method. Because the defects are usually small in size, we can spend less computational power to compute their contribution to the final step. Practically, it means we can use smaller Krylov subspaces to account for the defects. It means shorter Arnoldi iterations, which means smaller Hessenberg matrices, and overall a much smaller computational cost. We explained why the one-stage exponential method cost is equivalent to the one-stage implicit method cost. Because we use the formulation (4.19) and the fact that the defects are small, adding more stages to the exponential method will be less expensive than adding stages to the implicit method.

4.5 First developments in CEDRE

We first tried exponential methods in the solver CEDRE. The development was fast as it reuses a lot of existing parts from the implicit time integration: principally the Krylov subspace methods tools. The matrix used in the decomposition (4.2) can either be the traditional low-order Jacobian matrix or can use the matrix-free approximation (1.53). It shows that this work about exponential integration methods is compatible with our work about the Jacobian matrix, presented before. The Arnoldi decomposition (4.20) is computed by the same routines as the ones used for the GMRES algorithm. We only had to develop the Scaling and Squaring algorithm, with the computation of the Padé approximant to be able to use exponential Rosenbrock methods. Because it only required this development we decided to deviate from the original framework of this thesis and try this new type of time integration method. Now that everything was accessible, we started with the exponential Rosenbrock–Euler method, as it is the basic one.

To test the method, we decided to use a simple academic test case, adapted from the one proposed at the International Workshop on High Order CFD Methods (HiOCFD). It is a two-dimensional inviscid isentropic vortex that is convected in a periodic box. This vortex is a known exact solution of the unsteady Euler equations and is generally considered a good test case to analyse the accuracy of both spatial discretisation and time integration methods. It is a non-dimensionalised computation, and the mesh is regular over a square of length $L = 20$, with 100^2 cells. The vortex is convected in a flow at a Mach number of $\mathrm{Ma} = 0.5$, with the dimensioning parameters $P_\infty = 1$ and $T_\infty = 1$. The undisturbed velocity is then $U_\infty \underline{e}_x$ with $U_\infty = \mathrm{Ma}\sqrt{\gamma r_{\mathrm{gas}} T_\infty}$ and the vortex is defined using the polar coordinates system $(\underline{e}_r, \underline{e}_\theta)$:

$$\begin{cases} \underline{u} = U_\infty \underline{e}_x + \beta U_\infty \dfrac{r}{R_0} e^{-r^2/2R_0^2}\ \underline{e}_\theta \\ \\ T = T_\infty - \beta^2 U_\infty^2 \dfrac{\gamma - 1}{2\gamma r_{\mathrm{gas}}} e^{-r^2/R_0^2} \\ \\ P = P_\infty \left(1 - \dfrac{\beta^2 U_\infty^2}{T_\infty} \dfrac{\gamma - 1}{2\gamma r_{\mathrm{gas}}} e^{-r^2/R_0^2}\right)^{\gamma/(\gamma-1)} . \end{cases} \tag{4.28}$$

The heat capacity ratio γ is equal to 1.4, and the specific gas constant is $r_{\mathrm{gas}} = 1$. The vortex is defined by its characteristic radius $R_0 = 1$ and its intensity $\beta = 0.2$. This initial condition is shown in figure 4.1. As we solve inviscid Navier–Stokes equations, or Euler equations, the vortex moves in the $\underline{e}_x$ direction undisturbed, and as the domain is periodic the solution after one period $\Delta T = L/U_\infty = 33.8$ is the same as the initial solution.

We compare the exponential Rosenbrock–Euler method to the standard implicit Euler method and the classic fourth-order explicit Runge–Kutta method. Both the implicit and the exponential methods need to compute Jacobian matrices. To reduce the number of variables in this comparison, both will use the traditional Jacobian matrices which is the low-order approximation. We then end up with two equivalent methods in terms of computational cost. We simulate the vortex advection throughout one period ΔT. The solution for each method is shown in figure 4.1. We first look at the bottom center and right simulations. We see that the explicit Runge–Kutta method preserves well enough the vortex compared to the implicit method. It is because the RK4 method is a fourth-order method whereas the implicit method is a first-order method. However, the simulations are made at a CFL number of 0.9. With a higher CFL number of 1.5, the explicit

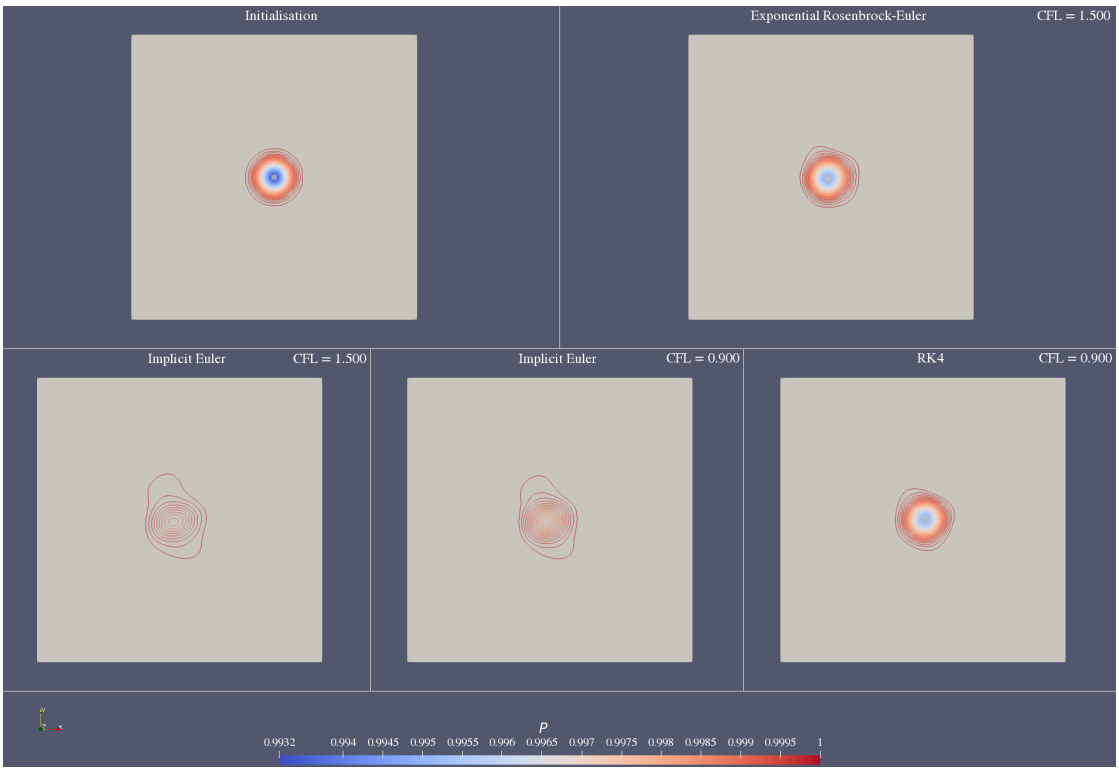


Figure 4.1: Initial solution (top left) and solution after one period for the exponential Rosenbrock–Euler method (top right), the implicit Euler method (bottom left and center) and the explicit RK4 method (bottom right). The first two simulations correspond to a CFL number of 1.5 whereas the last two to a CFL number of 0.9.

method is unstable and the computation fails. The implicit method is able to handle higher CFL numbers, of course as it is the reason implicit methods are used. But unsurprisingly we see in the bottom left figure that the solution is even worse. Finally, the exponential method is able to handle the higher CFL number and preserve the vortex well enough. It has the robustness of the implicit method and the precision of the higher-order explicit method.

The analysis of this simple test case is quite rudimentary. Indeed, we did not look finely at the error introduced by the time integration methods, but we visually compared the results. This work showed that at the cost of very few developments we have a method that could prove interesting for time integration with large time steps. Its purpose was not to precisely analyse exponential methods but to see their feasibility in our solver. Now that this preliminary test showed the quality of exponential time integration methods, we need to perform an actual analysis.

We see in figure 4.1 that even the methods that preserve the vortex, the exponential and the RK4 methods, do introduce some error. Indeed, the vortex after one period is not exactly the same as at the beginning, as it should be. We see that it loses in intensity, and the contour lines are not exactly concentric. Those symptoms are seen in both computations with the exponential and the higher-order explicit methods. Furthermore, the error in the result of those simulations looks similar. It is because this error does not come from the time integration method but from the spatial discretisation method. Indeed, the spatial discretisation method induces some error when evaluating the right-hand side of equation (4.1). This error is accumulated throughout the simulation, and because of it, we could not get the exact result no matter the time integration method. What we can do, however, is take a spatial discretisation method such as the error it adds is small compared to the error from the time integration scheme. This way, the error we get at the end of the computation is mostly due to the time integration method, and it allows us to compare methods. Here the spatial discretisation method used in CEDRE is a second-order method. We need to use a higher-order spatial discretisation method to perform the analysis of exponential time integration methods.

Chapter 5

Analysis of exponential integration methods in JAGUAR

Résumé du chapitre : Analyse des méthodes d'intégration exponentielles dans JAGUAR

Le but de ce chapitre est de comparer les méthodes exponentielles aux méthodes d'intégration classiques de JAGUAR pour valider leur intérêt.

Nous avons montré dans le chapitre précédent que CEDRE n'était pas le contexte idéal afin d'analyser des méthodes d'intégration temporelles très précises comme celles qui nous intéressent. Nous utilisons donc le solveur JAGUAR, basé sur la méthode des Differences Spectrales, que nous présentons au début de ce chapitre.

Nous analysons ensuite les méthodes exponentielles ajoutées à JAGUAR grâce à l'utilisation de la librairie SLEPc sur un cas simple, afin de constater l'ordre de ces méthodes à travers l'erreur globale en fin de simulation. Sur un autre cas académique, nous comparons ensuite notre ensemble de méthodes en regardant le temps d'exécution qu'il leur est nécessaire afin de réaliser une même simulation. Pour finir, nous utilisons les méthodes exponentielles sur un cas plus significatif car proche d'une application réelle du solveur: une aube de turbine.

We want to analyse time exponential integration methods in a framework that gives us access to high-order spatial discretisation methods. It is possible to use high-order Finite Volume methods, but this means using larger stencils which hurts parallelism. In CEDRE, users often stop at second-order methods, so we need to use another solver. As the spatial discretisation method does not play a direct role in the time integration and the work done in this thesis, we can step out from the Finite Volume framework. Furthermore, using a less complex solver will also help us try and develop new methods more easily than we already did with CEDRE. This is why we decided to accomplish our analysis of exponential time integration methods with the solver JAGUAR.

5.1 JAGUAR: a Spectral Difference solver

JAGUAR means proJect of an Aerodynamic solver using General Unstructured grids And high ordeR schemes. It is a reactive Navier–Stokes solver originally developed at CERFACS and has been jointly owned by ONERA and CERFACS for four years. It is made for unstructured grids and uses a Large Eddy Simulation model to solve unsteady turbulence effects. It means that the large turbulence scales are computed explicitly while the smaller ones are modelled. Its particularity is that it uses a spectral method as a spatial discretisation scheme: the Spectral Difference method. The Spectral Difference method follows some principles of the Discontinuous Galerkin formulation, in the sense that both are families of schemes of arbitrary order using polynomial approximations, but their mathematical foundations differ.

5.1.1 The Spectral Difference method

With the Spectral Difference method, the solution is represented by a polynomial of degree p inside each cell. It means that in the partial differential equation

$$\frac{\partial u}{\partial t} + \nabla \cdot F(u) = 0, \tag{5.1}$$

u is a p-degree polynomial of the coordinate variables, where the polynomial coefficients are functions of time. Then $F(u)$ has to be a $p+1$-order polynomial of the coordinate variables in order to have a p-order polynomial of the flux divergence. The key to the Spectral Difference method is how to compute a $p+1$-order $F(u)$ from a p-order u. This method uses key elements that were first mentioned by [115], and were later developed by [116].

Because the solution is represented in each cell by a polynomial, there is no reason for the solution to be continuous throughout cell interfaces. Indeed, the Spectral Difference method uses a piecewise representation of the solution. For example, let us consider the two-dimensional regular mesh, made of 2×2 cells over $[0,2]^2$. The exact analytical function u over the mesh in the left part of figure 5.1:

$$\begin{aligned} u\colon\ & [0,2]^2 \to \mathbb{R} \\ & (x,y) \mapsto \cos(5x)\tanh(5(1-y)) \end{aligned} \tag{5.2}$$

cannot be represented by polynomial exactly and the right picture of figure 5.1 represents the solution associated with the Spectral Difference method. The colour mapping corresponds to the output of the function but is not given as this function is of no particular interest: it is a rough example only. This function is interpolated with a second-order method using Lagrange polynomials over each cell. The result is shown in the right part of figure 5.1, where each colour corresponds to one cell. There are discontinuities at the cell interfaces, which shows a specificity of the Spectral Difference methods. If the function on the left part of figure 5.1 was used as an initial condition for a simulation, the Spectral Difference method would in fact use the polynomial interpolation on the right part. However, to respect the underlying conservation property of equation (5.1), it is necessary for F to be continuous over the whole computational domain. The Spectral Difference method will then make sure that the polynomial representations of the flux density are continuous throughout cell interfaces.

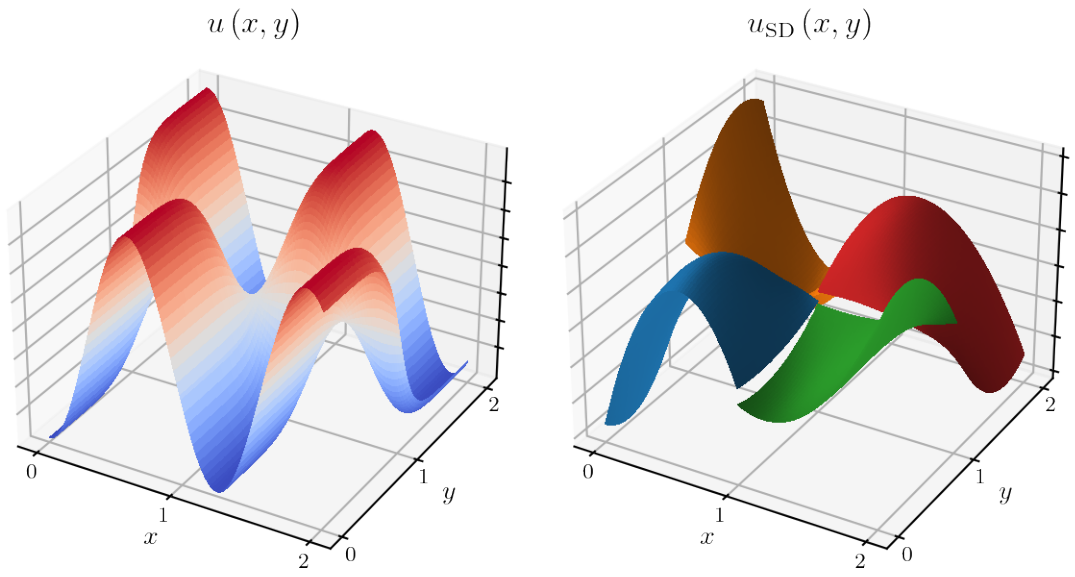


Figure 5.1: Continuous function (left) and discontinuous representation made by the Spectral Difference method (right).

To better understand how this method works, let us take a one-dimensional cell: the segment $[0, 1]$. Because the following is done at a fixed time, the dependency on the time is dropped, but the solution and the coefficients are in reality functions of the time, not scalars. The solution u inside this segment is then $u(x) = \sum_{i=0}^{p} a_i x^i$. Using Lagrange interpolation polynomials, it is equivalent to use the set of the $p+1$ coefficients a_i or the set of the $p+1$ values $u(x_i)$ computed at the distinct points $x_i \in [0, 1]$ called *solution points*. The solution as a p-order polynomial is represented by either one of those two sets. As the flux density F must be a $p+1$-order polynomial, it can also be represented by its values in the $p+2$ distinct *flux points*. To ensure that F is continuous at the segment end points, we take 0 and 1 as flux points. The choice of the rest of the flux points will be discussed later, but let us say for now that they are staggered with the solution points: each flux point, apart from the segment end points, is between two solution points and vice versa.

Figure 5.2 shows the steps the Spectral Difference make to compute the flux divergence $\nabla \cdot F(u)$ from the solution u, with the specific choice $p = 2$.

0. At first, the solution is represented with a red line by its value in the $p+1$ solution points. The solution points are marked by red dots. Parts of the solution from the left and right neighbouring cells can be seen. As the figure shows, the piecewise local solutions inside each cell may be discontinuous at the mesh interface. This is the starting point of the Spectral Difference method and will be called the 0-th step.

1. In the first step, the method computes the value of the solution in the $p+2$ flux points, marked by blue dots. As the polynomial representing the solution is known, this step just consists in evaluating it at flux points.

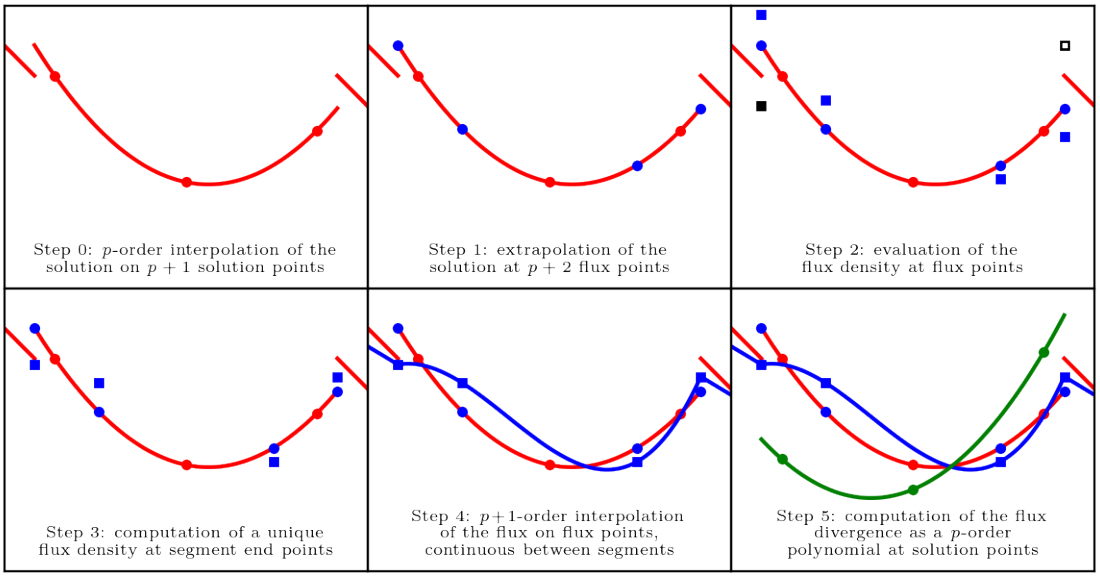


Figure 5.2: Steps of the Spectral Difference methods with $p = 2$.

2. In the second step, the method evaluates the flux density $F(u)$ in each flux point. This is possible because the solution was computed in those points in the previous step. The values of the flux density are marked by blue squares in the figure. However, segment end points are flux points for the two neighbouring cells, and therefore the figure shows a full black square for the flux at 0 from the left neighbouring cell and an empty black square for the flux at 1 from the right neighbouring cell to highlight this discontinuity.

3. Because there are different values of the flux density at the segment end points, the method would not preserve the conservative property of the partial differential equation. This is why in the third step, the method computes a unique interface flux density for both cells at each segment end point. The problem is to find the interface flux at the discontinuity of a piecewise solution. In other words, this is a Riemann problem. Once again, this is solved with a Riemann solver, exact or approximate, to get in the end a single value for the left and right parts of the discontinuity.

4. An interpolation from the value of the flux density at the $p+2$ flux points using Lagrange polynomials enables to get a p+1-order F as expected in the fourth step, represented with a blue line in the figure. Therefore, one can end up with a continuous representation of F, differentiable everywhere except at cell interfaces.

5. In the last step, the representation of F is differentiated to get the flux divergence, represented by a green line in the figure. Finally, a time integration method can use a p-order representation of $\nabla \cdot F$ to compute the solution at the next time step.

To work with any segment, not only $[0, 1]$, an isoparametric transformation is introduced to get back to this unity segment. It is the same when working with multiple dimensions, where the isoparametric transformation gets the cell back to the tensor powers of this segment. The placement of the solution points does not seem to matter much, but the placement of the flux

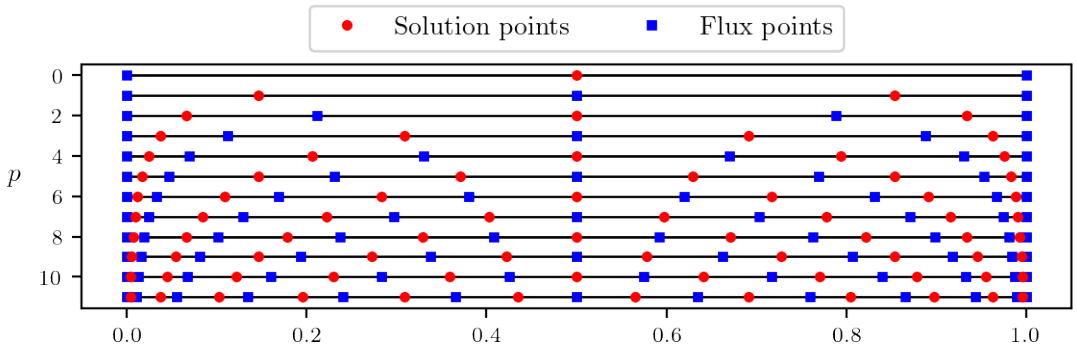


Figure 5.3: Solution points and flux points in the $[0, 1]$ segment used in the Spectral Difference method for $0 \leq p \leq 11$.

points does [117]. The $p + 1$ solution points we will use are defined in the $[0, 1]$ segment as the Chebyshev roots:

$$x_i = \frac{1}{2}\left(1 - \cos\left(\frac{2i+1}{2p+2}\pi\right)\right), \quad 0 \leq i \leq p. \tag{5.3}$$

They are traditionally defined inside the $[-1, 1]$ segment but are scaled into $[0, 1]$. They are the roots of the Chebyshev polynomials of the first kind and are often used in polynomial interpolation as they tend to minimise Runge's phenomenon. The $p + 2$ flux points are the p roots of the p-th Legendre polynomials to which are added the two segment end points. This choice has the property that there is a flux point between each contiguous solution point. There is no explicit formula for the Legendre polynomial roots as there is one for the Chebyshev polynomial roots. They are also defined in the $[-1, 1]$ segment and are scaled into $[0, 1]$. Finally, the solution and flux points can be seen in figure 5.3.

As a side note, the Spectral Difference method with $p = 0$ corresponds roughly to the first-order Finite Volume method. Indeed, the solution is assumed constant in the cell, represented by the value at its barycenter. The flux balance is made at the cell interfaces with a Riemann solver. This corresponds to the placement of the solution and flux points in figure 5.3 when $p = 0$. For any polynomial degree p of the solution, the proposed Spectral Difference method is naturally of order $p + 1$. Indeed, the method can represent exactly any polynomial of degree p and, as a consequence, the error term is of order $p + 1$. In JAGUAR, the polynomial order p ranges from 2 to 10, but $2 \leq p \leq 6$ for practical applications.

The demonstration of accuracy of the Spectral Difference method is proposed in a reference paper published recently [118]. JAGUAR was optimised to be efficient for parallel computations [119, 120, 121]. To perform Large Eddy Simulation, it is of paramount importance to implement an unsteady characteristic boundary condition: Navier–Stokes Characteristic Boundary Condition, or NSCBC. The initial formulation [122] was extended to deal with acoustic conditions and liner optimisation [123]. Recently, the solver was extended to handle $h - p$ adaptation [124] and the schemes were extended to triangles and tetrahedrons [125, 126]. Another improvement concerns the treatment of combustion, with a PhD thesis funded by Safran [127]. Finally, the last extension concerns the treatment of shocks [128]. This review shows the strong involvement of researchers in the solver.

5.1.2 Exponential integration methods in JAGUAR

Using a high-order spatial discretisation method and a refined mesh is the best-suited situation to test high-order time integration methods: the resulting error will come mostly from the time integration and not the spatial discretisation. The development of an exponential time integration method was inexpensive within CEDRE as the method reuses lots of already existing parts. For JAGUAR which only has explicit methods, the Arnoldi iteration and the exponential computation functions must be implemented to use exponential methods. They could be developed as internal stand-alone routines to produce a finely tuned method for JAGUAR. However, since the goal is to analyse exponential methods scientifically, it was decided to rely on the SLEPc library [129]. The Scalable Library for Eigenvalue Problem Computation, or SLEPc, is an extension of the software library PETSc [74, 75, 76]. Instead of rewriting the needed algorithms, it was chosen to use their SLEPc implementation. In particular, SLEPc handles what it calls *Matrix Function* objects or MFN:

> "Given a matrix A and a vector b, the call `MFNSolve(mfn, b, x)` computes $x = f(A)\,b$, where f is a function such as the exponential."

This is precisely what is needed to develop exponential methods: as it handles the previously introduced φ-functions with its MFN objects, SLEPc is then an obvious choice of a library for implementing exponential integration methods. Furthermore, because it relies on PETSc, it is efficiently scalable to fit our multiprocessing needs.

Exponential methods are based on a decomposition of the ordinary differential equation such as equation (4.2). However, JAGUAR uses only explicit time integration methods, so no Jacobian matrix is available. Computing analytically the Jacobian matrix would prove challenging as any ingredient should be differentiated, such as the Spectral Difference scheme, the Riemann solvers, the diffusion scheme, etc. It is not insurmountable but would amount to more work than what could be afforded during this thesis. It was decided instead to reuse the work from the previous part: the Jacobian matrix will not be formed, but its effect will be computed by a finite difference approximation. Another reason to work with the SLEPc library is that using a finite difference approximation is easy with it. More precisely, it is PETSc that handles this approximation. To sum up, to compute $\varphi_k(L)\,b$ that is required for the exponential methods with $L = f'(y)$, L is created with the appropriate PETSc data type. As it is a Jacobian matrix of a function, PETSc only needs to know the function to compute matrix-vector products. Then, after setting the MFN object of SLEPc, the library can compute the desired result. The Arnoldi iteration, the Scaling and Squaring algorithm and the exponential evaluation are all handled internally by the library. Overall, this procedure is extremely simple from our perspective and requires only a few lines of code to implement. This highlights the relevance of using the SLEPc library to easily test procedures and acquire the associated knowledge.

Other people that also work with JAGUAR did develop implicit time integration methods using PETSc. To continue in this direction, it was decided to add a time integration method based on the generic PETSc Time Stepper [71]. This way, a user may use most methods from PETSc in JAGUAR with no additional developments. It includes any explicit and implicit time integration methods, but also nonlinear solvers such as Newton's methods with various line search algorithms, linear solvers with various Krylov subspace methods and finally various preconditioning. However, this generic PETSc Time Stepper method was added on the side of this thesis, and therefore will not be discussed here, along with the previous work on implicit methods.

In the end, the high-order spatial discretisation method called the Spectral Difference method allows us to analyse and compare exponential methods that are available through the SLEPc library. In the following, attention is mainly focused on the three methods that were presented earlier in table 4.1: the Rosenbrock–Euler, ExpRB32 and ExpRB42 methods.

5.2 Analysis of exponential time integration methods

There are in JAGUAR all the tools necessary to analyse exponential time integration methods and several test cases must be chosen. During the analysis, the exponential methods and explicit Runge–Kutta methods will be compared, as the latest ones are the only available methods in JAGUAR and represent the state-of-the-art. Those methods are:

- RK2, the Midpoint method
- RK4, the classical four-stage fourth-order method
- RKo6s, a low-storage, low-dissipation and low-dispersion six-stage second-order method from [130] dedicated to unsteady computations
- TVDRK(3, 3), a three-stage third-order TVD method from [131, 132]
- SSPRK(5, 4), an optimal SSP five-stage fourth-order method from [133].

Low-storage methods are Runge–Kutta methods for which the matrix A in their Butcher tableau is lower diagonal, or in other words all a_{ij} are null except when $j = i - 1$. Also, all b_i are null except the last one that is equal to 1. It means that the final stage and all intermediate ones use only the previous stage. Therefore, there is no need to keep track of intermediate stages in memory, only to update the current value, which gives the name low-storage method. The last two methods are Runge–Kutta methods with an additional property originally called TVD [132] and more recently SSP [134]. Without going into too much detail, it means that they are methods that are convex combinations of explicit Euler steps so that their stability is guaranteed for sufficiently small time steps. To us, it means they are defined differently than other methods, using the coefficients $\alpha_{0<i\leq k,0\leq j<i}$ and $\beta_{0<i\leq k,0\leq j<i}$ as:

$$\begin{cases} y_{n+1} = y^{(k)} \\ \text{with} \quad y^{(i)} = \displaystyle\sum_{j=0}^{i-1} \alpha_{ij} y^{(j)} + \Delta t \beta_{ij} f\left(y^{(j)}\right), \quad 1 \leq i \leq k \\ \text{and} \quad y^{(0)} = y_n. \end{cases} \tag{5.4}$$

However, they are equivalent to traditional Runge–Kutta methods in the sense that they can be represented by the standard Butcher tableau, with:

$$\begin{cases} a_{ij} = \displaystyle\sum_{l=j+1}^{i-1} \alpha_{i-1,l-1} a_{l,j} + \beta_{i-1,j-1} \\ b_i = \displaystyle\sum_{l=i+1}^{k} \alpha_{k,l-1} a_{l,i} + \beta_{k,i-1}. \end{cases} \tag{5.5}$$

We can finally say that we are indeed working with explicit Runge–Kutta methods, as they were introduced initially.

5.2.1 Order analysis: convected inviscid isentropic vortex

This first case is designed to analyse the order of accuracy of the proposed methods and to validate their implementation. It is the simple case of the two-dimensional inviscid isentropic vortex, the same as the one considered with CEDRE. The case is still described by equation 4.28, but the numerical values are different. The convective flow Mach number is still 0.5, but this time $P_\infty = 1 \times 10^5$Pa and $T_\infty = 300$K. The heat capacity ratio is $\gamma = 1.4$ and the specific gas constant is $r_{\text{gas}} = 287.058\,\text{J}\,\text{kg}^{-1}\,\text{K}^{-1}$. Those values correspond to standard values for dry air. The mesh represent the two-dimensional box $[0, L]^2$ with $L = 0.1$m, and is made of $N \times N$ cells. The vortex is defined by its characteristic radius $R_0 = 0.005$m and its intensity $\beta = 0.2$. It corresponds in fact to the same vortex as the one used with CEDRE, in size and in intensity, but it has been scaled down into a smaller box and dimensioned to some given pressure and temperature. The present choice is in agreement with the prescription of the International Workshop on High Order CFD Methods. For this application, the Spectral difference method uses the approximated Riemann solver of Roe.

The period for those numerical values is $T = L/U_\infty = 5.760 \times 10^{-4}$s. After 20 periods, the vortex should recover the initial position and the 2-norm of the error between initial and computed solutions can be defined for any scalar variable $u(\underline{x}, t)$:

$$\text{err}(u) = \left(\int_{[0,L]^2} \left(u(\underline{x}, 20T) - u(\underline{x}, 0) \right)^2 \text{d}\underline{x} \right)^{1/2}. \tag{5.6}$$

Once the vortex moves in the domain, the numerical solution is subject to two schemes and errors are a consequence of both. Once the spatial scheme is defined, the mesh is sufficiently refined in order to transport the vortex accurately and the goal is to have a much lower influence of the spatial scheme than of the time integration scheme on the total error. The order of our time integration methods can be determined by looking at this error as a function of the time step.

The error at the end of the simulation is the global truncation error: the error the method makes while getting to a fixed time, no matter how many iterations it took. However, the order of a time integration method was previously defined using the local truncation error: the error it makes after a single small step. There is a link between global and local truncation errors for our single-step methods. Let us call τ_n the local truncation error and e_n the global truncation error at step n. Because single-step methods can be written as:

$$y_{n+1} = y_n + \Delta t_n g(y_n, \Delta t_n), \tag{5.7}$$

with an increment function g that is K-Lipschitz continuous in the y variable, the global truncation error is bounded [135]:

$$|e_n| \leq \frac{\max_{1 \leq i \leq n} |\tau_i|}{K \Delta t_n} \left(e^{K(t_n - t_0)} - 1 \right). \tag{5.8}$$

A method is a p-order method if $\tau_n = O\left(\Delta t^{p+1}\right)$. Therefore, a p-order method verify $e_n = O\left(\Delta t^p\right)$. However, the reciprocal is not true: observing a global truncation error $e_n = O\left(\Delta t^p\right)$ does not mean the order of the method is p. As a consequence, attention is focused on the expected order for the global truncation error at the end of the computation, as it is the error users look at. The proof of the order for a method is analytical, by writing the partial Taylor series of the exact solution, as we did earlier for the exponential Rosenbrock–Euler method.

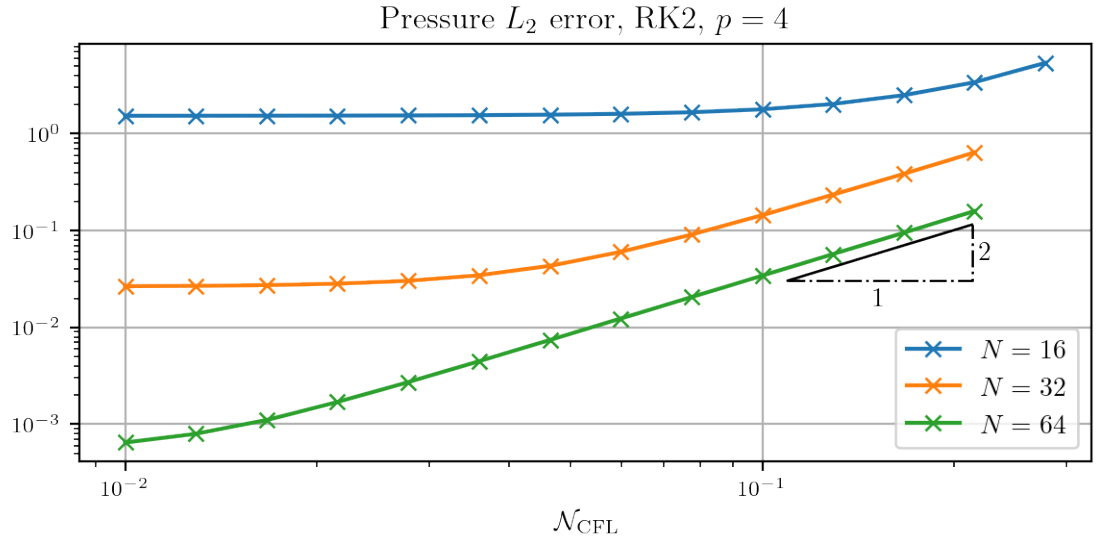


Figure 5.4: Global truncation error for the RK2 Midpoint method.

The point of this analysis is to see how the global truncation error depends on the time step Δt that is kept constant throughout the computations. Because some methods are more accurate than others, several computations with different mesh resolutions and Spectral Difference method orders were made. Let us start with the Midpoint method, associated with the order of the spatial discretisation method $p = 4$. The computation is stopped after 20 periods and the L_2 error for the pressure is analysed for different time steps. Results are shown using the CFL number as it is proportional to the time step:

$$\mathcal{N}_{\mathrm{CFL}} = \frac{N\,(p+1)\,(1+\mathrm{Ma})\,\sqrt{\gamma r_{\mathrm{gas}} T}}{L}\Delta t. \tag{5.9}$$

Using the CFL number instead of the time step allows comparing results with different N or even p. It can be seen here as a nondimensional time. Computations are made for multiple values of N: 16, 32 and 64, and the corresponding results are shown in figure 5.4. The 16×16 mesh does not allow for a good analysis: the error due to the spatial discretisation method is too important and prevents seeing the expected dependency between the global truncation error and the time step. The two other meshes however show the expected relation: $\mathrm{err} = O\left(\Delta t^2\right)$.

This analysis is repeated for other methods, and we look now at the RK4 method. However, figure 5.5 shows that the error is constant no matter the value of N. It is because the error due to the RK4 method is so small that it is dominated by the error due to the spatial discretisation scheme. It is useless to do the same for lower time steps, as the error would not change, therefore the RK4 error lines are continued by black dashed lines. As the time step decreases, the curves from the Midpoint scheme tend to the same dashed lines: this is proof that this lower bound for the error corresponds to the error of the spatial discretisation method. It shows once again that a spatial discretisation method with small errors is necessary in order to analyse time integration methods.

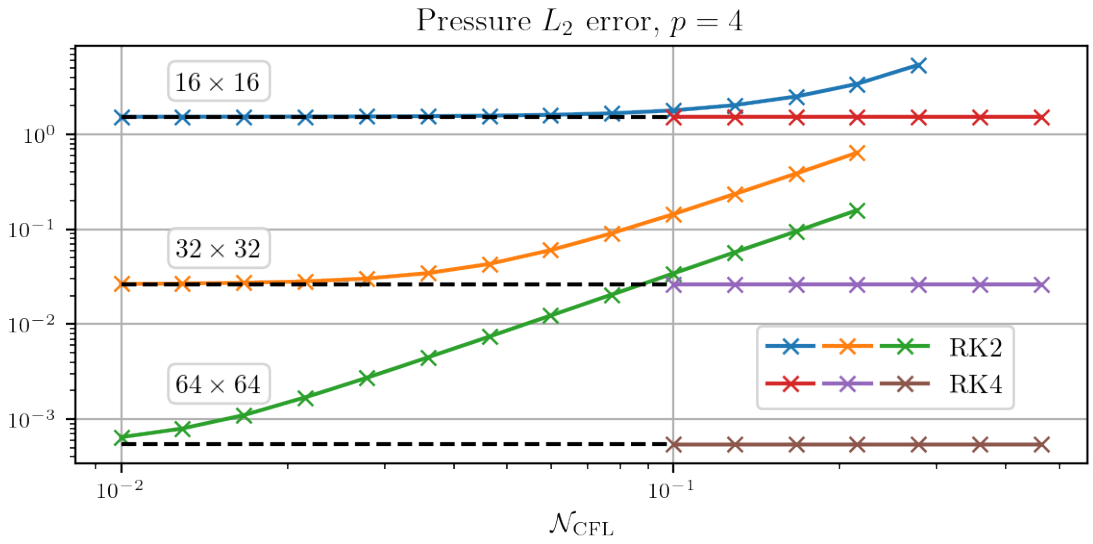


Figure 5.5: Global truncation error for the RK2 and RK4 method.

Increasing N up to 128 and then 256 was unsuccessful: the error was still constant. Instead of refining the mesh, an analysis playing with the Spectral Difference method can be performed by increasing the spatial discretisation order. As the theory behind this analysis does not directly depend on N and p, we can adjust them to better fit the method we want to analyse. This is done in figure 5.6 where the global truncation error is shown as a function of the time step, or CFL number, for the RK4, RKo6s and TVDRK(3, 3) methods. The expected slopes for the second and third-order methods are obtained but the result is more subtle for the fourth-order method. The error is once again dominated by the error of the spatial discretisation scheme. On the right part of the curve, one can guess the correct slope but it is necessary to plot the error for larger time step values to see it more clearly. However, those larger time step values are outside the stability domain of the method. In fact, all curves end on the right at the largest time step for which the computation did not fail. It means from figure 5.6 that the RKo6s method is more stable than both the RK4 and TVDRK(3, 3) methods. This is the issue with this analysis: for too small time steps, the error from the spatial discretisation method is dominant and the expected slope is not recovered. On the other hand, with too large time steps, computations fall outside the stability region of the method. The difficulty is to find the correct window that allows us to observe the desired slope by choosing suitable values for p and N. Because the error of the SSPRK(5, 4) is particularly small, it was not possible to find reasonable values below $p = 8$ and $N = 64$. Above those values, the computations get rather long so we skipped this method.

The analysis of traditional explicit Runge–Kutta methods can now be applied to the newly added exponential Rosenbrock methods. Their corresponding error curves are shown in figure 5.7. As expected, the respective slope values are 2, 3 and 4. Furthermore, by looking at the abscissa ranges, they work correctly with higher CFL numbers than all the explicit methods that were tested. The goal of this first analysis of the exponential integration methods is not to analyse their robustness, but it already seems better than with explicit methods. What is interesting from this analysis is that one can use higher-order methods with fewer Runge–Kutta stages. Furthermore, the additional computational cost of the additional stages is low. As discussed earlier, exponential

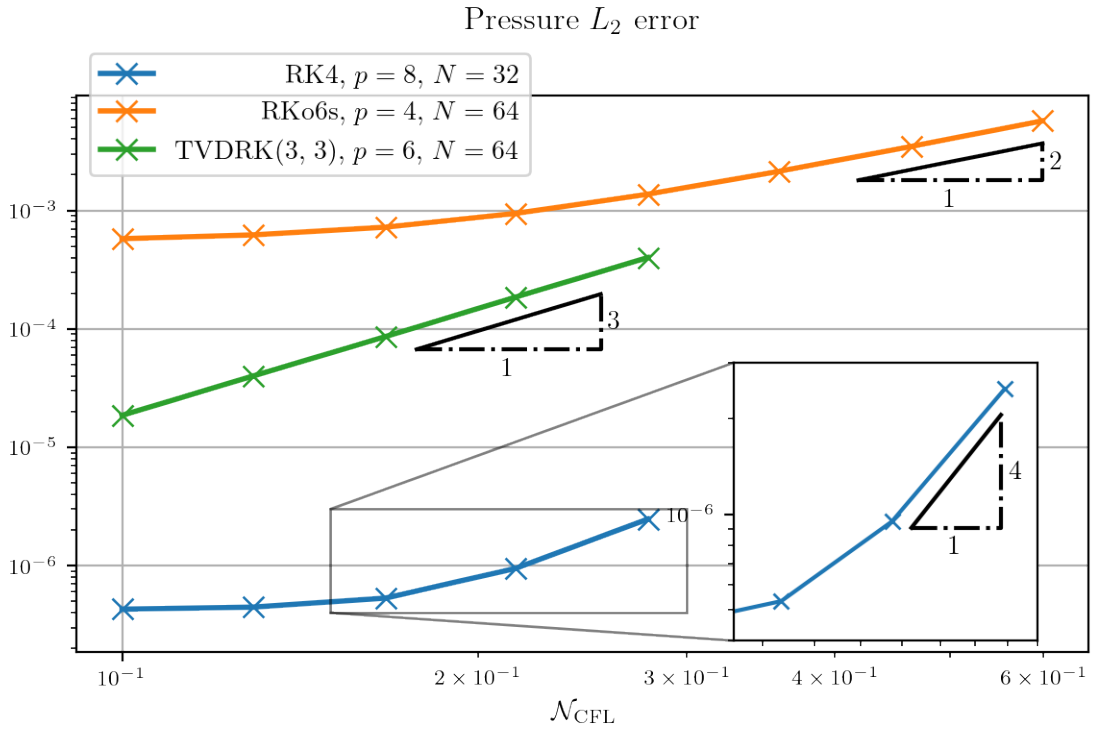


Figure 5.6: Global truncation error for the RK4, RKo6s and TVDRK(3, 3) methods.

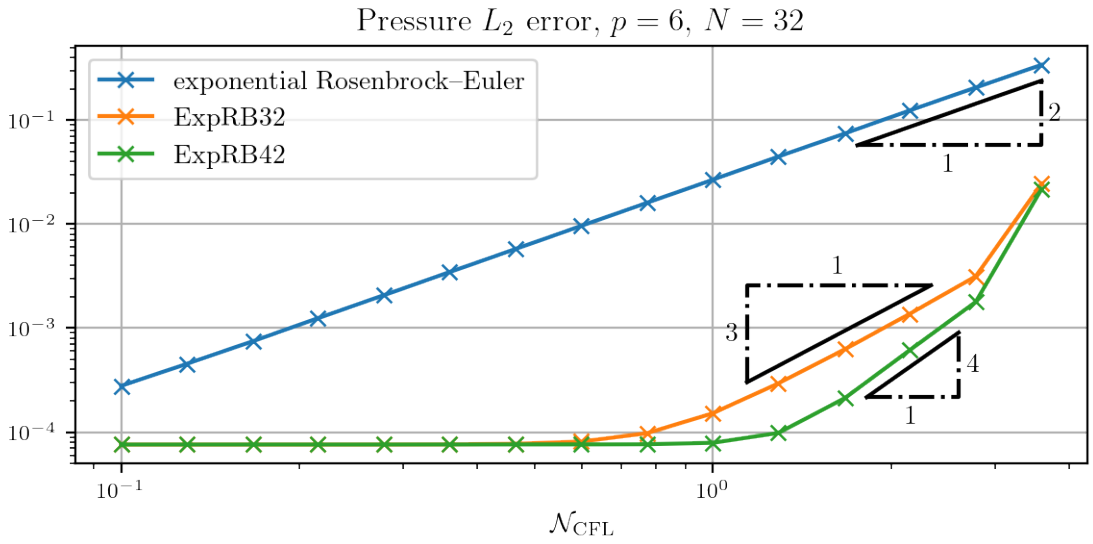


Figure 5.7: Global truncation error for the exponential Rosenbrock–Euler, ExpRB32 and ExpRB42 methods.

Rosenbrock methods can be reformulated as in equation (4.19) using the defects $D_{n,i}$. The first stage of an exponential Rosenbrock method is then an exponential Rosenbrock–Euler method, for which the contribution of the "full" right-hand side $f(y_n)$ is used. Here, this is done with the algorithms previously described, using a Krylov subspace of dimension 20. Then, because the defects are relatively smaller, the other stages compute their contribution with only 5 Krylov basis vectors. The two exponential Rosenbrock methods with two stages are not a lot more expensive than the Rosenbrock–Euler method. In comparison, a 2-stage explicit Runge–Kutta method will be twice as computationally expensive as the Euler method. This analysis was also performed with larger Krylov subspace methods. In this other configuration, the first stage uses 40 Krylov basis vectors for all three exponential methods, and the second stage uses 10 Krylov basis vectors for the ExpRB32 and ExpRB42 methods. However, using a larger subspace did not have any impact on the result of this analysis. For all (N, p) configurations, the error curve as a function of the time step was almost the same as with smaller Krylov subspace, such as they were almost indistinguishable when plotted in figures like figure 5.7.

We choose this first test case as it is widespread in the computational fluid dynamics community, as a tool to analyse the order of integration schemes. It often concerns spatial discretisation methods, but it can be considered for the time integration method as it gives access to the global truncation error, which is the error that interests the solver users. First, this analysis recovers the correct slope on the error curve as a function of the time step for the already existing explicit Runge–Kutta integration methods. Indeed, the error converges, as the time step decreases, to a minimum error value that comes from the spatial discretisation scheme. By refining the mesh and using higher-order spatial discretisation methods, we are able to reduce this minimal error and observe the expected curve for almost all methods. The same analysis but with the newly implemented exponential Rosenbrock methods showed the expected order. It also showed that the additional cost of using more stages with exponential methods is relatively small compared to the same additional cost for explicit Runge–Kutta methods. Finally, for this convected inviscid

isentropic vortex case, the exponential Rosenbrock methods were much more robust, as they work with higher CFL numbers. The analysis of their stability is the topic of the next section.

5.2.2 Robustness analysis: Taylor–Green vortex

The Taylor–Green vortex consists in following the evolution of a set of vortices in a periodic domain. It is a three-dimensional case, solution of the Navier–Stokes equations in a periodic box of length $2\pi L$ centered around the origin. The initial flow is given by:

$$\begin{cases} \underline{u} = \mathrm{Ma}\sqrt{\gamma r_{\mathrm{gas}} T_\infty} \begin{pmatrix} \sin\left(\frac{x}{L}\right)\cos\left(\frac{y}{L}\right)\cos\left(\frac{z}{L}\right) \\ \cos\left(\frac{x}{L}\right)\sin\left(\frac{y}{L}\right)\cos\left(\frac{z}{L}\right) \\ 0 \end{pmatrix} \\ T = T_\infty \\ P = P_\infty \left(1 + \frac{\gamma\,\mathrm{Ma}^2}{16}\left(\cos\left(\frac{2x}{L}\right) + \cos\left(\frac{2y}{L}\right)\right)\left(\cos\left(\frac{2z}{L}\right) + 2\right)\right) \end{cases} \tag{5.10}$$

over the domain $(x, y, z) \in [-\pi L, \pi L]^3$, as seen in figure 5.8. Even if $\underline{u}_z = 0$ at the initialisation, it becomes non-null afterwards and the problem is fully three-dimensional. The initial flow transitions to turbulence: the initial large scales decay into smaller ones, that end up dissipated. Figure 5.8 shows contours for which the second invariant of the velocity gradient, also called Q-criterion, is equal to 0.1. This was chosen as it roughly indicates the sizes of turbulent structures.

The numerical values are non dimensionalised: $L = 1$, $\mathrm{Ma} = 0.1$, $r_{\mathrm{gas}} = 71.4284$, $T_\infty = 1$ and $P_\infty = 71.4288$. The goal is to recover the Reynolds number $\mathrm{Re} = \rho_\infty U\infty L/\mu = 1600$ with the associated constant fluid viscosity is $\mu = 6.25 \times 10^{-4}$. The Prandtl number, the ratio of momentum diffusivity to thermal diffusivity, is equal to 0.71. Using the convective time $t_c = L/U_\infty$, it is known that the maximum dissipation happens around $8t_c$, and after $15t_c$ the flow is fully turbulent without any trace of the initial structures. The definition of the test case is part of the International Workshop on High-Order CFD Methods. A reference solution is also provided by the workshop, obtained using a spectral solver on a very refined mesh. The Spectral difference method uses the approximated Riemann solver of Roe.

The values of interest for this test case are the temporal evolution of the mean turbulent kinetic energy over the computational domain Ω:

$$E_k = \frac{1}{\rho_\infty |\Omega|}\int_\Omega \rho \frac{\|\underline{u}\|^2}{2}\mathrm{d}\Omega, \tag{5.11}$$

the mean enstrophy, defined by

$$\mathcal{E} = \frac{1}{\rho_\infty |\Omega|}\int_\Omega \rho \frac{\|\nabla \times \underline{u}\|^2}{2}\mathrm{d}\Omega. \tag{5.12}$$

Those values computed by the present simulation are compared with the reference values.

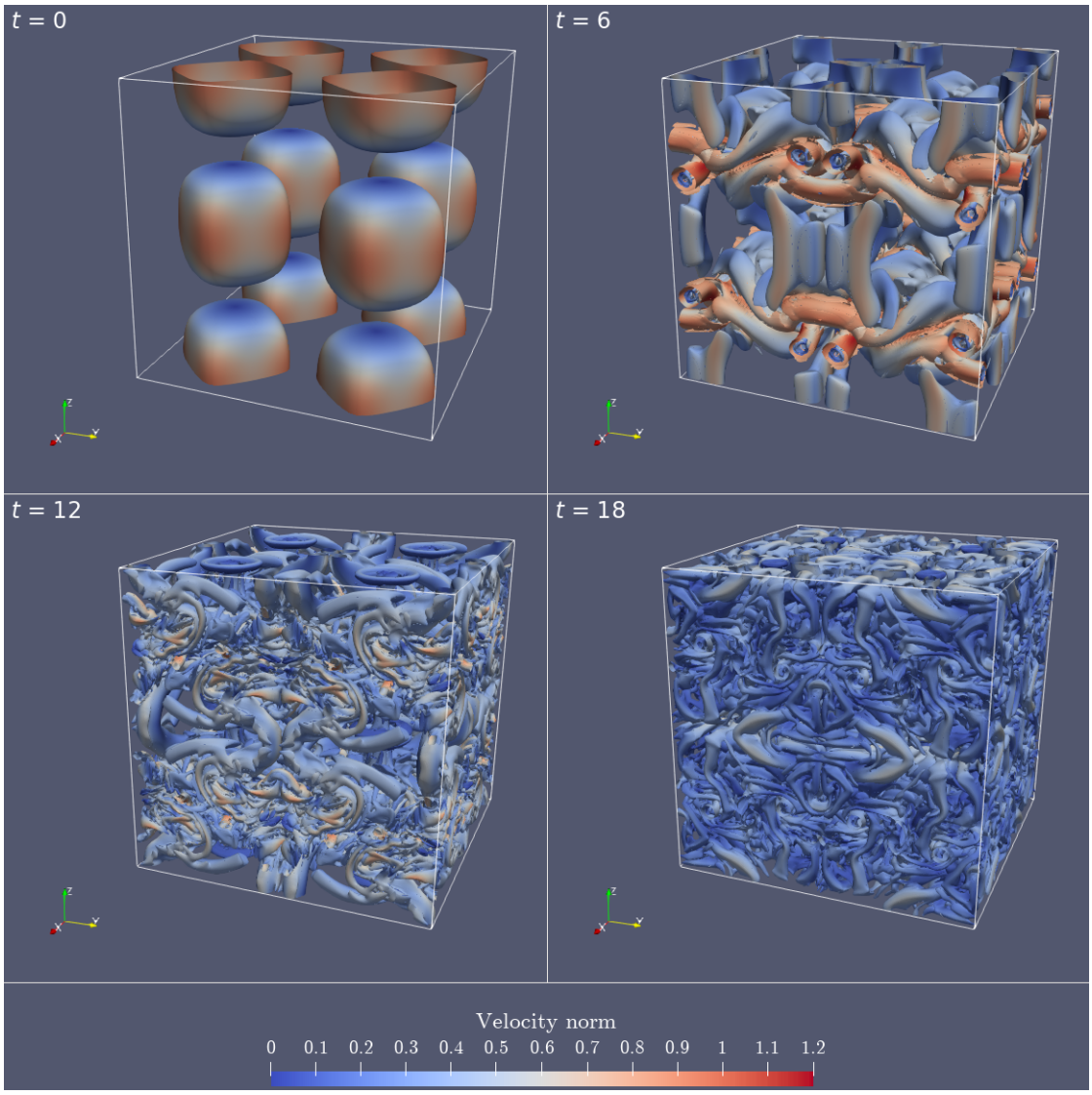


Figure 5.8: Isocontour (0.1) of the Q-criterion coloured by velocity norm of the Taylor–Green vortex at various times.

Method	$\mathcal{N}_{\text{CFL}}$	$N_{\text{iterations}}$	Time / iteration (s)	Total time (hh:mm:ss)
RKo6s	0.28	50000	1.588	22:03:20
TVDRK(3, 3)	0.14	100000	0.833	23:08:20
SSPRK(5, 4)	0.28	50000	1.497	20:47:30
ExpEuler(10)	1.4	10000	3.803	10:33:50
ExpEuler(20)	2.8	5000	8.144	11:18:40
ExpEuler(40)	5.6	2500	19.491	13:32:08
ExpEuler(80)	11.2	1250	52.792	18:19:50
ExpRB32	2.8	5000	10.391	14:25:55
ExpRB42	2.8	5000	19.787	27:28:55

Table 5.1: Statistics for various time integration methods. The time corresponds to elapsed real time, or wall-clock time. A computation corresponds to the simulation of the Taylor–Green vortex from equation (5.10) on time interval $[0, 20t_c]$.

The mesh used here is the same for all computations: a regular Cartesian mesh made of 80^3 cells. This choice is not natural from the workshop but previous experiments have led to the conclusion that such a mesh gives the best solution at the lowest CPU cost, for the Spectral Difference method with the order $p = 4$. Each computation is run on 308 CPU cores. The goal is to see if the result from the simulation matches reference data up to $20t_c$, and how much CPU time the computation took. The RKo6s, TVDRK(3, 3) and SSPRK(5, 4) methods are the already existing reference methods, and the exponential Rosenbrock–Euler, ExpRB32 and ExpRB42 methods are the newly implemented methods. For the exponential Rosenbrock methods, Krylov subspaces of dimension 20 are used for the first stage, and dimension 5 for the additional stages for the ExpRB32 and ExpRB42 methods. The time step is constant in a simulation, and some effort was done to use the highest time step compatible with each method. Above this upper bound on the time step, either the computation fails or the quality of the solution is not satisfying anymore. However, this higher time step can be increased for exponential methods by increasing the dimension of the Krylov subspaces used to compute φ-functions. By doing so, the computational cost of a single iteration is higher, but it reduces the error in the φ-functions evaluations. Symmetrically, it was also tried to reduce the dimension of the Krylov subspaces. It means that the method can not work with the same time step as before as it is now too high, but this reduces the cost of the iteration. This is to see if making more iterations that are cheaper, or in the opposite less that are more expensive can be a good idea. From now on, ExpEuler(d) method refers to the exponential Rosenbrock–Euler method that uses a Krylov subspace of dimension d.

The result of all simulations is shown in figure 5.9. As a simulation corresponds to the computation with the largest time step for which the results are satisfying, this figure does not give much information. At first sight, all methods produce results that are close to reference data. When looking more closely, there are some differences with the reference, but all JAGUAR computations are almost indistinguishable from one another. As a consequence, one can argue that the error from the spatial discretisation method is the origin of the difference between the reference and JAGUAR computations. It is then reasonable to assume that all curves correspond to valid computations. They all give the same result, which is taken as the reference solution for the considered mesh and the set of parameters of the study. Finally, as those methods give similar physical results, the correct way to compare them is to analse their statistics.

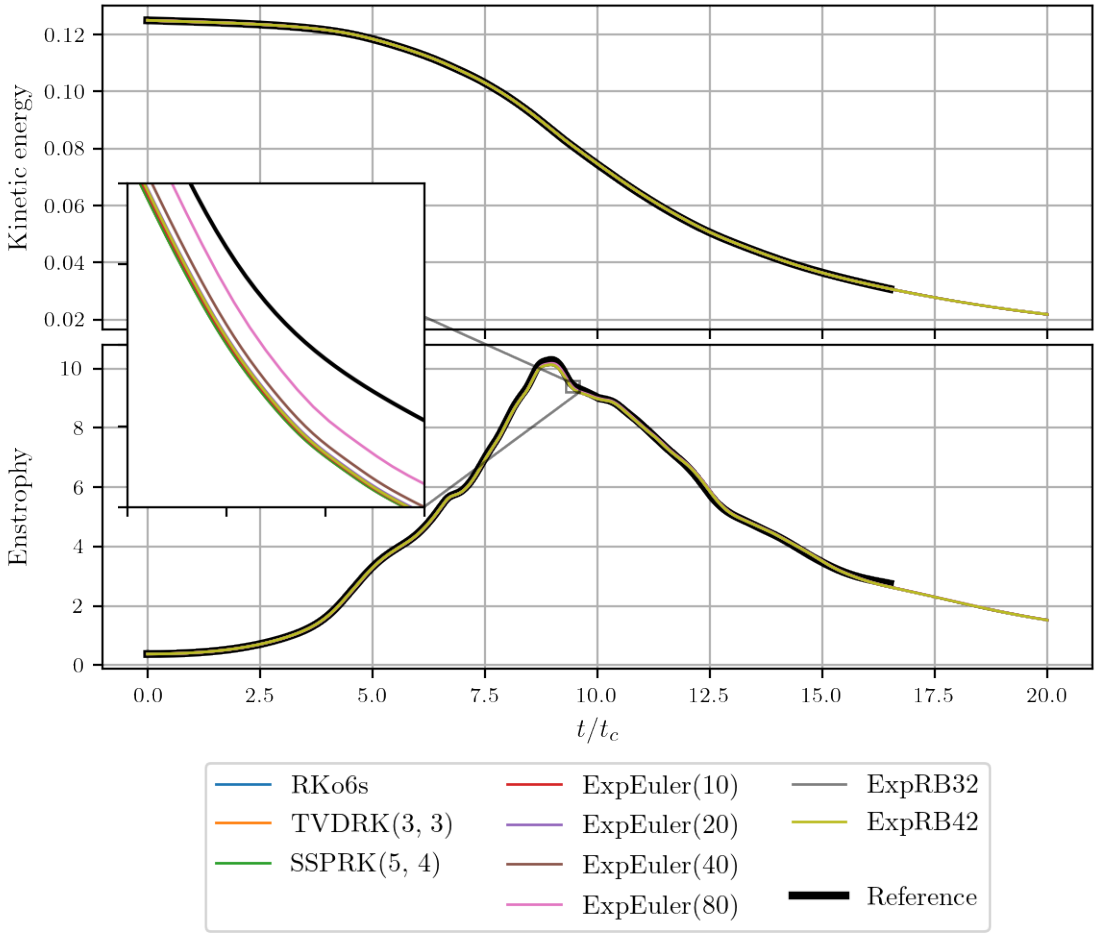


Figure 5.9: Kinetic energy and enstrophy for various time integration methods compared to reference data.

Statistics for each time integration method are provided in table 5.1. As the cost of the algorithms used in the time integration methods does not change between iterations, it makes sense to compute the average elapsed real time per iteration. Among the explicit methods, we see that the RKo6s and SSPRK(5, 4) methods run at a higher CFL number than the TVDRK(3, 3) method. However, they have more stages per iteration so each iteration takes longer, and the total elapsed real time is rather similar between all three methods. We note that the cost of one step of an explicit Runge–Kutta method is not proportional to the number of stages. Indeed, the implementation of the RKo6s method uses the fact that it is a low storage method to reduce the computational cost in terms of both time and memory. Nevertheless, the cost of an iteration of those explicit methods is much less than exponential methods. As expected, the cost of the exponential Rosenbrock–Euler method is similar to d^2 where d is the dimension of the Krylov subspace used for the Arnoldi iteration. As explained, the ExpRB32 method does not cost a lot more than the exponential Rosenbrock–Euler method that uses Krylov subspace with the same dimension. This is because the second stage uses only a small Krylov subspace, and this is what is seen from the results in table 5.1. We can draw two important conclusions from this analysis. First, we see that the exponential Rosenbrock methods are faster than the explicit Runge–Kutta methods available in JAGUAR. For some of them, they are almost twice as fast. Secondly, we see that the exponential Rosenbrock–Euler method can accurately simulate the Taylor–Green vortex at high CFL numbers. Accurate explicit methods would not work with such high CFL numbers, and even if implicit methods could they would not be accurate enough to capture and preserve the small scales of the flow. To reach high CFL numbers, we have to use larger Krylov subspaces, which makes the method a lot slower. The method used here should not be used for actual applications: the goal was just to show feasibility. It seems that with exponential methods, the robustness can be increased by paying the price in terms of the Krylov subspace dimension. We can think of ways to compute more accurately the φ-functions less expensively: by using restarted methods, for instance, we can control the maximal dimension of the Krylov subspace. However, this was not tried during this thesis.

This unsteady test case shows that exponential methods are interesting for our applications. We found a method that halves the elapsed real time, which is a great improvement for an unsteady simulation. We also showed that we could accurately simulate this flow while using high CFL numbers. It means that we have added more robust methods than the already existing ones. They could prove useful in applications where we are limited by the lack of stability of the explicit Runge–Kutta methods.

5.2.3 Industrial application: LS89

We decided to try exponential methods on a final test case. Contrary to the previous two, we stepped out of the academic context with an industrial application. It is a simulation of the flow inside the LS89 turbine blade cascade that was studied experimentally by the von Karman Institute [136]. We choose this test case as it has already been done with several solvers, including JAGUAR recently [137]. The computation uses a two-dimensional unstructured mesh, made of 2374 cells, that was extruded over 20 cells for a length of $0.15c$ with c the chord of the turbine blade. The final three-dimensional mesh made of 47 480 cells is shown in figure 5.10. The mesh is periodic in the extruded direction and the vertical direction. The boundary condition at the wall is isothermal, with $T_{\text{wall}} = 297.75\text{K}$. At inlet, the stagnation conditions are $P_0 = 184\,900\text{Pa}$ and $T_0 = 409.2\text{K}$. The outlet pressure is set to $116\,487\text{Pa}$. The upper and lower boundaries, as well as the front and rear planes, correspond to periodic boundary conditions. The Spectral difference method uses the HLLC approximate Riemann solver. This is an unsteady LES simulation: the

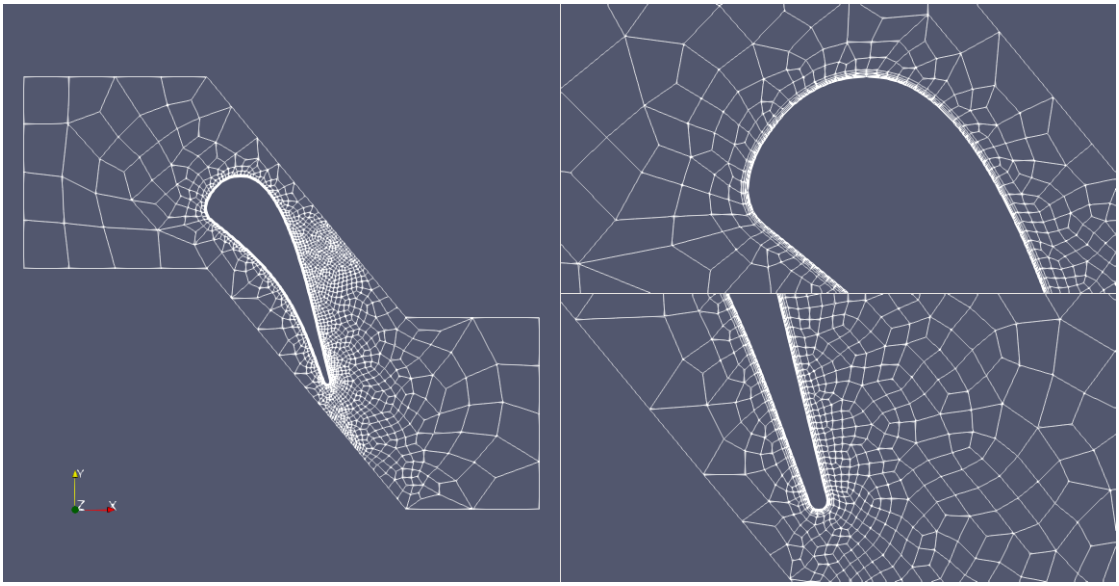


Figure 5.10: Mesh for the LS89 turbine blade. Close up on the leading and trailing edges.

interest lies in the time average values after a long simulation time.

From the previous results, we decide to use the exponential Rosenbrock–Euler method for the time integration. Indeed, it seems to be the quickest method, and we wish here to reduce the computational time. The other exponential methods would improve the accuracy of the physical results, so we keep only the Rosenbrock–Euler method to simplify the analysis.

The reason why we decided to use exponential methods for this numerical application is that in previous works, the explicit time step was limited by the viscous effect that prevails in the small cells of the boundary layer. Indeed, this test case was used to compare several solvers, but as JAGUAR only had explicit methods it took longer to find a solution [137]. Because exponential methods handle well the linear part of equations, and viscous effects are mostly linear, we thought that exponential methods would allow JAGUAR to break free from the constraint on the time step, and improve the overall speed of the solver. In figure 5.11, the Mach number is shown at the end of the computation. It is in line with what is expected, and this proves the capacity of exponential methods to work as time integration schemes for such applications.

The physical value investigated in [137] is the heat transfer coefficient. The reader is invited to look at this reference for the interpretation of the physical results. Here, we assume that JAGUAR produces a satisfying solution with its classical explicit TVD(3, 3) method, and we compare it with the exponential Rosenbrock–Euler method. The heat transfer coefficient is shown in figure 5.12, and we see that the results are indistinguishable between the two methods. In this figure, the origin of the curvilinear abscissa corresponds to the leftmost point on the blade, and positive (respectively negative) abscissa corresponds to the upper (respectively lower) part of the blade. This result validates the capability of exponential methods to be used as time integration methods in computational fluid dynamics.

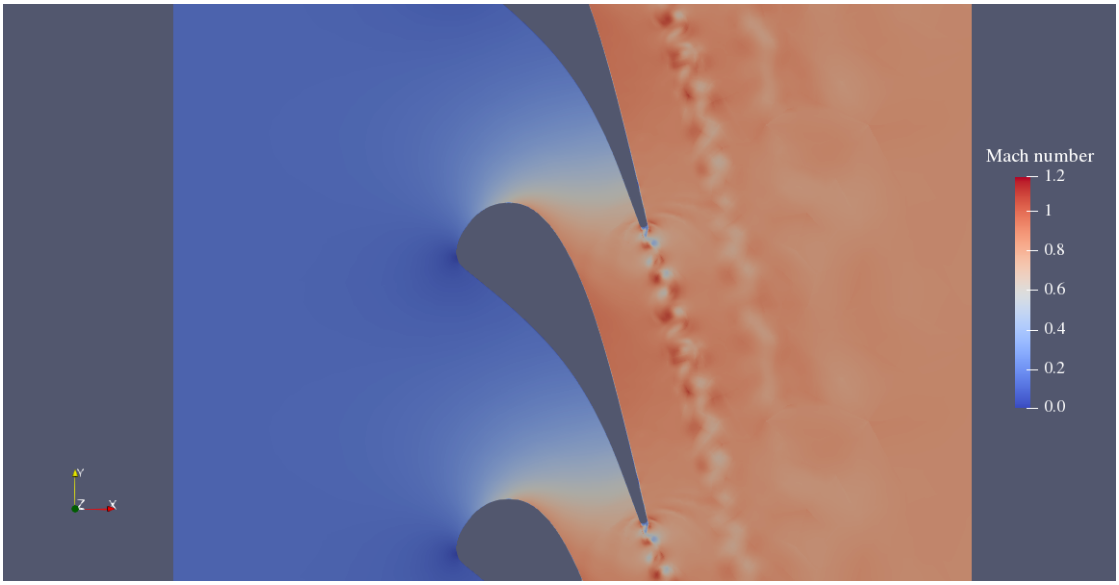


Figure 5.11: Mach number in the LS89 turbine blade cascade.

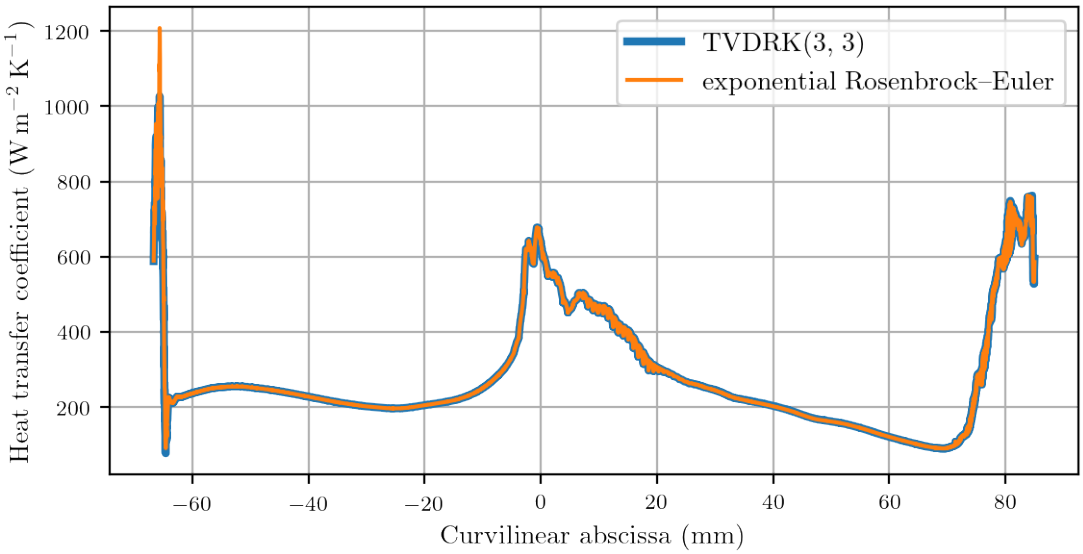


Figure 5.12: Heat transfer coefficient from the TVDRK(3, 3) and exponetial Rosenbrock–Euler methods. Negative abscissa corresponds to the lower part of the blade and positive abscissa to the upper part.

Our logic was that if the time step for this application is limited by viscous effects, and viscous effects are mainly linear, exponential methods would accept higher time steps, and in the end, the computation would be faster. However, our initial hypothesis was wrong: the time step is limited by advection and not diffusion. It was originally true at the time of some previous investigations [137], but as the diffusive scheme was recently greatly improved [138], it is not true anymore. This is why in the end, there are no significant improvements in terms of computational speed when using exponential integration methods. Even if we did not find the improvements we were originally looking for, this numerical experiment did show that exponential time integration methods can be used in actual computational fluid applications.

Conclusion

The work made towards exponential methods during this thesis shows their interest in the time integration of computational fluid dynamics problems. It is now clear that they are precise methods, and using them makes a lot of sense when the spatial discretisation method is accurate as well, with high-order methods for instance. Then, some numerical experiments with the JAGUAR solver showed their quality. For one application, they were particularly quicker and more stable than classical explicit integration methods. For the last application, they did not improve the solver performances as expected, but it could be argued that we expected too much because of some wrong hypothesis on our part. Overall, this work showed that exponential methods are worth investigating.

Conclusion and perspectives

Jacobian-Free Newton–Krylov for implicit time integration

This thesis was interested in solving efficiently steady problems for multiphysics applications. It means improving the convergence, stability and speed of already existing methods, particularly for the software system CEDRE. A preliminary analysis of existing methods within CEDRE and the literature focussed our effort on improving the nonlinear and linear resolutions required by implicit time integration methods. In particular, the Jacobian-Free Newton–Krylov method caught our interest. This is first because it reuses many algorithms that were already available in CEDRE. The other reason is that it is a better alternative to what was currently done, which is using a poor approximation of the Jacobian matrix. Indeed, using the JFNK method amounts to using a full Jacobian matrix that accounts for all numerical models. We hoped that using a more accurate matrix for the linear problem would improve the quality of the nonlinear solution, which would in turn improve the overall quality of the implicit time integration method. With a careful implementation, it was added to the software system to ensure compatibility with other solvers and existing algorithms.

To check the quality of the Jacobian-Free Newton–Krylov method, it was tested on several applications. Those applications were chosen to represent typical CEDRE computations, to evaluate if the method is interesting for the solver. The first application was the simulation of the turbulent flow around a wing profile in two dimensions. When compared with the traditional method, the new one that is matrix-free gives similar results when looking at flow data. The improvement is solely on the convergence, as the JFNK method converges. The second application is the same wing profile, but this time using a much finer mesh to finely represent the boundary layer. This time, the traditional method lacks the ability to converge satisfyingly when the JFNK method, however, does. It shows that the JFNK method improves the convergence of the solver on such applications.

As CEDRE aims to solve problems in the field of multiphysics, it is not enough to test the new method on aerodynamics test cases. This is why the next chosen application was the simulation of the hypersonic reactive flow around a solid sphere. It is a simplified typical reentry application. Similar results are found when looking at the non-reactive equivalent of this test case: the JFNK method convergence is better. However, when using a fine mesh with the reactive model, the conclusion is no longer in favour of the matrix-free method as the older method converges better. Still, the difference in the convergence between the two methods is small compared to the same difference in the non-reactive case. A possible conclusion is that in the best-case scenario, when the traditional method converges well, the JFNK does not improve the quality of the solution. When the traditional method struggles to find a steady solution, however, the JFNK method

becomes interesting.

Finally, the Jacobian-Free Newton–Krylov method was tested on a newly implemented fluid model. Indeed, as CEDRE is under current development, new fluid models are sometimes added to represent flow features with higher fidelity. For instance, the new model used here accounts more precisely for thermodynamic disequilibrium. A newly implemented model often does not give access to their Jacobian matrices, as it would require much more work from developers and is not a priority. It means that users must restrict themselves to using explicit time integration methods. As the JFNK method does not require Jacobian matrices, it is a good candidate to be used as a time integration method with such models. It was then compared with the explicit Midpoint method. In the end, the JFNK method converges better than the explicit method and is also quicker in terms of CPU time. The result is even more in favour of the implicit method when using a simplified version of the reactive model that disregards ionisation.

One key argument of the Jacobian-Free Newton–Krylov method is the possibility to handle the Jacobian matrix approximation in the same order of accuracy as the spatial discretization. We showed that accuracy can lead to improved convergence. Rough approximations of the Jacobian matrices should be avoided in the future, or considered only for tests or as preconditioners. If the choice of CEDRE developers is finally to revert to a Jacobian matrix-based formulation, we can assume that the computation of the Jacobian matrix will be a key ingredient. Doing this by hand is time-consuming and can lead to implementation issues. An alternative could be considering code differentiation. But in the context of CEDRE, it means revisiting the full structure of the whole set of solvers.

The downfall of the Jacobian-Free Newton–Krylov method is its slowness. Indeed, it requires multiple right-hand side evaluations, and such evaluations are quite expensive in our solver. It means that for a user, there is no interest to use the JFNK methods on problems for which the traditional method gives satisfying results and convergence. It is still useful to achieve precise convergence when the traditional method fails to do so. However, improving the efficiency of right-hand side evaluations is an active research topic within the team in charge of the solver. Such improvement will benefit the JFNK method a lot more than it will benefit the traditional method, so the interest in the JFNK method should increase.

In light of this work, it seems natural to want to try the Jacobian-Free Newton–Krylov method with other fluid models that also do not have yet an available Jacobian matrix. Such models already exist in CEDRE, and they are restricted to the explicit time integration method, as was the model detailed in this thesis before it used the matrix-free method. It is reasonable to assume that the JFNK method will yield similar results, and will unlock implicit methods for those models.

Going further, one might want to try the same method but with another solver from the platform that is CEDRE. Indeed, this would amount to very few developments which is why this idea is interesting. Finally, if multiple solvers use the matrix-free method, it means that CEDRE would be able to integrate implicitly those solvers in a coupled fashion. Because of the structural choices made in CEDRE, this is currently impossible unless refactoring a significant amount of code.

As the Jacobian-Free Newton–Krylov method does not use an actual matrix, it is limited in terms of available preconditioners. More generally, preconditioners available in CEDRE are quite simple and the linear solver would benefit from better preconditioners from the literature that already works well in other computational fluid dynamics solvers. Physics-based preconditioner could improve the linear solver performances which would improve robustness, convergence and speed of the overall integration method [6, 53]. In particular, for the Multi-TEmperature model one that was used in this thesis, a physics-based preconditioner would help handle the stiff part corresponding to electrons and ionisation.

Exponential integration methods for unsteady time integration with large time steps

Another topic of interest in this thesis was exponential time integration methods. Indeed, such methods reuse many parts and algorithms that are classicly used for implicit time integration. Furthermore, methods used for implicit time integration are often adapted to solve unsteady problems with large time steps. A preliminary analysis showed that the accuracy of exponential methods was similar to the one of explicit methods while being able to use relatively large time steps as implicit methods. As exponential methods are quite precise, they require spatial discretisation methods that are at least as accurate. This is why exponential methods were then studied with the JAGUAR solver that uses a Spectral Difference method.

A first numerical experiment showed that the newly added exponential methods had the expected order of accuracy in JAGUAR, and prove more stable than explicit methods already available in the solver. Then, the same methods were used in another test case. It showed that exponential methods can reduce significantly the CPU time required to fulfil the computation. Furthermore, some methods were able to use time steps 40 times higher than explicit methods and were still quicker to fulfil the computation. This highlight the quality of exponential methods for unsteady time integration when using large time steps. In a final test case, it was shown that exponential methods still work fine on an actual LES computation, with fine cells in a wall boundary condition. Our first hope was that exponential methods would prove faster, but in the end, their performances were similar to the ones of explicit methods.

This preliminary work did demonstrate the feasibility of exponential methods in JAGUAR and their suitability for unsteady time integration. Rewriting the exponential computation routines within JAGUAR instead of using the external SLEPc library would improve their quickness even more. Indeed, using a well-written library is often a good idea to benefit from its quality as its developers are experts in their fields, but values are constantly copied back and forth between JAGUAR and SLEPc data types with the current implementation. The solution is either to modify the solver so that it uses SLEPc data types globally, or to rewrite SLEPc methods suited for JAGUAR data types. The latter would also allow for specific optimisations. For instance, we mentioned earlier that exponential methods often compute φ-functions of the same matrix. One could reuse spectral information between each of those computations to speed them up and get more accurate results.

As JAGUAR had only explicit integration methods before exponential methods were added, no attention was paid to preconditioning. Now that there are algorithms that use Krylov subspace methods, preconditioners could prove beneficial for the time integration [139].

The last JAGUAR application that was presented here was originally chosen as it was limited in the time step size by the diffusive part of the equations, which is mostly linear. Exponential methods would have then improved the time integration as they deal efficiently with the linear parts. However, this initial statement was not true, and therefore the gain from using exponential methods did not appear. It would be interesting to see what would happen on a test case for which this statement holds.

Going outside the scope of JAGUAR, exponential methods could prove to be great candidates as time integration methods on other solvers. For instance, solvers that simulate plasmas and arc lightning are significantly limited in the time step size by viscous effects. As such effects are mostly linear, exponential integration methods would allow breaking free from this limitation. The same idea applies to the field of thermic for which large physical times need to be simulated. Exponential methods should be compatible with larger time steps, which would reduce the overall computational time.

Exponential methods are becoming more and more active in computational fluid dynamics. For the same reason we decided to use them along a Spectral Difference method, they are often used with high-order spatial discretisation methods. For instance, the Discontinuous-Galerkin method from [140] is used with an exponential Rosenbrock method described in [141, 142]. In [143], exponential integrators are used to solve multiphysics problems. Many other recent references from the literature use exponential methods as time integrators for similar computational fluid dynamics problems. This shows that exponential methods are worth investigating.

Appendix A

Conservative preconditioning

Let us consider the ordinary differential equation:

$$M\frac{\mathrm{d}Q}{\mathrm{d}t} = \mathrm{F}\left(Q\left(t\right)\right). \tag{A.1}$$

This equation comes from the Finite Volume discretisation, where Q is the vector of the conservative variables and F a conservative right-hand side. To keep this simple, we consider that there is a single conservative variable, so that the dimension of Q is equal to the number of cells. This is only to simplify the notations, but everything can be adapted to use more conservative variables. The mass matrix M is a diagonal matrix where the ith diagonal coefficient corresponds to the volume of the ith cell. We note by S the vector of the same dimension as Q made of ones and transposed. This way, multiplying on the left a vector by S amount to sum a vector components. The conservation property of the equation means that $S\,\mathrm{F}\left(Q\right) = 0$ or equivalently that the sum of the conservative variable over the domain SMQ is constant.

When solving this equation with the explicit Euler method, we have at the nth step that:

$$M\delta Q_n = \Delta t\,\mathrm{F}\left(Q_n\right) \tag{A.2}$$

with $\delta Q_n = Q_{n+1} - Q_n$. Is is clear that $SMQ_{n+1} = SMQ_n$, and so the explicit Euler method preserves the conservation property.

When using the implicit Euler method with a single linearisation, we need the Jacobian matrix J_n of the function F evaluated in Q_n. Since $J_n = \mathrm{F}'\left(Q_n\right)$ and $S\,\mathrm{F}\left(Q_n\right) = 0$, we have $SJ_n = 0$ also. The method gives the increment δQ_n as the solution of:

$$\left(M - \Delta t J_n\right)\delta Q_n = \Delta t\,\mathrm{F}\left(Q_n\right). \tag{A.3}$$

Multiplying this relation on the left by S and using that $SJ_n = S\,\mathrm{F}\left(Q_n\right) = 0$, we have $SM\delta Q_n = 0$, which means this method also preserves the conservation property.

However, we do not usually use the exact solution of the linear problem but the solution of a subspace Krylov method. Let us consider that we use k steps of a Krylov subspace method with a zero initial guess. Then, the increment belongs to the corresponding Krylov subspace:

$$\delta Q_n \in \mathrm{Vect}\left(\left(M - \Delta t J_n\right)^i \mathrm{F}\left(Q_n\right)\right)_{0\leq i<k}. \tag{A.4}$$

Now, there are no reasons for $SM\delta Q_n$ to be equal to zero. For example, the solution computed with the smallest Krylov subspace dimension ($k = 1$) is parallel to $\mathrm{F}(Q_n)$, and then $SM\delta Q_n \propto SM\,\mathrm{F}(Q_n)$ and therefore is not null.

With preconditioning however, we can recover the conservation property. If we use the invert of the mass matrix as a preconditioner, either a left or a right one, we have that:

$$\delta Q_n \in \mathrm{Vect}\,(v_i)_{0\leq i<k} \quad \text{with} \quad v_i = \left(\mathrm{Id} - \Delta t M^{-1} J_n\right)^i M^{-1}\,\mathrm{F}(Q_n)\,. \tag{A.5}$$

We can verify by recurrence that $SMv_i = 0$:

- $SMv_0 = SMM^{-1}\,\mathrm{F}(Q_n) = S\,\mathrm{F}(Q_n) = 0$
- if $SMv_i = 0$, then $SMv_{i+1} = SM\left(\mathrm{Id} - \Delta t M^{-1} J_n\right) v_i = SMv_i - \Delta t S J_n v_i = 0$ as $SJ_n = 0$.

Then, we have that $SM\delta Q_n = 0$ which means the preconditionned method preserves the conservation property.

Bibliography

[1] A. Refloch, B. Courbet, A. Murrone, P. Villedieu, C. Laurent, P. Gilbank, J. Troyes, L. Tessé, G. Chaineray, J. Dargaud, E. Quémerais, and F. Vuillot, "CEDRE software," *AerospaceLab journal*, pp. 1–10, 2011.

[2] G. Selva, *Méthodes itératives pour l'intégration implicite des équations de l'aérothermochimie sur des maillages non-structurés.* PhD thesis, Ecole Centrale Paris, 1998. Thèse de doctorat dirigée par Giovangigli, Vincent Mécanique des fluides Châtenay-Malabry, Ecole centrale de Paris 1998.

[3] Z. Y. Wong, F. Kwok, R. N. Horne, and H. A. Tchelepi, "Sequential-implicit Newton method for multiphysics simulation," *Journal of Computational Physics*, 2019.

[4] D. Coulette, E. Franck, P. Helluy, A. Ratnani, and E. Sonnendrücker, "Implicit time schemes for compressible fluid models based on relaxation methods," *Computers & Fluids*, vol. 188, pp. 70–85, 2019.

[5] D. Z. Huang, W. Pazner, P.-O. Persson, and M. J. Zahr, "High-order partitioned spectral deferred correction solvers for multiphysics problems," *Journal of Computational Physics*, vol. 412, p. 109441, 2020.

[6] H. Park, R. R. Nourgaliev, R. C. Martineau, and D. A. Knoll, "On physics-based preconditioning of the Navier-Stokes equations," *Journal of Computational Physics*, vol. 228, no. 24, pp. 9131–9146, 2009.

[7] C. Content, P.-Y. Outtier, and P. Cinnella, "Coupled/uncoupled solutions of RANS equations using a Jacobian-free Newton–Krylov method," in *21st AIAA Computational Fluid Dynamics Conference*, 2013.

[8] R. Turpault, *Modélisation, approximation numérique et applications du transfert radiatif en déséquilibre spectral couplé avec l'hydrodynamique.* Theses, Université Sciences et Technologies - Bordeaux I, Dec. 2003.

[9] P. Seize, L. Matuszewski, and G. Puigt, "Towards multi-physics simulations using Jacobian-free Newton–Krylov method," in *AIAA AVIATION 2022 Forum*, 2022.

[10] R. Eymard, T. Gallouët, and R. Herbin, "Finite volume methods," in *Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3)*, vol. 7 of *Handbook of Numerical Analysis*, pp. 713–1018, Elsevier, 2000.

[11] N. Leterrier, *Discrétisation spatiale en maillage non-structuré de type général.* PhD thesis, Université Pierre et Marie Curie - Paris VI, 2003. Thèse de doctorat dirigée par Giovangigli, Vincent Calcul scientifique Paris 6 2003.

[12] P. L. Roe, "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, vol. 43, no. 2, pp. 357–372, 1981.

[13] E. F. Toro, *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Berlin Heidelberg, 2009.

[14] B. Van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method," *Journal of Computational Physics*, vol. 32, no. 1, pp. 101–136, 1979.

[15] F. Haider, J. Croisille, and B. Courbet, "Stability analysis of the cell centered finite-volume MUSCL method on unstructured grids," *Numer. Math.*, vol. 113, pp. 555–600, 2009.

[16] F. Haider, J. Croisille, and B. Courbet, "Efficient implementation of high order reconstruction in finite volume methods," in *Finite Volumes for Complex Applications IV: problems and perspectives* (J. Fort, J. Furst, J. Halama, R. Herbin, and F. Hubert, eds.), vol. 4 of *Springer proceedings in mathematics*, (Prague, Czech Republic), pp. 553–562, Springer, June 2011.

[17] F. Haider, B. Courbet, and J. Croisille, "A high-order interpolation for the finite volume method: the coupled least squares reconstruction," *Computers & Fluids*, vol. 176, pp. 20–38, 2018.

[18] G. Pont, P. Brenner, P. Cinella, B. Maugars, and J.-C. Robinet, "Multiple-correction hybrid $k-$exact schemes for high-order compressible RANS-LES simulations on general unstructured grids," *Journal of Computational Physics*, vol. 350, pp. 45–83, 2017.

[19] C. Le Touze, A. Murrone, and H. Guillard, "Multislope MUSCL method for general unstructured meshes," *Journal of Computational Physics*, vol. 284, pp. 389–418, 2015.

[20] V. Venkatakrishnan, "On the accuracy of limiters and convergence to steady state solutions," tech. rep., AlAA 93-0880, 1993.

[21] M. Berger and M. Aftosmis, "Analysis of slope limiters on irregular grids," *43rd AIAA Aerospace Sciences Meeting and Exhibit - Meeting Papers*, Feb. 2005.

[22] C. Kelley and D. Keyes, "Convergence analysis of Pseudo-Transient Continuation," *SIAM Journal on Numerical Analysis*, vol. 35, Aug. 1996.

[23] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *IBM Journal of Research and Development*, vol. 11, no. 2, pp. 215–234, 1967.

[24] A. Iserles, *A first course in the numerical analysis of differential equations*. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2 ed., 2008.

[25] E. Hairer and G. Wanner, *Solving ordinary differential equations II. Stiff and differential-algebraic problems*, vol. 14. Springer Berlin Heidelberg, Jan. 1996.

[26] G. G. Dahlquist, "A special stability problem for linear multistep methods," *BIT Numerical Mathematics*, vol. 3, pp. 27–43, Mar. 1963.

[27] E. Hairer, S. Norsett, and G. Wanner, *Solving ordinary differential equations I. Nonstiff problems*, vol. 8. Springer Berlin Heidelberg, Jan. 1993.

[28] R. Alexander, "Diagonally implicit Runge–Kutta methods for stiff O.D.E.'s," *SIAM Journal on Numerical Analysis*, vol. 14, no. 6, pp. 1006–1021, 1977.

[29] H. H. Rosenbrock, "Some general implicit processes for the numerical solution of differential equations," *The Computer Journal*, vol. 5, pp. 329–330, Jan. 1963.

[30] G. Wanner, "On the integration of stiff differential equations," in *Numerical Analysis: Proceedings of the Colloquium on Numerical Analysis Lausanne, October 11–13, 1976* (J. Descloux and J. Marti, eds.), (Basel), pp. 209–226, Birkhäuser Basel, 1977.

[31] A. Gaul, *Recycling Krylov subspace methods for sequences of linear systems – analysis and applications*. PhD thesis, TU Berlin, July 2014.

[32] Y. Saad, *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Apr. 2003.

[33] F. Pacull, S. Aubert, and M. Buisson, "A study of ILU factorization for Schwarz preconditioners with application to Computational Fluid Dynamics," in *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, vol. 95, Apr. 2011.

[34] M. A. Olshanskii and E. E. Tyrtyshnikov, *Iterative methods for linear systems*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014.

[35] V. Simoncini and D. B. Szyld, "Recent computational developments in Krylov subspace methods for linear systems," *Numerical Linear Algebra with Applications*, vol. 14, no. 1, pp. 1–59, 2007.

[36] L. N. Trefethen and D. Bau, *Numerical linear algebra*. SIAM: Society for Industrial and Applied Mathematics, June 1997.

[37] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[38] M. Franco, J.-S. Camier, J. Andrej, and W. Pazner, "High-order matrix-free incompressible flow solvers with GPU acceleration and low-order refined preconditioners," *Computers & Fluids*, vol. 203, p. 104541, 2020.

[39] O. G. Ernst and M. J. Gander, "Why it is difficult to solve Helmholtz problems with classical iterative methods," in *Numerical Analysis of Multiscale Problems* (I. G. Graham, T. Y. Hou, O. Lakkis, and R. Scheichl, eds.), ch. 10, pp. 325–363, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[40] S. Mercier, *Fast nonlinear solvers in solid mechanics*. PhD thesis, Université Toulouse 3 Paul Sabatier, 2015. Thèse de doctorat dirigée par Gratton, Serge et Vasseur, Xavier Mathématiques appliquées Toulouse 3 2015.

[41] H. A. van der Vorst, "Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[42] O. Coulaud, L. Giraud, P. Ramet, and X. Vasseur, "Deflation and augmentation techniques in Krylov subspace methods for the solution of linear systems," Tech. Rep. 8265, INRIA, Feb. 2013.

[43] X. Vasseur, *Contribution to the study of efficient iterative methods for the numerical solution of partial differential equations.* PhD thesis, Habilitation à Diriger des Recherches, Institut National Polytechnique de Toulouse, June 2016.

[44] P. Jolivet and P.-H. Tournier, "Block iterative methods and recycling for improved scalability of linear solvers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, IEEE Press, 2016.

[45] O. Nevanlinna, "How fast can iterative methods be," in *Recent Advances in Iterative Methods* (G. Golub, M. Luskin, and A. Greenbaum, eds.), pp. 135–147, Springer New York, 1994.

[46] A. Greenbaum, V. Pták, and Z. Strakos, "Any nonincreasing convergence curve is possible for GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 3, pp. 465–469, 1996.

[47] A. Greenbaum and Z. Strakos, "Matrices that generate the same Krylov residual spaces," in *Recent Advances in Iterative Methods* (G. Golub, M. Luskin, and A. Greenbaum, eds.), pp. 95–118, Springer New York, 1994.

[48] L. N. Trefethen, "Pseudospectra of matrices," *Numerical analysis*, vol. 91, pp. 234–266, 1991.

[49] L. N. Trefethen, "Spectra and pseudospectra," in *The Graduate Student's Guide to Numerical Analysis '98: Lecture Notes from the VIII EPSRC Summer School in Numerical Analysis* (M. Ainsworth, J. Levesley, and M. Marletta, eds.), pp. 217–250, Springer Berlin Heidelberg, 1999.

[50] J. Liesen and P. Tichý, "Convergence analysis of Krylov subspace methods," *GAMM-Mitteilungen*, vol. 27, no. 2, pp. 153–173, 2004.

[51] M. Huhtanen, "Aspects of nonnormality for iterative methods," *Linear Algebra and its Applications*, vol. 394, pp. 119–144, 2005.

[52] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, "Approximating the inverse of a matrix for use in iterative algorithms on vector processors," *Computing*, vol. 22, pp. 257–268, Sept. 1979.

[53] W. Liu, L. Zhang, Y. Zhong, Y. Wang, Y. Che, C. Xu, and X. Cheng, "CFD high-order accurate scheme Jacobian-Free Newton Krylov method," *Computers & Fluids*, vol. 110, pp. 43–47, 2015. ParCFD 2013.

[54] B. R. Ahrabi and D. J. Mavriplis, "An implicit block ILU smoother for preconditioning of Newton-Krylov solvers with application in high-order stabilized finite-element methods," *Computer Methods in Applied Mechanics and Engineering*, vol. 358, 2020.

[55] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific Computing*, vol. 14, no. 2, pp. 461–469, 1993.

[56] V. Simoncini and D. Szyld, "Flexible inner-outer Krylov subspace methods," *SIAM Journal on Numerical Analysis*, vol. 40, no. 6, pp. 2219–2239, 2002.

[57] X. Pinel, *A perturbed two-level preconditioner for the solution of three-dimensional heterogeneous Helmholtz problems with applications to geophysics*. PhD thesis, Institut National Polytechnique de Toulouse, 2010. Thèse de doctorat dirigée par Gratton, Serge Mathématiques, Informatiques et Télécommunication Toulouse, INPT 2010.

[58] V. Simoncini, "A new variant of restarted GMRES," *Numerical Linear Algebra with Applications*, vol. 6, no. 1, pp. 61–77, 1999.

[59] A. H. Baker, E. R. Jessup, and T. Manteuffel, "A technique for accelerating the convergence of restarted GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 4, pp. 962–984, 2005.

[60] A. Chapman and Y. Saad, "Deflated and augmented Krylov subspace techniques," *Numerical Linear Algebra with Applications*, vol. 4, no. 1, pp. 43–66, 1997.

[61] R. Morgan, "GMRES with deflation restarting," *Siam Journal on Scientific Computing*, vol. 24, Aug. 2002.

[62] L. G. Ramos, R. Kehl, and R. Nabben, "Projections, deflation and multigrid for nonsymmetric matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 41, no. 1, pp. 83–105, 2020.

[63] A. Griewank, *Evaluating derivatives*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[64] L. Hascoët and V. Pascual, "The Tapenade automatic differentiation tool: principles, model, and specification," Research Report RR-7957, INRIA, May 2012.

[65] M. Bilanceri, F. Beux, I. Elmahi, H. Guillard, and M. V. Salvetti, "Comparison of explicit and implicit time advancing in the simulation of a 2D sediment transport problem," in *Finite Volumes for Complex Applications VI Problems & Perspectives* (J. Fort, J. Fürst, J. Halama, R. Herbin, and F. Hubert, eds.), pp. 125–133, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[66] G. Kenway, C. Mader, P. He, and J. Martins, "Effective adjoint approaches for Computational Fluid Dynamics," *Progress in Aerospace Sciences*, June 2019.

[67] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your Jacobian? Graph coloring for computing derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, 2005.

[68] D. A. Knoll and D. E. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[69] H.-B. An, J. Wen, and T. Feng, "On finite difference approximation of a matrix-vector product in the Jacobian-free Newton-Krylov method," *Journal of Computational and Applied Mathematics*, vol. 236, no. 6, pp. 1399–1409, 2011.

[70] M. Martel, "Semantics of roundoff error propagation in finite precision calculations," *Higher-Order and Symbolic Computation*, vol. 19, pp. 7–30, Mar. 2006.

[71] S. Abhyankar, J. Brown, E. M. Constantinescu, D. Ghosh, B. F. Smith, and H. Zhang, "PETSc/TS: a modern scalable ODE/DAE solver library," *arXiv preprint arXiv:1806.01437*, 2018.

[72] M. Pernice and H. F. Walker, "NITSOL: a Newton iterative solver for nonlinear systems," *SIAM Journal on Scientific Computing*, vol. 19, no. 1, pp. 302–318, 1998.

[73] J. E. Dennis and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. Society for Industrial and Applied Mathematics, 1996.

[74] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang, "PETSc Web page." `https://petsc.org/`, 2022.

[75] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang, "PETSc/TAO users manual," Tech. Rep. ANL-21/39 - Revision 3.18, Argonne National Laboratory, 2022.

[76] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing* (E. Arge, A. M. Bruaset, and H. P. Langtangen, eds.), pp. 163–202, Birkhäuser Press, 1997.

[77] T. Hellstrom, L. Davidson, and A. Rizzi, "Reynolds stress transport modelling of transonic flow around the RAE2822 airfoil," in *32nd Aerospace Sciences Meeting and Exhibit*, 1994.

[78] R. K. Lobb, "Experimental measurement of shock detachment distance on spheres fired in air at hypervelocities," in *The High Temperature Aspects of Hypersonic Flow* (W. C. Nelson, ed.), vol. 68 of *AGARDograph*, ch. 26, pp. 519–527, Elsevier, 1964.

[79] Y. Dobrov, V. Gimadiev, A. Karpenko, and K. Volkov, "Numerical simulation of hypersonic flow with non-equilibrium chemical reactions around sphere," *Acta Astronautica*, vol. 194, pp. 468–479, 2022.

[80] R. W. MacCormack, "Carbuncle computational fluid dynamics problem for blunt-body flows," *Journal of Aerospace Information Systems*, vol. 10, no. 5, pp. 229–239, 2013.

[81] P. Cordesse, *Contribution to the study of combustion instabilities in cryotechnic rocket engines : coupling diffuse interface models with kinetic-based moment methods for primary atomization simulations*. Theses, Université Paris-Saclay, June 2020.

[82] T. Soubrié, *Prise en compte de l'ionisation et du rayonnement dans la modélisation des écoulements de rentrée terrestre et martienne*. PhD thesis, École nationale supérieure de l'aéronautique et de l'espace, 2006.

[83] F. Coquel and C. Marmignon, "A Roe-type linearization for the Euler equations for weakly ionized multi-component and multi-temperature gas," in *12th Computational Fluid Dynamics Conference*, 1995.

[84] M. Kim, A. Gülhan, and I. D. Boyd, "Modeling of electron energy phenomena in hypersonic flows," *Journal of Thermophysics and Heat Transfer*, vol. 26, no. 2, pp. 244–257, 2012.

[85] A. Blanco and E. Josyula, "Numerical modeling of hypersonic weakly ionized external flowfields with Poisson's equation," *AIAA Journal*, vol. 58, no. 8, pp. 3464–3475, 2020.

[86] C. Park, "Thermochemical relaxation in shock tunnels," *Journal of Thermophysics and Heat Transfer*, vol. 20, no. 4, pp. 689–698, 2006.

[87] X. Zhong, "Additive semi-implicit Runge-Kutta methods for computing high-speed nonequilibrium reactive flows," *Journal of Computational Physics*, vol. 128, no. 1, pp. 19–31, 1996.

[88] D. Z. Huang, P.-O. Persson, and M. J. Zahr, "High-order, linearly stable, partitioned solvers for general multiphysics problems based on implicit-explicit Runge-Kutta schemes," *Computer Methods in Applied Mechanics and Engineering*, vol. 346, pp. 674–706, 2019.

[89] J. Nievergelt, "Parallel methods for integrating ordinary differential equations," *Communications of the ACM*, vol. 7, pp. 731–733, Dec. 1964.

[90] J.-L. Lions, Y. Maday, and G. Turinici, "Résolution d'EDP par un schéma en temps «pararéel »," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 332, no. 7, pp. 661–668, 2001.

[91] M. J. Gander and S. Vandewalle, "Analysis of the Parareal time-parallel time-integration method," *SIAM Journal on Scientific Computing*, vol. 29, no. 2, pp. 556–578, 2007.

[92] T. Lunet, *Stratégies de parallélisation espace-temps pour la simulation numérique des écoulements turbulents*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace, 2018. Thèse de doctorat dirigée par Gratton, Serge et Bodart, Julien Mathématiques appliquées Toulouse, ISAE 2018.

[93] M. J. Gander and S. Güttel, "PARAEXP: a parallel integrator for linear initial-value problems," *SIAM Journal on Scientific Computing*, vol. 35, no. 2, 2013.

[94] A. Gopinath and A. Jameson, "Time spectral method for periodic unsteady computations over two and three dimensional bodies," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Jan. 2005.

[95] A. Gopinath and A. Jameson, "Application of the time spectral method to periodic unsteady vortex shedding," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, vol. 8, Jan. 2006.

[96] K. Ekici, R. Djeddi, H. Li, and J. I. Frankel, "Modeling periodic and non-periodic response of dynamical systems using an efficient Chebyshev-based time-spectral approach," *Journal of Computational Physics*, vol. 417, 2020.

[97] D. A. Pope, "An exponential method of numerical integration of ordinary differential equations," *Commun. ACM*, vol. 6, pp. 491–493, Aug. 1963.

[98] W. S. Edwards, L. S. Tuckerman, R. A. Friesner, and D. C. Sorensen, "Krylov methods for the incompressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 110, no. 1, pp. 82–102, 1994.

[99] M. Hochbruck and A. Ostermann, "Explicit exponential Runge–Kutta methods for semilinear parabolic problems," *SIAM Journal on Numerical Analysis*, vol. 43, no. 3, pp. 1069–1090, 2005.

[100] M. Hochbruck and A. Ostermann, "Exponential integrators," *Acta Numerica*, vol. 19, pp. 209–286, 2010.

[101] Q. Nie, Y.-T. Zhang, and R. Zhao, "Efficient semi-implicit schemes for stiff systems," *Journal of Computational Physics*, vol. 214, no. 2, pp. 521–537, 2006.

[102] H. P. Bhatt, A. Q. M. Khaliq, and B. A. Wade, "Efficient Krylov-based exponential time differencing method in application to 3D advection-diffusion-reaction systems," *Applied Mathematics and Computation*, vol. 338, pp. 260–273, 2018.

[103] M. Narayanamurthi and A. Sandu, "Efficient implementation of partitioned stiff exponential Runge-Kutta methods," *Applied Numerical Mathematics*, vol. 152, pp. 141–158, 2020.

[104] M. Caliari, F. Cassini, and F. Zivcovich, "BAMPHI: matrix-free and transpose-free action of linear combinations of φ-functions from exponential integrators," *Journal of Computational and Applied Mathematics*, vol. 423, p. 114973, 2023.

[105] A. Koskela and A. Ostermann, "Exponential Taylor methods: analysis and implementation," *Computers & Mathematics with Applications*, vol. 65, no. 3, pp. 487–499, 2013. Efficient Numerical Methods for Scientific Applications.

[106] M. Hochbruck, A. Ostermann, and J. Schweitzer, "Exponential Rosenbrock-type methods," *SIAM Journal on Numerical Analysis*, vol. 47, no. 1, pp. 786–803, 2009.

[107] Q. Du and W.-x. Zhu, "Stability analysis and application of the exponential time differencing schemes," *International Journal of Computer Mathematics - IJCM*, vol. 22, Mar. 2004.

[108] M. Hochbruck, A. Ostermann, and J. Schweitzer, "Explicit integrators of Rosenbrock-type," *Oberwolfach Rep.*, vol. 3, pp. 1107–1110, Apr. 2006.

[109] V. T. Luan, "Fourth-order two-stage explicit exponential integrators for time-dependent PDEs," *Applied Numerical Mathematics*, vol. 112, pp. 91–103, 2017.

[110] Y. Saad, "Analysis of some Krylov subspace approximations to the matrix exponential operator," *SIAM Journal on Numerical Analysis*, vol. 29, no. 1, pp. 209–228, 1992.

[111] R. B. Sidje, "Expokit: a software package for computing matrix exponentials," *ACM Trans. Math. Softw.*, vol. 24, pp. 130–156, Mar. 1998.

[112] M. Eiermann and O. G. Ernst, "A restarted Krylov subspace method for the evaluation of matrix functions," *SIAM Journal on Numerical Analysis*, vol. 44, no. 6, pp. 2481–2504, 2006.

[113] N. J. Higham and A. H. Al-Mohy, "Computing matrix functions," *Acta Numerica*, vol. 19, pp. 159–208, 2010.

[114] N. J. Higham, "The Scaling and Squaring method for the matrix exponential revisited," *SIAM Review*, vol. 51, no. 4, pp. 747–764, 2009.

[115] D. A. Kopriva, "A conservative staggered-grid Chebyshev multidomain method for compressible flows. II. a semi-structured method," *Journal of Computational Physics*, vol. 128, no. 2, pp. 475–488, 1996.

[116] Y. Liu, M. Vinokur, and Z. J. Wang, "Spectral difference method for unstructured grids I: basic formulation," *Journal of Computational Physics*, vol. 216, no. 2, pp. 780–801, 2006.

[117] K. Van den Abeele, C. Lacor, and Z. J. Wang, "On the stability and accuracy of the spectral difference method," *Journal of Scientific Computing*, vol. 37, pp. 162–188, Nov. 2008.

[118] J. Vanharen, G. Puigt, X. Vasseur, J.-F. Boussuge, and P. Sagaut, "Revisiting the spectral analysis for high-order spectral discontinuous methods," *Journal of Computational Physics*, vol. 337, pp. 379–402, 2017.

[119] A. Cassagne, "Implémentation multi GPU de la méthode spectral differences pour un code de CFD," Master's thesis, EPSI Bordeaux and CERFACS, 2014.

[120] A. Cassagne, G. Puigt, and J.-F. Boussuge, "High-order method for a new generation of large eddy simulation," tech. rep., joint PRACE project report from CERFACS (Toulouse) and Cines (Montpellier), 2015.

[121] G. Marait, "Performances du code de mécanique des fluides JAGUAR sur CPU et accélérateur," Master's thesis, Bordeaux INP ENSEIRB MATMECA and CERFACS, 2015.

[122] R. Fiévet, H. Deniau, and E. Piot, "Strong compact formalism for characteristic boundary conditions with discontinuous spectral methods," *Journal of Computational Physics*, vol. 408, p. 109276, 2020.

[123] J. I. Cardesa, R. Fiévet, E. Piot, H. Deniau, and C. Airiau, "Optimizing an acoustic liner by automatic differentiation of a compressible flow solver," *Journal of Computational Science*, vol. 61, p. 101703, 2022.

[124] R. Hartmann, A. Balan, F. Bassi, J.-F. Boussuge, A. de Brauer, J.-S. Cagnone, A. Colombo, V. Couaillier, O. Coulaud, A. Crivellini, M. Franciolini, A. Ghidoni, K. Hillewaert, M. de la Llave Plata, G. Manzinali, F. Naddei, G. Noventa, G. Puigt, B. C. Vermeire, and P. E. Vincent, "Space adaptive methods/meshing," in *TILDA: Towards Industrial LES/DNS in Aeronautics: Paving the Way for Future Accurate CFD - Results of the H2020 Research Project TILDA, Funded by the European Union, 2015 -2018* (C. Hirsch, K. Hillewaert, R. Hartmann, V. Couaillier, J.-F. Boussuge, F. Chalot, S. Bosniakov, and W. Haase, eds.), pp. 103–190, Cham: Springer International Publishing, 2021.

[125] A. Veilleux, G. Puigt, H. Deniau, and G. Daviller, "Stable spectral difference approach using Raviart-Thomas elements for 3D computations on tetrahedral grids," *Journal of Scientific Computing*, vol. 91, 2022.

[126] A. Veilleux, G. Puigt, H. Deniau, and G. Daviller, "A stable spectral difference approach for computations with triangular and hybrid grids up to the 6th order of accuracy," *Journal of Computational Physics*, vol. 449, p. 110774, 2022.

[127] T. Marchal, H. Deniau, J.-F. Boussuge, B. Cuenot, and R. Mercier, "Extension of the spectral difference method to combustion," 2021.

[128] H. Deniau and G. Puigt, "Entropy stable spectral difference scheme," in *European workshop on high order nonlinear numerical methods for evolutionary PDEs: theory and applications (HONOM2022), April 4-8, Braga, Portugal*, 2022.

[129] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Trans. Math. Softw.*, vol. 31, pp. 351–362, Sept. 2005.

[130] C. Bogey and C. Bailly, "A family of low dispersive and low dissipative explicit schemes for flow and noise computations," *Journal of Computational Physics*, vol. 194, no. 1, pp. 194–214, 2004.

[131] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of Computational Physics*, vol. 77, no. 2, pp. 439–471, 1988.

[132] S. Gottlieb and C.-W. Shu, "Total variation diminishing Runge–Kutta schemes.," *Mathematics of Computation*, vol. 67, Aug. 1996.

[133] R. J. Spiteri and S. J. Ruuth, "A new class of optimal high-order strong-stability-preserving time discretization methods," *SIAM Journal on Numerical Analysis*, vol. 40, no. 2, pp. 469–491, 2002.

[134] S. Gottlieb, C.-W. Shu, and E. Tadmor, "Strong stability-preserving high-order time discretization methods," *SIAM Review*, vol. 43, no. 1, pp. 89–112, 2001.

[135] E. Süli and D. F. Mayers, *An introduction to numerical analysis*. Cambridge University Press, 2003.

[136] T. Arts and M. Lambert de Rouvroit, "Aero-thermal performance of a two-dimensional highly loaded transonic turbine nozzle guide vane: a test case for inviscid and viscous flow computations," *Journal of Turbomachinery*, vol. 114, pp. 147–154, Jan. 1992.

[137] V. Brunet, E. Croner, A. Minot, J. de Laborderie, E. Lippinois, S. Richard, J.-F. Boussuge, J. Dombard, F. Duchaine, L. Gicquel, T. Poinsot, G. Puigt, G. Staffelbach, L. Segui, O. Vermorel, N. Villedieu, J.-S. Cagnone, K. Hillewaert, M. Rasquin, G. Lartigue, V. Moureau, V. Couaillier, E. Martin, M. de la Llave Plata, J.-M. Le Gouez, and F. Renac, "Comparison of various CFD vodes for LES simulations of turbomachinery: from inviscid vortex convection to multi-stage compressor," in *ASME Turbo Expo 2018: Turbomachinery Technical Conference and Exposition*, vol. Volume 2C: Turbomachinery of *Turbo Expo: Power for Land, Sea, and Air*, June 2018.

[138] T. Marchal, H. Deniau, J.-F. Boussuge, B. Cuenot, T. Mercier, RenaudMarchal, H. Deniau, J.-F. Boussuge, B. Cuenot, and R. Mercier, "Extension of the spectral difference method to premixed laminar and turbulent combustion," *Flow, Turbulence and Combustion (under review)*, 2023.

[139] P. Castillo and Y. Saad, "Preconditioning the matrix exponential operator with applications," *Journal of Scientific Computing*, vol. 13, Dec. 1997.

[140] S. Li, "A parallel discontinuous Galerkin method with physical orthogonal basis on curved elements," *Procedia Engineering*, vol. 61, pp. 144–151, 2013. 25th International Conference on Parallel Computational Fluid Dynamics.

[141] S.-J. Li, L.-S. Luo, Z. J. Wang, and L. Ju, "An exponential time-integrator scheme for steady and unsteady inviscid flows," *Journal of Computational Physics*, vol. 365, pp. 206–225, 2018.

[142] S.-J. Li, "Time advancement of the Navier-Stokes equations: p-adaptive exponential methods," *Journal of Flow Control, Measurement & Visualization*, vol. 08, pp. 63–76, Jan. 2020.

[143] M. Narayanamurthi and A. Sandu, "Partitioned exponential methods for coupled multiphysics systems," *Applied Numerical Mathematics*, vol. 161, pp. 178–207, 2021.

Abstracts

Résumé de la thèse en français

Cette thèse s'intéresse aux performances de l'intégration temporelle du code CEDRE sur des problèmes stationnaires. CEDRE est une plateforme logicielle visant la résolution des problèmes multi-fluides pour des applications en énergétique à échelle industrielle. Elle est composée de plusieurs solveurs, chacun dédié à un ensemble de phénomènes physiques. Plus précisément, nous regardons comment améliorer la rapidité, robustesse et convergence de l'intégration temporelle. Pour des raisons de stabilité nous nous intéressons à des méthodes implicites, en particulier à la méthode d'Euler implicite. Ces méthodes nécessitent la résolution de problèmes non-linéaires, qui nécessitent à leur tour la résolution de problèmes linéaires. Le passage de l'un à l'autre se fait par la présence de la Jacobienne des fonctions du problème non-linéaire. Une méthode de Krylov est déjà existante dans CEDRE pour l'inversion de systèmes linéaires: la méthode GMRES. Nous utilisons le fait qu'elle n'a pas explicitement besoin de la matrice pour l'inverser et mettons en place une méthode JFNK. Le but est d'améliorer la précision de la matrice Jacobienne utilisée, en espérant que cela améliorera la précision globale de l'intégration temporelle. Ceci est justifié par le fait qu'avant cette thèse la Jacobienne utilisée est très approximée, notamment en ce qui concerne les modélisations fines des solveurs, comme les termes sources turbulents, et les méthodes de reconstruction, comme les méthodes MUSCL. Une implémentation d'une méthode sans-matrice est mise en place de manière générique de sorte que tout solveur de CEDRE puisse utiliser cette formulation. Cela ouvre de plus la porte à une résolution implicite couplée des solveurs, chose non permise avec la structure actuelle de CEDRE. La méthode JFNK est comparée aux méthodes préexistantes de CEDRE sur des applications typiques de complexité croissantes choisies afin de représenter les fonctionnalités du solveur.

Dans un second temps, nous élargissons le contexte d'étude en nous intéressant aux méthodes d'intégration exponentielles, cette fois avec le solveur JAGUAR. Ce changement de solveur est justifié par la plus grande précision apportée par la méthode des Différences Spectrales utilisée comme schéma de discrétisation spatiale, précision nécessaire à l'analyse de ces nouveaux schémas temporels très précis. Le choix de s'intéresser aux méthodes exponentielle est justifié par le fait que ces méthodes réutilisent beaucoup des ingrédients de l'approche JFNK précédente. Nous choisissons, implémentons et analysons un ensemble de méthodes exponentielles, en comparaison à des méthodes déjà présentes, sur plusieurs cas pour montrer leur intérêt.

Thesis abstract in English

This thesis focuses on the performance of the time integration of the CEDRE code on steady problems. CEDRE is a software platform aimed at solving multi-fluid problems for industrial-scale energetics applications. It is composed of several solvers, each dedicated to a set of physical phenomena. More precisely, we are looking at improving speed, robustness and convergence of the time integration. For stability reasons, we are interested in implicit methods, in particular the implicit Euler method. These methods require solving nonlinear problems, which in turn requires solving linear problems. The transition from one to the other corresponds to the presence of the Jacobian matrix of functions from the non-linear problem. A Krylov method already exists in CEDRE for the inversion of linear systems: the GMRES method. We use the fact that it does not explicitly need the matrix to invert it and implement a JFNK method. The aim is to improve the accuracy of the Jacobian matrix used, in hope that this will improve the overall accuracy of the time integration. This is justified by the fact that prior to this thesis the Jacobian matrix used is very approximate, especially with respect to many of the fine modelling features of the solver, such as turbulent source terms, as well as reconstruction methods, such as the MUSCL one. A formulation of the matrix-free method is generically implemented so that any CEDRE solver can use this formulation. This also opens the door to coupled implicit solving, something not allowed with the current CEDRE structure. The JFNK method is compared to pre-existing CEDRE methods on typical applications of increasing complexity chosen to represent the functionality of the solver.

In a second step, we broaden the context by looking at exponential integration methods, this time with the JAGUAR solver. This change of solver is justified by the higher accuracy brought by the Spectral Difference method that JAGUAR uses as a spatial discretization scheme, necessary to analyze these new very accurate temporal methods. The choice to focus on exponential methods is justified by the fact that these methods reuse many of the ingredients of the previous JFNK approach. We select, implement and analyse a set of exponential methods, in comparison to existing methods, on several test cases to show their interest.

Méthodologies permettant l'obtention efficace de solutions multi-physiques stationnaires pour des applications en énergétique

Cette thèse s'intéresse aux performances de l'intégration temporelle du code CEDRE sur des problèmes stationnaires. CEDRE est une plateforme logicielle visant la résolution des problèmes multi-fluides pour des applications en énergétique à échelle industrielle. Elle est composée de plusieurs solveurs, chacun dédié à un ensemble de phénomènes physiques. Plus précisément, nous regardons comment améliorer la rapidité, robustesse et convergence de l'intégration temporelle. Pour des raisons de stabilité nous nous intéressons à des méthodes implicites, en particulier à la méthode d'Euler implicite. Ces méthodes nécessitent la résolution de problèmes non-linéaires, qui nécessitent à leur tour la résolution de problèmes linéaires. Le passage de l'un à l'autre se fait par la présence de la Jacobienne des fonctions du problème non-linéaire. Une méthode de Krylov est déjà existante dans CEDRE pour l'inversion de systèmes linéaires: la méthode GMRES. Nous utilisons le fait qu'elle n'a pas explicitement besoin de la matrice pour l'inverser et mettons en place une méthode JFNK. Le but est d'améliorer la précision de la matrice Jacobienne utilisée, en espérant que cela améliorera la précision globale de l'intégration temporelle. Ceci est justifié par le fait qu'avant cette thèse la Jacobienne utilisée est très approximée, notamment en ce qui concerne les modélisations fines des solveurs, comme les termes sources turbulents, et les méthodes de reconstruction, comme les méthodes MUSCL. Une implémentation d'une méthode sans-matrice est mise en place de manière générique de sorte que tout solveur de CEDRE puisse utiliser cette formulation. Cela ouvre de plus la porte à une résolution implicite couplée des solveurs, chose non permise avec la structure actuelle de CEDRE. La méthode JFNK est comparée aux méthodes préexistantes de CEDRE sur des applications typiques de complexité croissantes choisies afin de représenter les fonctionnalités du solveur.

Dans un second temps, nous élargissons le contexte en nous intéressant aux méthodes d'intégration exponentielles, cette fois avec le solveur JAGUAR. Ce changement de solveur est justifié par la plus grande précision apportée par la méthode des Différences Spectrales utilisée comme schéma de discrétisation spatiale, précision nécessaire à l'analyse de ces nouveaux schémas temporels très précis. Le choix de s'intéresser aux méthodes exponentielle est justifié par le fait que ces méthodes réutilisent beaucoup des ingrédients de l'approche JFNK précédente. Nous choisissons, implémentons et analysons un ensemble de méthodes exponentielles, en comparaison à des méthodes déjà présentes, sur plusieurs cas pour montrer leur intérêt.

Mots-clés : METHODE NUMERIQUE ; ENERGETIQUE ; MULTIPHYSIQUE ; STATIONNAIRE

Efficient methods for steady multi-physics simulations

This thesis focuses on the performance of the time integration of the CEDRE code on steady problems. CEDRE is a software platform aimed at solving multi-fluid problems for industrial-scale energetics applications. It is composed of several solvers, each dedicated to a set of physical phenomena. More precisely, we are looking at improving speed, robustness and convergence of the time integration. For stability reasons, we are interested in implicit methods, in particular the implicit Euler method. These methods require solving nonlinear problems, which in turn requires solving linear problems. The transition from one to the other corresponds to the presence of the Jacobian matrix of functions from the non-linear problem. A Krylov method already exists in CEDRE for the inversion of linear systems: the GMRES method. We use the fact that it does not explicitly need the matrix to invert it and implement a JFNK method. The aim is to improve the accuracy of the Jacobian matrix used, in hope that this will improve the overall accuracy of the time integration. This is justified by the fact that prior to this thesis the Jacobian matrix used is very approximate, especially with respect to many of the fine modelling features of the solver, such as turbulent source terms, as well as reconstruction methods, such as the MUSCL one. A formulation of the matrix-free method is generically implemented so that any CEDRE solver can use this formulation. This also opens the door to coupled implicit solving, something not allowed with the current CEDRE structure. The JFNK method is compared to pre-existing CEDRE methods on typical applications of increasing complexity chosen to represent the functionality of the solver.

In a second step, we broaden the context by looking at exponential integration methods, this time with the JAGUAR solver. This change of solver is justified by the higher accuracy brought by the Spectral Difference method that JAGUAR uses as a spatial discretization scheme, necessary to analyze these new very accurate temporal methods. The choice to focus on exponential methods is justified by the fact that these methods reuse many of the ingredients of the previous JFNK approach. We select, implement and analyse a set of exponential methods, in comparison to existing methods, on several test cases to show their interest.

Keywords : NUMERICAL METHODS ; ENERGETICS ; MULTIPHYSICS ; STEADY

ONERA
THE FRENCH AEROSPACE LAB

BP 80100 - Chemin de la Hunière FR-91123 PALAISEAU CEDEX - Tél. : +33 1 80 38 60 60 - Fax : +33 1 80 38 65 10

www.onera.fr