Automatic differentiation using operator overloading (ADOO) for implicit resolution of hyperbolic single phase and two-phase flow models

François FRAYSSE[a] and Richard Saurel[a,b]

[a] RS2N, Recherche Scientifique et Simulation Numérique, St Zacharie, France
[b] LMA, Laboratoire de Mécanique et d'Acoustique, UMR 7031 AMU - CNRS - Centrale Marseille, Marseille, France

Abstract: Implicit time integration schemes are widely used in computational fluid dynamics to speed-up computations. Indeed, implicit schemes usually allow for less stringent time-step stability constraints than their explicit counterpart. The derivation of an implicit scheme is however a challenging and time-consuming task, increasing substantially with the model equations complexity since this method usually requires fairly accurate evaluation of the spatial scheme's matrix Jacobian. This article presents a flexible method to overcome the difficulties associated to the computation of the derivatives, based on the forward mode of automatic differentiation using operator overloading (ADOO). Flexibility and simplicity of the method are illustrated through implicit resolution of various flow models of increasing complexity such as the compressible Euler equations, a two-phase flow model in full equilibrium (Le Martelot et al., 2014) and a symmetric variant (Saurel et al., 2003) of the two-phase flow model of (Baer & Nunziato, 1986) dealing with mixtures in total disequilibrium.

Keywords: automatic differentiation, implicit, two-phase, finite volume, unstructured meshes

I. Introduction

In the early days of numerical simulation, the industry growing needs for fast and accurate fluid flow predictions lead to numerous developments in the field of Computational Fluid Dynamics (CFD). Among them, in the aerospace community for instance, implicit time integrators arose from the motivation of computational time saving required to reach steady-state, first for the Euler equations (Mulder & Van Leer, 1983) and later for the laminar compressible Navier-Stokes equations (Venkatakrishnan & Barth, 1989) and for the Reynolds-Averaged Navier-Stokes equations (Barth & Linton, 1995). Towards better fidelity and thanks to the improvement in computational resources, the development of fast numerical schemes for unsteady flows became necessary and feasible. Taking advantage of the numerous acceleration techniques developed for the explicit time integrators solving steady-state flows, discrete unsteady equations have been solved as successive stationary problems at each time-step (Jameson, 1991). However, for problems where the physical time scale is greater than the spatial scale divided by the eigenvalue, fully implicit methods are known to be more efficient (Pulliam, 1993) (Dubuc et al., 1998).

Implicit methods for unsteady flows are known to suffer from higher diffusive and dispersive errors. Nevertheless, developments of high-order implicit methods such as those based on backward Taylor series or implicit Runge-Kutta (Gottlieb et al., 2001) opened the way to implicit schemes applied to higher fidelity situations including direct numerical simulation (Martin & Candler, 2006) (Liu et al., 2016).

One of the major difficulties with implicit schemes is that they require the derivatives of the spatial numerical scheme with respect to the flow variables. The complexity of the involved expressions rapidly grows with the model equations and spatial scheme sophistication such as for example, Riemann problem-based methods. This fact motivated the development of various methods requiring only a crude approximation of the Jacobian

matrix such as for example alternative direction implicit (Briley & McDonald, 1977) where the implicit operator is dimensionally split (restricted to structured grids) or the lower upper symmetric Gauss-Seidel method where the operator is decomposed in lower and upper dominant factors (Jameson & Turkel, 1981). These approximations while decreasing the development time usually severely degrade stability of the implicit scheme yielding effective time-steps much smaller than the full unfactored form or slow convergence of the unsteady residual.

While appealing, approximation of the matrix Jacobian through finite differentiation is an expensive strategy and subject to round-off errors not easily controlled. Krylov subspace methods for solving linear systems (Saad & Schultz, 1986) lead to matrix-free implicit algorithms known as Newton-Krylov-matrix-free. In this approach, the matrix Jacobian is not explicitly built, indeed the Krylov methods such as for example the generalized minimal residual method (GMRes) involves only matrix Jacobian-vector products which can directly be approximated through finite differentiation. Nevertheless, it is well-known that the convergence rate of the GMRes method can be highly improved by the application of a preconditioner. However, forming a good preconditioner without the explicit form of the matrix is not a trivial task and the whole method is still subject to numerical error affecting both the linear and the non-linear convergence rate.

Automatic Differentiation (AD) (Wengert, 1964) (Griewank & Walther, 2000) is an algorithmic method for evaluating derivatives up to any order by propagating the chain rule to a high-level language computer program. AD combines the main advantages of symbolic and numerical differentiation without their major drawbacks. In contrast to numerical differentiation it provides exact derivatives (up to round-off error) and is less computationally demanding. AD performs on computer algorithms directly and as such is able to differentiate any complex function as well as numerical procedures such as root-finding methods. Two main approaches to AD are usually considered: AD based on source code transformation (ADSCT) and AD based on operator overloading (ADOO), the key point of the present contribution.

- ADSCT is a procedure that takes as an argument a function or a routine of a computer code, parses all basic operations and intrinsic functions, and returns another function or routine containing the tangent or adjoint derivatives which can be compiled together with the original code. This method is quite easy to apply to any programming language. The generated code complexity does not increase with respect to the original code allowing efficient compiler time optimizations. This method is however rather complex to implement. Examples of codes based on this approach include ADIFOR (Bischof et al., 1996) and TAPENADE (Hascoet & Pascual, 2013).
- ADOO determines the derivatives present in a computer code by appending a second component to all dependent variables. This variable contains the derivatives of the expression with respect to the independent variables and is propagated by applying elementary differentiation arithmetic based on the chain rule. ADOO benefits from object-oriented programming and more precisely from the ability of defining operators on user derived data-types, which is a computer language dependent feature. Programs based on this technique include ADOL-C and ADOL-F for C and Fortran programming language respectively (Walther & Griewank, 2012).

Implicit time integration in conjunction with AD has been used in the past: (Hovland & McInnes, 2001) and (Bramkamp, et al., 2006) in the context of Newton-Krylov method applied to the steady state Euler equations used ADIFOR. (Xia, et al., 2014) used Tapenade to compute the Jacobian matrix associated to their WENO-DG discretization of Euler and laminar Navier-Stokes equations. Implicit discretization of MHD equations using OpenAD (Utke, et al., 2008) has been carried out by (Reynolds, et al., 2012). Sediment transport equation solved implicitly using Tapenade has been performed in (Bilanceri, et al., 2012).

AD tools such as for example Tapenade or ADOL-C can be used in the so-called forward or reverse (adjoint) mode depending on the number of independent variables. Reverse mode is algorithmically more complicated than the forward mode but more computationally efficient when the number of independent variables is much

higher than the number of dependent variables. This situation arises in many problems of optimization where the sensitivities of several objective functions with respect to numerous design parameters have to be evaluated, see for example (Espath, et al., 2011), (Mohammadi & Pironneau, 2004), (Newman, et al., 1999). On the other hand, when the number of independent variables is small, as is the case for Jacobian of flux functions, the algorithmically simpler forward mode is preferred.

In this article we will show that forward AD mode can be implemented in existing numerical codes in a straightforward manner without requiring external packages. It will be shown how AD can be applied to the computation of complex matrix Jacobians and how it considerably simplifies the design of implicit schemes for sophisticated systems of equations.

The paper is organized as follows. In Section II, the derivation of a general implicit scheme targeting unsteady flows is presented. AD applied to the evaluation of the matrix Jacobian is detailed in Section III with practical examples in Fortran programming language. Then, various fluid flow models of increasing complexity are discretized implicitly and tested in Section IV, including the compressible Euler equations, a two-phase flow model in full equilibrium (Le Martelot et al., 2014) and a symmetric variant (Saurel et al., 2003) of the two-phase flow model of (Baer & Nunziato, 1986), widely used to model non-equilibrium mixtures.

II. Implicit time integration

The fluid flow models considered in the present article can be written under the following general differential form:

$$\frac{\partial \mathbf{Q}}{\partial t} + \mathrm{div}\left(\overline{\overline{\mathrm{F}}}\left(\mathbf{Q}\right)\right) + \overline{\overline{\mathrm{G}}}\left(\mathbf{Q}\right)\cdot\mathrm{div}\left(\overline{\overline{\mathrm{H}}}\left(\mathbf{Q}\right)\right) = 0 \qquad (1)$$

where $\mathbf{Q}$ is the vector of conservative variables, $\mathrm{div}\left(\overline{\overline{\mathrm{F}}}\left(\mathbf{Q}\right)\right)$ the conservative part and $\overline{\overline{\mathrm{G}}}\left(\mathbf{Q}\right)\cdot\mathrm{div}\left(\overline{\overline{\mathrm{H}}}\left(\mathbf{Q}\right)\right)$ the non-conservative part of the system. The discrete finite volume form is obtained by first integrating (1) over a time-independent element Ω_i of boundary $\partial\Omega_i$ and applying the Green-Ostrogradski theorem to the conservative part:

$$\int_{\Omega_i}\frac{\partial \mathbf{Q}}{\partial t}\mathrm{dV}_{\Omega_i} + \int_{\partial\Omega_i}\mathbf{F}\left(\mathbf{Q}\right)\cdot\mathbf{n}_{\partial\Omega_i}\mathrm{dS}_{\partial\Omega_i} + \int_{\Omega_i}\overline{\overline{\mathrm{G}}}\left(\mathbf{Q}\right)\cdot\mathrm{div}\left(\overline{\overline{\mathrm{H}}}\left(\mathbf{Q}\right)\right)\mathrm{dV}_{\Omega_i} = 0 \qquad (2)$$

Then, assuming second-order quadrature rule for the surface and volume integrals, averaging the tensor $\overline{\overline{\mathrm{G}}}\left(\mathbf{Q}\right)$ over the domain Ω_i and introducing approximate Riemann fluxes $\mathbf{F}^*$ and $\mathbf{H}^*$, system (2) can be rewritten as:

$$\mathrm{V}_i\frac{\mathrm{d}\overline{\mathbf{Q}}_i}{\mathrm{dt}} + \sum_{\mathrm{f}_{ij}\in\partial\Omega_i}\mathbf{F}^*\left(\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_k\right\}_i\right),\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_k\right\}_j\right)\right)\cdot\mathbf{n}_{\mathrm{f}_{ij}}\mathrm{S}_{\mathrm{f}_{ij}} + \mathbf{G}\left(\overline{\mathbf{Q}}_i\right)\sum_{\mathrm{f}_{ij}\in\partial\Omega_i}\mathbf{H}^*\left(\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_k\right\}_i\right),\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_k\right\}_j\right)\right)\cdot\mathbf{n}_{\mathrm{f}_{ij}}\mathrm{S}_{\mathrm{f}_{ij}} = 0 \qquad (3)$$

In the above system, $\overline{\mathbf{Q}}_i$ is the volume average of $\mathbf{Q}$ defined as $\overline{\mathbf{Q}}_i = \frac{1}{\mathrm{V}_i}\int_{\Omega_i}\mathbf{Q}\mathrm{dV}_{\Omega_i}$, f_{ij} is a linear surface of normal unit $\mathbf{n}_{\mathrm{f}_{ij}}$ separating elements indexed by i and j and pointing towards element j. Function $\mathrm{g}(\ )$ is an interpolation operator reconstructing left and right states at face f_{ij} satisfying a maximum principle. This

function takes as argument a set of conservative vectors forming a compact stencil centered in cell i for the set $\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{i}}$ and in cell j for the set $\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{j}}$.

The numerical approximation (3) with respect to non-conservative terms in the present formulation is particularly simple, but appropriate for various two-phase flow models with non-equilibrium effects (Saurel et al., 2009) (Furfaro & Saurel, 2015). As shown in these references appropriate Riemann solvers considering these terms have to be addressed.

The implicit discrete temporal integration using Backward-Difference-Formula (BDF) under $\theta-\phi$ form gives the following non-linear system for the unknown set of vectors $\left\{\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}\right\}_{\mathrm{i}}=\bigcup_{\mathrm{f_{ij}}}\left(\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{i}}\bigcup\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{j}}\right)$ forming the domain of dependence of cell i:

$$\begin{aligned}\mathbf{P}_{\mathrm{i}}\left(\left\{\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}\right\}_{\mathrm{i}}^{\mathrm{n+1}}\right)=&\,\mathrm{V_i}\frac{1+\phi}{\theta}\frac{\overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n+1}}-\overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n}}}{\Delta\mathrm{t}}+\sum_{\mathrm{f_{ij}}\in\partial\Omega_{\mathrm{i}}}\mathbf{F}^{*}\left(\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{i}}^{\mathrm{n+1}}\right),\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{j}}^{\mathrm{n+1}}\right)\right)\cdot\mathbf{n}_{\mathrm{f_{ij}}}\mathrm{S}_{\mathrm{f_{ij}}}\\&+\mathbf{G}\left(\overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n+1}}\right)\sum_{\mathrm{f_{ij}}\in\partial\Omega_{\mathrm{i}}}\mathbf{H}^{*}\left(\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{i}}^{\mathrm{n+1}}\right),\mathrm{g}\left(\left\{\overline{\mathbf{Q}}_{\mathrm{k}}\right\}_{\mathrm{j}}^{\mathrm{n+1}}\right)\right)\cdot\mathbf{n}_{\mathrm{f_{ij}}}\mathrm{S}_{\mathrm{f_{ij}}}\\&-\mathrm{V_i}\frac{\phi}{\theta}\frac{\overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n}}-\overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n-1}}}{\Delta\mathrm{t}}=0\end{aligned}\qquad(4)$$

In the above system, the first-order BDF method is obtained using the couple $(\theta,\phi)=(1,0)$ and the second-order BDF uses the couple $(\theta,\phi)=(1,1/2)$.

It can be shown that BDF of order 1 (BDF1) and 2 (BDF2) are L-stable, furthermore BDF1 is unconditionally Strong Stability Preserving (SSP) for the linear case (Total Variation Diminishing-stable). However, BDF2 and all other high-order methods (>1) are only conditionally SSP (Gottlieb, et al., 2001), e.g. it exists a constant $\mathrm{c}>0/\left\|\mathbf{Q}^{\mathrm{n+1}}\right\|<\max\left(\left\|\mathbf{Q}^{\mathrm{n}}\right\|,...,\left\|\mathbf{Q}^{\mathrm{n+1-s}}\right\|\right),\text{for }\Delta\mathrm{t}<\mathrm{c}\Delta\mathrm{t}_{\mathrm{EE}}$ where $\Delta\mathrm{t}_{\mathrm{EE}}$ is the maximum time step for TVD Explicit Euler integration. This constant c is not easily bounded in the general case, and often $\Delta\mathrm{t}$ is bounded by criteria based on relevant physics of the problem under study.

System (4) written for all cells of the mesh represents a coupled non-linear set of equations for the next time-step unknown vector $\overline{\mathbf{Q}}^{\mathrm{n+1}}$. Its solution is approximated by the iterative Newton-Raphson (NR) method. NR method iteration is written as $\overline{\mathbf{Q}}^{\mathrm{r+1}}=\overline{\mathbf{Q}}^{\mathrm{r}}+\Delta\overline{\mathbf{Q}}^{\mathrm{r}}$ where the increment $\Delta\overline{\mathbf{Q}}^{\mathrm{r}}$ is given by the solution of the linear system $\frac{\partial\mathbf{P}}{\partial\overline{\mathbf{Q}}^{\mathrm{r}}}\Delta\overline{\mathbf{Q}}^{\mathrm{r}}=-\mathbf{P}\left(\overline{\mathbf{Q}}^{\mathrm{r}}\right)$. At convergence of the NR method, e.g. when $\left\|\frac{\Delta\overline{\mathbf{Q}}^{\mathrm{r}}}{\overline{\mathbf{Q}}^{\mathrm{n}}}\right\|<\varepsilon_{\mathrm{NL}}$ ($\varepsilon_{\mathrm{NL}}$ is a tolerance parameter) the solution at the next time-step is obtained: $\overline{\mathbf{Q}}^{\mathrm{n+1}}=\lim_{\left\|\frac{\Delta\overline{\mathbf{Q}}^{\mathrm{r}}}{\overline{\mathbf{Q}}^{\mathrm{n}}}\right\|\to\varepsilon_{\mathrm{NL}}}\overline{\mathbf{Q}}^{\mathrm{r+1}}$.

The NR method thus requires a sequence of linear problems to solve each time-step involving the derivative of the function $\mathbf{P}$ with respect to the conservative variables $\frac{\partial \mathbf{P}}{\partial \overline{\mathbf{Q}}^{r}}$. In compact form, the Newton iteration is obtained through the solution of the following linear system:

$$\begin{aligned}\left(\frac{1+\phi}{\theta}\frac{V_i}{\Delta t}\mathbf{I}_d+\mathbf{J}\right)\Delta\overline{\mathbf{Q}}_i^{r} = &-\sum_{f_{ij}\in\partial\Omega_i}\mathbf{F}^{*}\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\\ &-\mathbf{G}\left(\overline{\mathbf{Q}}_i^{r}\right)\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^{*}\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\\ &-V_i\frac{1+\phi}{\theta}\frac{\overline{\mathbf{Q}}_i^{s}-\overline{\mathbf{Q}}_i^{n}}{\Delta t}+V_i\frac{\phi}{\theta}\frac{\overline{\mathbf{Q}}_i^{n}-\overline{\mathbf{Q}}_i^{n-1}}{\Delta t}\end{aligned}\qquad(5)$$

where the matrix Jacobian $\mathbf{J}$ gathers all the conservative flux and non-conservative terms derivatives present in the implicit operator $\frac{\partial \mathbf{P}}{\partial \overline{\mathbf{Q}}^{r}}$. In case of unstructured grids, $\mathbf{J}$ is a sparse non-symmetric block-matrix whose level of fill-in depends on the union of all spatial domains of dependence which is equal to $\mathrm{card}\left(\bigcup_i\left\{\left\{\overline{\mathbf{Q}}_k\right\}\right\}_i\right)$. Let us consider $\overline{\mathbf{Q}}_m\subset\left\{\left\{\overline{\mathbf{Q}}_k\right\}\right\}_i$, the block matrix $\mathbf{J}_{im}$ of $\mathbf{J}$ is related to $\frac{\partial \mathbf{P}}{\partial \overline{\mathbf{Q}}^{r}}$ through the following equalities:

$$\begin{aligned}\text{if } m\neq i,\ \left.\frac{\partial\mathbf{P}_i}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}} &= \sum_{f_{ij}\in\partial\Omega_i}\left.\frac{\partial\mathbf{F}^{*}}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)}\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}+\mathbf{G}\left(\overline{\mathbf{Q}}_i^{r}\right)\sum_{f_{ij}\in\partial\Omega_i}\left.\frac{\partial\mathbf{H}^{*}}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)}\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\\ &=\mathbf{J}_{im}\end{aligned}$$

$$\begin{aligned}\text{if } m=i,\ \left.\frac{\partial\mathbf{P}_i}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}} &= \sum_{f_{ij}\in\partial\Omega_i}\left.\frac{\partial\mathbf{F}^{*}}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)}\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}+G\left(\overline{\mathbf{Q}}_i^{r}\right)\sum_{f_{ij}\in\partial\Omega_i}\left.\frac{\partial\mathbf{H}^{*}}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)}\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\\ &\quad+\left.\frac{\partial\mathbf{G}}{\partial\overline{\mathbf{Q}}_m^{r}}\right|_{\overline{\mathbf{Q}}_i^{r}}\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^{*}\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}+V_i\frac{1+\phi}{\theta}\mathbf{I}_d\\ &=\mathbf{J}_{im}+V_i\frac{1+\phi}{\theta}\mathbf{I}_d\end{aligned}$$

Performing a single Newton iteration results in the so-called linearized implicit method, which requires only one linear system resolution per time-step. Such approximation allows significant computational savings, but its accuracy is clearly restricted to the case where the magnitude of the unsteady residual is lower than the temporal scheme's truncation error: $\left\|\frac{\Delta\overline{\mathbf{Q}}^{r=1}}{\overline{\mathbf{Q}}^{n}}\right\|\approx O\left(\Delta t^{k}\right)$. Nevertheless, this condition is fulfilled in the following situations:

- Weak temporal non-linearity;
- Accurate estimation of the initial guess in the Newton's algorithm;

- Accurate evaluation of the Jacobian matrix.

Backward difference formula of order two is not a self-starting method as it requires the solution at time n-1. Consequently, the numerical integration for the first time-step needs another discretization scheme. In this paper, the first-order backward difference formula is used to start the numerical integration.

The second implicit scheme considered in this paper is a semi-implicit scheme of the Runge-Kutta family. More precisely, the two-step second-order Strong Stability Preserving Singly Diagonally Implicit Runge Kutta scheme (SSPSDIRK-2) (Kennedy & Carpenter, 2016). This is a multi-step method and as such is self-starting. The semi-implicit terminology refers to the fact that the various Runge-Kutta steps are decoupled, resulting in the present case in two sequential non-linear equations per time-step. This scheme can be written in the following compact form:

$$\underbrace{\begin{aligned}\left(\frac{1+\phi}{\theta}\frac{V_i}{\Delta t}\mathbf{I}_d + a_{11}\mathbf{J}\right)\Delta\overline{\mathbf{Q}}_i^{r,1} = -a_{11}&\left(\sum_{f_{ij}\in\partial\Omega_i}\mathbf{F}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r,1}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r,1}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}} + \mathbf{G}\left(\overline{\mathbf{Q}}_i^{r,1}\right)\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r,1}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r,1}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\right)\\ &- V_i\frac{\overline{\mathbf{Q}}_i^{r,1}-\overline{\mathbf{Q}}_i^{n}}{\Delta t}\\ &\text{with}\quad \overline{\mathbf{Q}}_i^{1,1}=\overline{\mathbf{Q}}_i^{n},\ \overline{\mathbf{Q}}_i^{r+1,1}=\overline{\mathbf{Q}}_i^{r,1}+\Delta\overline{\mathbf{Q}}_i^{r,1},\ \overline{\mathbf{Q}}_i^{1}=\lim_{\Delta\overline{\mathbf{Q}}_i^{r,1}\to 0}\overline{\mathbf{Q}}_i^{r+1,1}\end{aligned}}_{\textbf{Step 1}}$$

$$\underbrace{\begin{aligned}\left(\frac{1+\phi}{\theta}\frac{V_i}{\Delta t}\mathbf{I}_d + a_{22}\mathbf{J}\right)\Delta\overline{\mathbf{Q}}_i^{r,2} = -a_{11}&\left(\sum_{f_{ij}\in\partial\Omega_i}\mathbf{F}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r,2}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r,2}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}} + \mathbf{G}\left(\overline{\mathbf{Q}}_i^{r,2}\right)\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{r,2}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{r,2}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\right)\\ &- a_{21}\left(\sum_{f_{ij}\in\partial\Omega_i}\mathbf{F}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{1}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{1}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}} + \mathbf{G}\left(\overline{\mathbf{Q}}_i^{1}\right)\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{1}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{1}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\right)\\ &- V_i\frac{\overline{\mathbf{Q}}_i^{r,2}-\overline{\mathbf{Q}}_i^{n}}{\Delta t}\\ &\text{with}\quad \overline{\mathbf{Q}}_i^{1,2}=\overline{\mathbf{Q}}_i^{1},\ \overline{\mathbf{Q}}_i^{r+1,2}=\overline{\mathbf{Q}}_i^{r,2}+\Delta\overline{\mathbf{Q}}_i^{r,2},\ \overline{\mathbf{Q}}_i^{2}=\lim_{\Delta\overline{\mathbf{Q}}_i^{r,2}\to 0}\overline{\mathbf{Q}}^{r+1,2}\end{aligned}}_{\textbf{Step 2}}$$

$$\overline{\mathbf{Q}}_i^{n+1} = \overline{\mathbf{Q}}_i^{n} + \frac{\Delta t}{V_i}\sum_{s=1}^{2} b_s\left(\sum_{f_{ij}\in\partial\Omega_i}\mathbf{F}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{s}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{s}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}} + \mathbf{G}\left(\overline{\mathbf{Q}}_i^{s}\right)\sum_{f_{ij}\in\partial\Omega_i}\mathbf{H}^*\left(g\left(\left\{\overline{\mathbf{Q}}_k\right\}_i^{s}\right),g\left(\left\{\overline{\mathbf{Q}}_k\right\}_j^{s}\right)\right)\cdot\mathbf{n}_{f_{ij}}S_{f_{ij}}\right)$$

with

$$a = \begin{pmatrix} 1-\frac{\sqrt{2}}{2} & 0 \\ \sqrt{2}-1 & 1-\frac{\sqrt{2}}{2}\end{pmatrix}, b = \begin{pmatrix}\frac{1}{2}\\ \frac{1}{2}\end{pmatrix}$$

For moderately and highly unsteady flows, accurate Jacobian matrix computation is of primary importance to reach good convergence rate in the Newton's method (up to quadratic).

III. Evaluation of the Jacobian matrix

1. Introduction

Two main strategies are usually used to compute the Jacobian matrix of the spatial numerical scheme:

- Analytic differentiation. This approach involves both hand and symbolic differentiation of the numerical flux and overall scheme. The main advantage of this approach is the full determination and control of the elements of the matrix resulting generally in an accurate and efficient Jacobian evaluation. Various drawbacks are however present. First, the complexity of the differentials grows rapidly with the model and numerical scheme sophistication making this method error-prone. The level of flexibility is very low since any slight modification in the model or in the numerical scheme needs an adaptation of the Jacobian matrix. Extra difficulties arise when the numerical scheme involves root-finding algorithms such as sophisticated boundary conditions and non-explicit equations of state (EOS). Symbolic differentiation is somehow more flexible but often at the cost of rather complicated expressions thus resulting in significant computational overhead.
- Numerical differentiation. This method is based on finite differencing and as such is not an exact method. The aim is to approximate the implicit operator as,

$$\frac{\partial \mathbf{P}}{\partial \overline{\mathbf{Q}}^{\mathrm{r}}} \approx \frac{\mathbf{P}(\overline{\mathbf{Q}}^{\mathrm{r}} + \varepsilon_{\mathrm{FD}}) - \mathbf{P}(\overline{\mathbf{Q}}^{\mathrm{r}})}{\varepsilon_{\mathrm{FD}}}$$

where $\varepsilon_{\mathrm{FD}}$ represents a numerical tolerance. While being flexible and easy to implement, this approach has two major drawbacks. First, the computational cost is relatively high as one evaluation of the function P is necessary for each degree of freedom. The second drawback lies in the estimate for $\varepsilon_{\mathrm{FD}}$. Indeed, its value is of fundamental importance as too high or too small values yield inaccurate derivative approximations because of truncation and round-off errors respectively (Dennis & Schnabel, 1983). Round-off error inherent to finite differentiation can be avoided using complex numbers, see (Squire & Trapp, 1998), but needs extensive recoding. This approach gained however a lot of interest and success when used together with Krylov subspace linear solvers (Brown & Saad, 1990). Indeed, in these solvers the implicit operator only appears as a product with the solution increment $\Delta\overline{\mathbf{Q}}^{\mathrm{r}}$. Consequently, the full matrix does not need explicit computation, instead the matrix-vector product is approximated through finite differencing:

$$\frac{\partial \mathbf{P}}{\partial \mathbf{Q}^{\mathrm{r}}} \Delta\mathbf{Q}^{\mathrm{r}} \approx \frac{\mathbf{P}(\mathbf{Q}^{\mathrm{r}} + \varepsilon_{\mathrm{FD}} \Delta\mathbf{Q}^{\mathrm{r}}) - \mathrm{P}(\mathbf{Q}^{\mathrm{r}})}{\varepsilon_{\mathrm{FD}}}$$

While being attractive in terms of computational speed with respect to the full matrix approximation, this method still suffers of non-trivial estimate of $\varepsilon_{\mathrm{FD}}$. Inaccurate approximation of the implicit operator can severely degrade the convergence rate of the Newton's method or accuracy of the linearized implicit method. Furthermore, it is well known that the convergence rate of the iterative Krylov subspace linear solvers degrades rapidly when the diagonal dominance decreases (e.g. when the time-step increases). Convergence rate of the linear solver is usually significantly increased by the application of a preconditioner which tends to decrease the spectral radius of the implicit operator. Efficient fully matrix-free preconditioning is nonetheless not trivial and is still an active research area.

2. Automatic differentiation

As we shall see in the following, forward mode of ADOO can be implemented in a straightforward manner into an existing Fortran code. Let us first consider the following simple example:

$$f(x) = \sin(x) + x^2$$
$$f'(x) = \cos(x) + 2x$$

The first step consists in replacing the real variables x and f by two objects x_{AD} and f_{AD} having two components $\begin{bmatrix} v \\ dv \end{bmatrix}$: v being the value of the variable itself and dv the value of its derivative. The Fortran 90 standard allows the construction of such objects, more commonly called derived data types (DDT). The Fortran 2003 standard allows each component of a DDT, and in particular dv, to be an array which gives a greater flexibility in handling derivatives with respect to various independent variables. The evaluation of the derivative following the ADOO approach is performed in three steps:

$$\underbrace{x_{AD} = \begin{bmatrix} x \\ 1 \end{bmatrix}}_{\text{Initialization}} \Rightarrow f_{AD}(x_{AD}) = \underbrace{\sin_{AD}\left(\begin{bmatrix} x \\ 1 \end{bmatrix}\right) +_{AD} \begin{bmatrix} x \\ 1 \end{bmatrix}^{2_{AD}}}_{\text{Operator overloading}} = \underbrace{\begin{bmatrix} \sin(x) \\ \cos(x) \end{bmatrix} +_{AD} \begin{bmatrix} x^2 \\ 2x \end{bmatrix} = \begin{bmatrix} \sin(x) + x^2 \\ \cos(x) + 2x \end{bmatrix} \equiv \begin{bmatrix} f(x) \\ f'(x) \end{bmatrix}}_{\text{Propagation}}$$

- Initialization. It consists in copying the value of x in the component v of DDT x_{AD} and initializing the component dv to 1, indeed in this case $dv = \dfrac{dx}{dx} = 1$
- Operator overloading. A DDT is a construct defined by the programmer. As such, no rules are predefined for operations involving one or more DDTs and they need to be coded. Fortunately, the number of basic operations and intrinsic functions is finite, and their coding can be done once and for all. In the above example, rules must be defined for the operators $\sin_{AD}, +_{AD}, (\ )^{2_{AD}}$. Definitions of these rules is the core of the ADOO method and will be detailed hereafter.
- Propagation. Once all needed operators and intrinsic functions are overloaded they can be applied to any DDT and the derivatives then propagate through the chain rule.

To better explain how ADOO works, let us consider the one-dimensional compressible Euler equations closed by the ideal gas equation of state (EOS):

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{Q})}{\partial x} = 0 \text{ with } \mathbf{Q} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}, \mathbf{F}(\mathbf{Q}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(\rho E + p) \end{pmatrix} \text{ and } E = e(\rho, p) + \frac{1}{2}u^2 = \frac{p}{\rho(\gamma - 1)} + \frac{1}{2}u^2$$

where ρ, u, p, E, e, γ denote the density, the velocity, the pressure, the total energy, the internal energy and the polytropic gas constant respectively. Let us assume a BDF1 numerical scheme in time and a first-order finite volume discretization in space on a uniform grid indexed by i and of spacing Δx. A single Newton iteration (linearized implicit) simplifies the non-linear system (4) to the following tridiagonal linear system to be solved at each time-step:

$$\left(\frac{\Delta x}{\Delta t}\mathbf{I}_d + \frac{\partial \mathbf{F}^{*^n}_{i,i+1}}{\partial \overline{\mathbf{Q}}^n_i} - \frac{\partial \mathbf{F}^{*^n}_{i-1,i}}{\partial \overline{\mathbf{Q}}^n_i}\right)\Delta \overline{\mathbf{Q}}^n_i - \frac{\partial \mathbf{F}^{*^n}_{i-1,i}}{\partial \overline{\mathbf{Q}}^n_{i-1}}\Delta \overline{\mathbf{Q}}^n_{i-1} + \frac{\partial \mathbf{F}^{*^n}_{i,i+1}}{\partial \overline{\mathbf{Q}}^n_{i+1}}\Delta \overline{\mathbf{Q}}^n_{i+1} = -\left[\mathbf{F}^{*^n}_{i,i+1} - \mathbf{F}^{*^n}_{i-1,i}\right]$$
$$\text{with } \mathbf{F}^{*^n}_{i,i+1} \equiv \mathbf{F}^*\left(\overline{\mathbf{Q}}^n_i, \overline{\mathbf{Q}}^n_{i+1}\right) \text{ and } \Delta \overline{\mathbf{Q}}^n_i = \overline{\mathbf{Q}}^{n+1}_i - \overline{\mathbf{Q}}^n_i \qquad (6)$$

Let us approximate the intercell fluxes by the two-waves Riemann solver of Rusanov (Rusanov, 1962) which has a relatively low algorithmic complexity:

$$\mathbf{F}^{*}\left(\overline{\mathbf{Q}}_{i}^{n},\overline{\mathbf{Q}}_{i+1}^{n}\right)=\frac{1}{2}\left(\mathbf{F}\left(\overline{\mathbf{Q}}_{i}^{n}\right)+\mathbf{F}\left(\overline{\mathbf{Q}}_{i+1}^{n}\right)-S_{MAX}\left(\overline{\mathbf{Q}}_{i+1}^{n}-\overline{\mathbf{Q}}_{i}^{n}\right)\right)$$

$$\text{with } S_{MAX}=MAX\left(\left|u_{i}^{n}\right|+c_{i}^{n},\left|u_{i+1}^{n}\right|+c_{i+1}^{n}\right) \text{ and } c_{i}^{n}=\sqrt{\frac{\gamma p_{i}^{n}}{\rho_{i}^{n}}}$$

Let us focus on the evaluation of the local Jacobian matrix $\frac{\partial \mathbf{F}_{i,i+1}^{*n}}{\partial \overline{\mathbf{Q}}_{i}^{n}}$ appearing in (6) in the special case $S_{max}=u_{i}^{n}+c_{i}^{n},u_{i}^{n}>0$. Omitting the time superscript n and average symbol $\bar{}$, the exact Jacobian obtained by hand differentiation is given by:

$$\frac{\partial \mathbf{F}_{i,i+1}^{*n}}{\partial \overline{\mathbf{Q}}_{i}^{n}}=\begin{pmatrix} \frac{u_i+c_i}{2}-\left(\frac{\gamma(\gamma-1)}{4\rho_i c_i}\left(u_i^2-E_i\right)-\frac{u_i}{2\rho_i}\right)(\rho_{i+1}-\rho_i) & \frac{1}{2}-\left(\frac{1}{2\rho_i}-\frac{\gamma(\gamma-1)}{4\rho_i c_i}u_i\right)(\rho_{i+1}-\rho_i) & -\frac{\gamma(\gamma-1)}{4\rho_i c_i}(\rho_{i+1}-\rho_i) \\ \frac{\gamma-3}{4}u_i^2-\left(\frac{\gamma(\gamma-1)}{4\rho_i c_i}\left(u_i^2-E_i\right)-\frac{u_i}{2\rho_i}\right)(\rho_{i+1}u_{i+1}-\rho_i u_{i+1}) & \frac{3-\gamma}{2}u_i+u_i+c_i-\left(\frac{1}{2\rho_i}-\frac{\gamma(\gamma-1)}{4\rho_i c_i}u_i\right)(\rho_{i+1}u_{i+1}-\rho_i u_i) & \frac{\gamma-1}{2}-\frac{\gamma(\gamma-1)}{4\rho_i c_i}(\rho_{i+1}u_{i+1}-\rho_i u_i) \\ \frac{(\gamma-1)u_i^3-\gamma u_i E_i}{2}-\left(\frac{\gamma(\gamma-1)}{4\rho_i c_i}\left(u_i^2-E_i\right)-\frac{u_i}{2\rho_i}\right)(\rho_{i+1}E_{i+1}-\rho_i E_{i+1}) & \frac{2\gamma E_i-3(\gamma-1)u_i^2}{4}-\left(\frac{1}{2\rho_i}-\frac{\gamma(\gamma-1)}{4\rho_i c_i}u_i\right)(\rho_{i+1}E_{i+1}-\rho_i E_i) & \frac{(\gamma+1)u_i+c_i}{2}-\frac{\gamma(\gamma-1)}{4\rho_i c_i}(\rho_{i+1}E_{i+1}-\rho_i E_i) \end{pmatrix} \quad (7)$$

In order to compare this hand differentiated Jacobian to the one obtained by ADOO in Fortran language, the first step consists in defining a DDT as shown in Figure 1.

```
PUBLIC :: AutoDiffType
TYPE AutoDiffType
 REAL(KIND=RP)              ::  v
 REAL(KIND=RP)              ::  dv(NCons,Nderiv)
END TYPE AutoDiffType
```

Figure 1: Fortran definition of a derived data type.

Note the dimension of the component dv which corresponds to the total number of independent variables. In the case of one-dimensional Euler equations NCons represents the number of conservative variables which is equal to 3 and Nderiv the number of conservative vectors the intercell flux depends on. Nderiv is equal to 2 in the case of first-order spatial discretization $\left(\overline{\mathbf{Q}}_{i}^{n},\overline{\mathbf{Q}}_{i+1}^{n}\right)$.

The second step consists in overloading operators and intrinsic functions needed to compute the intercell flux in order to handle computations involving DDTs. Examples are given in Figure 2.

```
PURE ELEMENTAL SUBROUTINE d_assign_d(d1,d2)
TYPE(AutoDiffType),  INTENT(OUT) :: d1
TYPE(AutoDiffType),  INTENT( IN) :: d2

d1%v  = d2%v
d1%dv = d2%dv

END SUBROUTINE d_assign_d

PURE ELEMENTAL SUBROUTINE d_assign_r(d,r)
TYPE(AutoDiffType),  INTENT(OUT) :: d
REAL(KIND=RP)     ,  INTENT( IN) :: r

d%v  = r
d%dv = ZERO

END SUBROUTINE d_assign_r

PURE ELEMENTAL SUBROUTINE r_assign_d(r,d)
REAL(KIND=RP)     ,  INTENT(OUT) :: r
TYPE(AutoDiffType),  INTENT( IN) :: d

r = d%v

END SUBROUTINE r_assign_d
```

```
PURE ELEMENTAL FUNCTION d_times_d(d1, d2)
TYPE(AutoDiffType), INTENT(IN) :: d1, d2
TYPE(AutoDiffType)             :: d_times_d

d_times_d%v  = d1%v*d2%v
d_times_d%dv = d1%v*d2%dv + d2%v*d1%dv

END FUNCTION d_times_d

PURE ELEMENTAL FUNCTION d_times_r(d, r)
TYPE(AutoDiffType), INTENT(IN) :: d
REAL(KIND=RP)     , INTENT(IN) :: r
TYPE(AutoDiffType)             :: d_times_r

d_times_r%v  = r*d%v
d_times_r%dv = r*d%dv

END FUNCTION d_times_r

PURE ELEMENTAL FUNCTION r_times_d(r, d)
TYPE(AutoDiffType), INTENT(IN) :: d
REAL(KIND=RP)     , INTENT(IN) :: r
TYPE(AutoDiffType)             :: r_times_d

r_times_d%v  = r*d%v
r_times_d%dv = r*d%dv

END FUNCTION r_times_d
```

```
PURE ELEMENTAL FUNCTION SIN_d(d)
TYPE(AutoDiffType), INTENT(IN) :: d
TYPE(AutoDiffType)             :: SIN_d

SIN_d%v = SIN(d%v)
SIN_d%dv = d%dv * COS(d%v)

END FUNCTION SIN_d

PURE ELEMENTAL FUNCTION COS_d(d)
TYPE(AutoDiffType), INTENT(IN) :: d
TYPE(AutoDiffType)             :: COS_d

COS_d%v = COS(d%v)
COS_d%dv = -d%dv * SIN(d%v)

END FUNCTION COS_d

PURE ELEMENTAL FUNCTION EXP_d(d)
TYPE(AutoDiffType), INTENT(IN) :: d
TYPE(AutoDiffType)             :: EXP_d

EXP_d%v  = EXP(d%v)
EXP_d%dv = d%dv*EXP(d%v)

END FUNCTION EXP_d
```

Figure 2: Examples of operator and function overloading. These functions allow the use of basic arithmetic with REAL type, AutoDiffType type or mix of both.

Let us focus on the function d_times_d described in Figure 2. This function takes as arguments two DDTs and returns a DDT whose component v holds the product of the values and the component dv holds the conventional derivative of the product. Note the keyword ELEMENTAL, which is a Fortran 95 standard and allows to call this function with d1 and d2 being scalar DDTs or arrays of DDTs in a transparent way. The functions d_times_r and r_times_d are necessary when multiplying a DDT by a REAL constant. Note that similar rules may be defined for the product of an INTEGER by a DDT and vice versa.

At this point, all basic arithmetic operators and functions in the code involving DDTs would have to be rewritten to use the aforementioned routines. Fortunately, in Fortran, an automatic selection of the right function can be achieved through interfacing, see example in Figure 3.

```
PUBLIC :: ASSIGNMENT(=)
INTERFACE ASSIGNMENT(=)
 MODULE PROCEDURE d_assign_d
 MODULE PROCEDURE d_assign_r
 MODULE PROCEDURE r_assign_d
END INTERFACE
```

```
PUBLIC :: OPERATOR(*)
INTERFACE OPERATOR(*)
 MODULE PROCEDURE d_times_d
 MODULE PROCEDURE d_times_r
 MODULE PROCEDURE r_times_d
END INTERFACE
```

```
PUBLIC :: SIN
INTERFACE SIN
 MODULE PROCEDURE SIN_d
END INTERFACE

PUBLIC :: COS
INTERFACE COS
 MODULE PROCEDURE COS_d
END INTERFACE

PUBLIC :: EXP
INTERFACE EXP
 MODULE PROCEDURE EXP_d
END INTERFACE
```

Figure 3: Example of operator and function overloading. Interface blocks allowing the use of the symbols =,* and the intrinsic functions SIN,COS and EXP whether the arguments are of type REAL or type AutoDiffType or mix of both.

The selection is based on the type of arguments at compile time.

Once all required operators and functions are properly overloaded, the last step is to initialize the component dv of the independent variables to the identity matrix $\left(\frac{\partial \overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n}}}{\partial \overline{\mathbf{Q}}_{\mathrm{i}}^{\mathrm{n}}} = \frac{\partial \overline{\mathbf{Q}}_{\mathrm{i+1}}^{\mathrm{n}}}{\partial \overline{\mathbf{Q}}_{\mathrm{i+1}}^{\mathrm{n}}} = \mathbf{I}_{\mathrm{d}} \right)$ as shown in Figure 4.

```
SUBROUTINE InitDerivatives(VcL,VcR)
TYPE(AutoDiffType),INTENT(INOUT) :: VcL(:),VcR(:)

VcL(VC_Rho)%dv(:,:)=ZERO
VcL(VC_RhoU)%dv(:,:)=ZERO
VcL(VC_RhoE)%dv(:,:)=ZERO

VcR(VC_Rho)%dv(:,:)=ZERO
VcR(VC_RhoU)%dv(:,:)=ZERO
VcR(VC_RhoE)%dv(:,:)=ZERO

VcL(VC_Rho)%dv(VC_Rho,LEFT)=ONE
VcL(VC_RhoU)%dv(VC_RhoU,LEFT)=ONE
VcL(VC_RhoE)%dv(VC_RhoE,LEFT)=ONE

VcR(VC_Rho)%dv(VC_Rho,RIGHT)=ONE
VcR(VC_RhoU)%dv(VC_RhoU,RIGHT)=ONE
VcR(VC_RhoE)%dv(VC_RhoE,RIGHT)=ONE

END SUBROUTINE InitDerivatives
```

Figure 4: Initialization of the derivatives of the left and right vectors of conservative variables prior application of the chain rule.

Finally, a call to the routine computing the Rusanov flux (Figure 5) will automatically compute all the derivatives of the flux with respect to the left and to the right vectors of conservative variables: the dv component of the DDT variable "Flux(:)" holds all the necessary derivatives. For example, $\frac{\partial F^*_{\rho_{L,R}}}{\partial \rho_L}$ is stored in Flux(VC_Rho)%dv(VC_Rho,LEFT) and $\frac{\partial F^*_{\rho_{L,R}}}{\partial \rho_R}$ is stored in Flux(VC_Rho)%dv(VC_Rho,RIGHT).

```
SUBROUTINE RusanovFlux(VcL,VcR,Flux)
TYPE(AutoDiffType),INTENT(IN)::VcL(:),VcR(:)
TYPE(AutoDiffType),INTENT(OUT)::Flux(:)
TYPE(AutoDiffType)::rhoL,uL,pL,EL,cL
TYPE(AutoDiffType)::rhoR,uR,pR,ER,cR
TYPE(AutoDiffType)::Smax
TYPE(AutoDiffType),DIMENSION(Ncons)::fluxL,fluxR

rhoL=VcL(VC_Rho)
uL=VcL(VC_RhoU)/rhoL
EL=VcL(VC_RhoE)/rhoL
pL=rhoL*(GAMMA-ONE)*(EL-HALF*uL**2)

rhoR=VcR(VC_Rho)
uR=VcR(VC_RhoU)/rhoR
ER=VcR(VC_RhoE)/rhoR
pR=rhoR*(GAMMA-ONE)*(ER-HALF*uR**2)

cL=SQRT(GAMMA*pL/rhoL)
cR=SQRT(GAMMA*pR/rhoR)

Smax=MAX(ABS(uL)+cL,ABS(uR)+cR)

fluxL=[rhoL*uL,rhoL*uL**2+pL,uL*(rhoL*EL+pL)]
fluxR=[rhoR*uR,rhoR*uR**2+pR,uR*(rhoR*ER+pR)]

Flux(:)=HALF*(fluxL(:)+fluxR(:)-Smax*(VcR(:)-VcL(:)))

END SUBROUTINE RusanovFlux
```

Figure 5: Fortran routine for the computation of the Rusanov flux as well as all its derivatives with respect to the left and right vectors of conservative variables. It is worth to note that except the declaration of the variables where TYPE(AutoDiffType) is inserted, the rest of the subroutine corresponds to the conventional one used in the explicit version of the method.

Without loss of generality, let us detail the computation of $\frac{\partial F^{*}_{\rho_{L,R}}}{\partial \rho_L}$. After the initialization step detailed in Figure 4, the left and right dv components of the conservative vectors V_{c_L}, V_{c_R} are filled with the following values:

$$V_{c_L}(VC\_Rho) \Rightarrow \begin{cases} v: \rho_L \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}^T \end{cases} \quad V_{c_R}(VC\_Rho) \Rightarrow \begin{cases} v: \rho_R \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}^T \end{cases}$$

$$V_{c_L}(VC\_RhoU) \Rightarrow \begin{cases} v: \rho_L u_L \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}^T \end{cases} \quad V_{c_R}(VC\_RhoU) \Rightarrow \begin{cases} v: \rho_R u_R \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^T \end{cases}$$

$$V_{c_L}(VC\_RhoE) \Rightarrow \begin{cases} v: \rho_L E_L \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}^T \end{cases} \quad V_{c_R}(VC\_RhoE) \Rightarrow \begin{cases} v: \rho_R E_R \\ dv: \begin{matrix} L \\ R \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \end{cases}$$

In the Rusanov flux routine of Figure 5, the first instruction after the variables declarations (rhoL=VcL(VC_RHO)) is a DDT copy which is allowed since the assignment operator has been overloaded. This operation copies the components v and dv of $V_{c_L}(VC\_Rho)$ into the components v and dv of the variable ρ_L. The next line extracts the left-face side value of the velocity u_L from the conservative vector. Overloading the division operator gives the following value for the derivative of u_L with respect to ρ_L:

$$u_L = \frac{V_{c_L}(VC\_RhoU)}{V_{c_L}(VC\_Rho)} \Rightarrow u_L\%dv(1,1) = \frac{V_{c_L}(VC\_RhoU)\%dv(1,1) \times V_{c_L}(VC\_Rho)\%v - V_{c_L}(VC\_RhoU)\%v \times V_{c_L}(VC\_Rho)\%dv(1,1)}{\left(V_{c_L}(VC\_Rho)\%v\right)^2}$$

$$\Rightarrow u_L\%dv(1,1) = \frac{0 \times \rho_L - \rho_L u_L \times 1}{(\rho_L)^2} = \frac{-u_L}{\rho_L}$$

The same result is obtained by hand calculations. Denoting $m = \rho u$, the velocity is obtained as $u = \frac{m}{\rho}$.

Hence,

$$\frac{\partial u_L}{\partial \rho_L} = \frac{\partial\left(\frac{m_L}{\rho_L}\right)}{\partial \rho_L} = -\frac{m_L}{\rho_L^2} = -\frac{\rho_L u_L}{\rho_L^2} = -\frac{u_L}{\rho_L}$$

Proceeding the same way for the next lines and assuming $S_{MAX} = u_L + c_L$, the following intermediate derivatives are obtained:

$$E_L\%dv(1,1) = \frac{-E_L}{\rho_L}$$

$$p_L\%dv(1,1) = \frac{(\gamma-1)}{2} u_L^2$$

$$c_L\%dv(1,1) = \frac{\gamma(\gamma-1)}{4\rho_L c_L} - \frac{c_L}{2\rho_L}$$

$$S_{MAX}\%dv(1,1) = \frac{-u_L}{\rho_L} + \frac{\gamma(\gamma-1)}{4\rho_L c_L} - \frac{c_L}{2\rho_L}$$

Finally, the derivative of the Rusanov flux with respect to the left density is obtained as:

$$\frac{\partial F^{*}_{L,R}}{\partial \rho_L} = \frac{u_L + c_L}{2} - \left(\frac{\gamma(\gamma-1)u_L^2}{8\rho_L c_L} - \frac{u_L}{2\rho_L} - \frac{c_L}{4\rho_L} \right)(\rho_R - \rho_L)$$

which after some elementary algebraic manipulation gives the same expression as the one given in (7) obtained by hand differentiation.

IV. Numerical examples

In this section numerical integration using time implicit schemes is performed on various flow models and various spatial schemes to demonstrate the flexibility offered by the automatic differentiation. One-dimensional and two-dimensional computations are considered using unstructured meshes decomposed in subdomains using Metis graph partitioning (Karypis & Kumar, 1998). Simulations are performed in parallel using the MPI protocol and the GNU-gfortran compiler. The linear systems arising from the implicit time discretization are solved by the PETSc libraries (Balay et al., 2018) using block-Jacobi preconditioned GMRes solver (Saad & Schultz, 1986).

One-dimensional and two-dimensional Euler equations are addressed first on a shock-tube test problem followed by an unsteady transonic flow past a cylinder. Various test problems using a two-phase flow model in equilibrium and a two-phase flow model in disequilibrium are carried out next.

1. Euler equations

a. 1D Sod shock-tube test-case

The one-dimensional shock-tube test of Sod (1978) is performed first. The computational configuration is composed of a tube of 1m length with a membrane located at x=0.5m separating a chamber at atmospheric conditions on the left side and a low-pressure chamber on the right side. Air governed by the ideal gas EOS is considered with polytropic coefficient $\gamma = 1.4$. The pressure is set to $p = 10^5 \mathrm{Pa}$ to the left of the membrane and $p = 10^4 \mathrm{Pa}$ to the right while the densities are set to $\rho = 1 \mathrm{kg.m^{-3}}$ and $\rho = 0.125 \mathrm{kg.m^{-3}}$ respectively. The computational domain is discretized with 10 000 uniform cells. First-order finite volume computations are carried out using explicit Euler integration scheme with a time-step controlled by the CFL stability constraint $\mathrm{CFL} = \lambda \frac{\Delta t}{\Delta x} = 0.5$, λ being the fastest wave velocity over the domain at a given time $\lambda = \mathrm{MAX}_{\Omega}\left(|u^n| + c^n \right)$. The time explicit numerical solution is compared to BDF1 with a single Newton iteration for increasing CFL numbers: 10,50 and 100.

Four numerical flux functions are tested in the present study: the approximate two-waves Riemann solver of Rusanov (Rusanov, 1962), the flux splitting scheme AUSM$^+$ (Liou, 1996), the three-wave approximate Riemann solver HLLC (Toro et al., 1994) and the Godunov scheme based on the exact Riemann problem solution (Godunov, 1959). The matrix Jacobian of all schemes is built using the ADOO method presented in Section III.2. While the algorithmic complexity of the Rusanov and HLLC solvers allow a relatively easy symbolic differentiation (see Rinaldi et al., (2014) for the implicit HLLC scheme), the AUSM$^+$ (see Colonia et al., (2014) for analytic derivatives) and the Godunov scheme require a fairly amount of work. The AUSM$^+$ scheme involves numerous conditional branches for the evaluation of the split Mach number and split pressure.

The Godunov scheme is based on the exact solution of the Riemann problem (Godunov, 1959) and involves a root-finding algorithm for the determination of the star pressure, where symbolic differentiation often requires some assumptions at this stage. In contrast, ADOO differentiates the entire numerical scheme, propagating

the derivatives inside the fixed-point iterative algorithm in a fully transparent way. The only precaution to be made in the automatic differentiation of root-finding algorithms of Newton type used in the Riemann solver is to set a tolerance allowing both function and function derivative to be converged, the latter generally having a slower rate (see Appendix B).

Density plots comparing explicit Euler and BDF1 time integration schemes are presented in Figure 6, for the Rusanov, HLLC, AUSM$^+$ and Godunov flux functions. Excellent stability for all schemes is obtained even for relatively high CFL number. Unconditional SSP property in the linear case for BDF1 together with exact Jacobians transpose very well in this non-linear case.

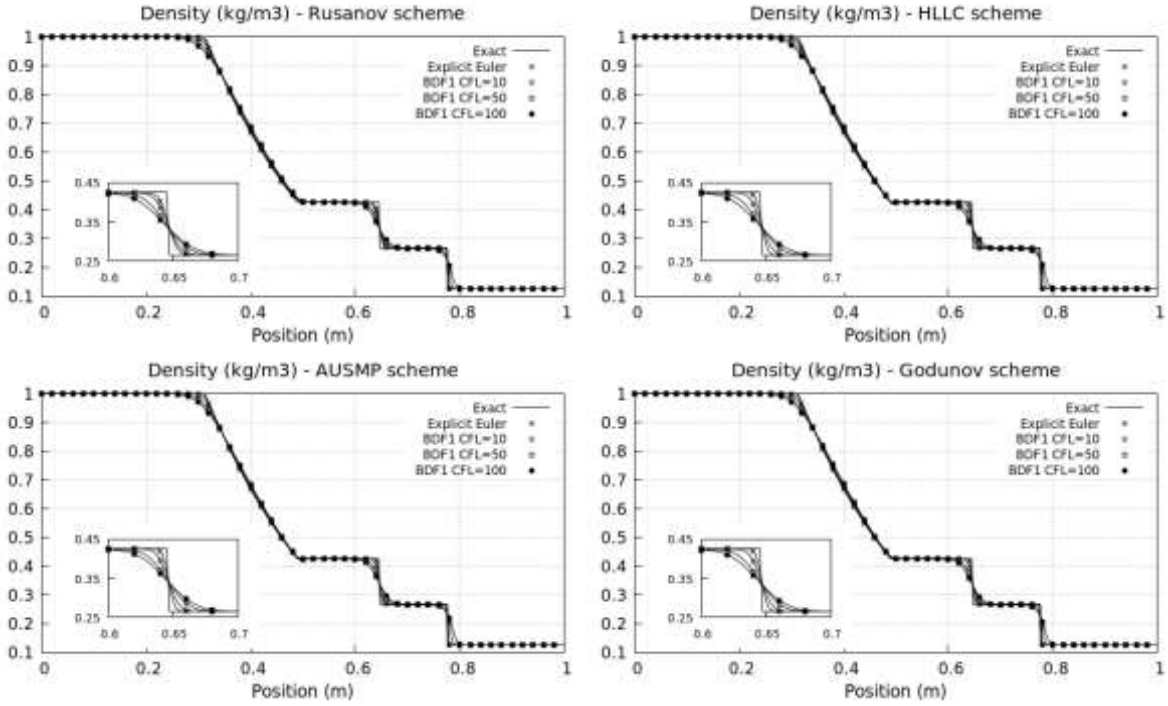

Figure 6: 1D Sod shock-tube problem, density plot. Comparison of explicit Euler and BDF1 numerical solutions using the Rusanov, HLLC, AUSMP and Godunov (left to right, top to bottom respectively) schemes with various CFL numbers.

BDF1 with HLLC scheme and CFL=100 results in computation speed-up of about 10. Note that in the present case, these 1D simulations have been carried out in the unstructured multi-dimensional code DALPHADT using the general GMRes linear solver provided by PETSc libraries. As such, the computational gain is representative of higher dimensions simulations.

ADOO time overhead has been measured with respect to analytic differentiation using the Rusanov flux function on a purely structured grid arrangement as well. Total simulation time has been recorded using both methods and use of ADOO showed negligible impact. Indeed, a time overhead lower than 0.1% with the fast block-tridiagonal Thomas algorithm has been observed. In multi-dimensional simulations with sparse non-symmetric matrices and general linear solvers, this overhead is expected to become even smaller.

b. 2D transonic flow past a circular cylinder

The second test-case using the compressible Euler equations is a 2D transonic flow around a circular cylinder of 1m diameter. The computational domain is rectangular (see Figure 7) with non-reflective boundary conditions (BC) assuming low-amplitude waves through algebraic acoustic relations. The circular cylinder BC assumes slip condition, consistent with the inviscid Euler equations. Free-stream BC as well as initial condition (IC) uses a Mach number (Ma) of 0.5 and an angle of attack of 0°. The domain is discretized with 50000 triangles.

A second-order finite volume spatial scheme is used. Cell-center gradients are computed using weighted least-squares based on a stencil composed of all cells sharing a vertex with the control volume. Multi-dimensional limiting process is ensured using a vertex-based extension (Park et al., 2010) of the Barth and Jespersen limiter (Barth & Jespersen, 1989). The implicit operator is built using ADOO assuming dependence on direct neighbors only, e.g. assuming the set $\left\{\left\{\overline{\mathbf{Q}}_k\right\}\right\}_i$ composed of cells sharing a face with cell i (often referred as first-order Jacobian in the literature).

Indeed, full Jacobian evaluation needs the differentiation of the reconstructed states with respect to the whole set of average variables used in the gradient computation. This almost ensures quadratic convergence of the Newton algorithm, but dramatically increases memory requirements to store the extra blocks in the matrix. Indeed, on a regular 2D grids composed of triangles $\mathrm{card}\left(\left\{\left\{\overline{\mathbf{Q}}_k\right\}\right\}_i\right)=22$ using weighted least-squares gradient approximation together with vertex-based limiter of (Park et al., 2010), resulting in 22 non-zero block matrices per row. Restriction of $\left\{\left\{\overline{\mathbf{Q}}_k\right\}\right\}_i$ to direct neighbors only involves a Jacobian matrix composed of 4 non-zero blocks per row. This assumption nevertheless introduces a slight inconsistency between the Jacobian matrix and the right-hand-side resulting in the loss of quadratic convergence of the non-linear residual. This also has the effect of slowing down the convergence rate of the linear solver by decreasing the diagonal dominance property. Consequently, a second order Jacobian, despite ensuring quadratic convergence of the non-linear residual, is usually globally more expensive than first-order Jacobian which only yields linear Newton convergence rate.

Second-order spatial accuracy is ensured through the right-hand-side of the linear system where fluxes are computed from reconstructed states, independently of the non-linear residual magnitude if it is below the temporal truncation error.

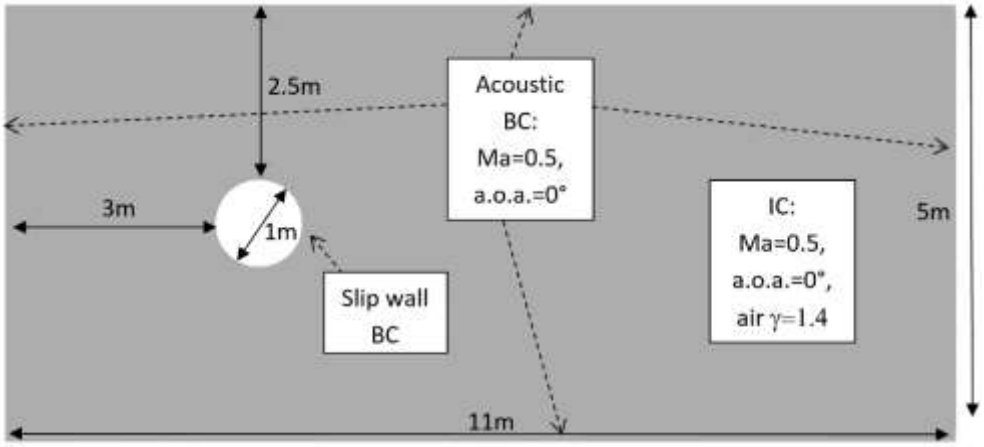

Figure 7: Single-phase transonic flow past a circular cylinder. Computational set-up.

Under these operating conditions, the flow becomes transonic near the top of the cylinder and attached shock-waves form downstream, ahead of the rear stagnation point. Due to the important total pressure loss across the shock and the high curvature, vorticity is generated, and the flow detaches from the wall resulting in an unsteady periodic flow (Salas, 1983) (see solution snapshots in Figure 8).

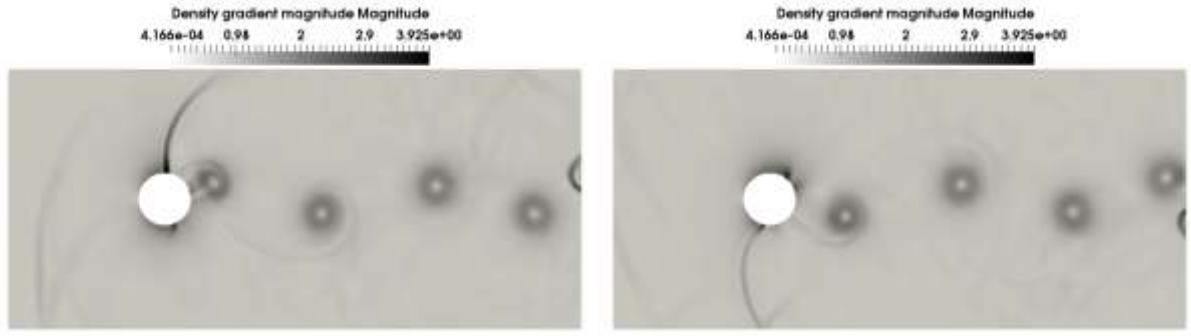

Figure 8: Single-phase transonic flow past a circular cylinder. Contours of density gradients magnitude when periodic regime is reached. Solutions are displayed with a time interval of 0.01s when periodic regime is reached. BDF1-AUSM$^+$ scheme on a mesh composed of 50000 triangles.

Explicit and implicit computations have been carried out using the Rusanov, AUSM$^+$ and HLLC schemes for a physical time of 0.3s. Ideal gas EOS is used with a polytropic coefficient $\gamma = 1.4$. The explicit scheme employs a second-order two-step SSP-Explicit-Runge-Kutta method (SSPERGK-2) (Gottlieb et al., 2001).

The lift coefficient is computed as $C_L = \dfrac{\int_s p\vec{n}.\vec{e}_y dS}{\frac{1}{2}\rho_\infty u_\infty^2 S_w}$, with ρ_∞, u_∞ the free-stream density and velocity and $S_w = 1m^2$ the projected wetted surface. Lift coefficient is plotted as a function of time using AUSM$^+$ approximate Riemann solver, SSPERGK-2 with CFL=0.5 and one-iteration BDF1 with CFL=10 and CFL=20 in Figure 9 (left).

The results agree with those of Pandolfi & Larocca, (1989). The lift coefficient values obtained with the first-order implicit scheme are very close to those given by the second-order explicit scheme despite a slight phase lag increasing with the numerical integration time-step. This phase lag can be highly decreased using the second-order BDF2 scheme at the cost of solving the non-linear problem up to a reasonable tolerance, see Figure 9 (right). The Newton-BDF2 scheme with second-order spatial reconstruction is in fact a quasi-Newton approach due to the first-order Jacobian assumption, and only linear convergence has been obtained. Newton algorithm is stopped when the first criterion is met: a decrease factor of 10^6 to the magnitude of the non-linear residual or a maximum of 10 iterations. Newton-BDF2 is however a rather costly method and no computational gain has been observed with respect to the explicit integration. On the other hand, a factor of 8 in computational time has been obtained using BDF1 (CFL=20) with respect to SSPERGK-2 on a 16-cores Intel Xeon E5-2687W-v2.

Same computations have been carried out using the Rusanov and HLLC fluxes and results are displayed in Figure 10. Explicit and implicit results are in overall good agreement.

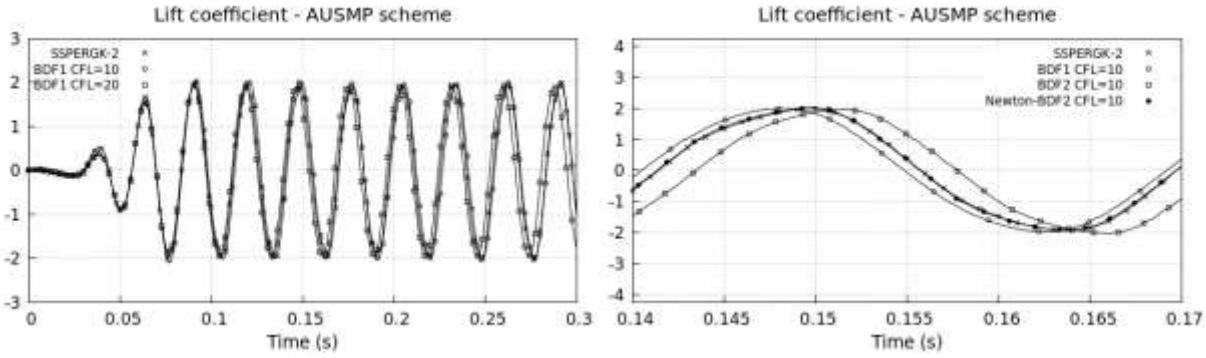

Figure 9: 2D transonic flow past a circular cylinder. Left: comparison of lift coefficient using explicit and implicit schemes based on AUSM$^+$ spatial flux. Right: BDF2 and Newton-BDF2 schemes are compared to BDF1 and SSPERGK-2.

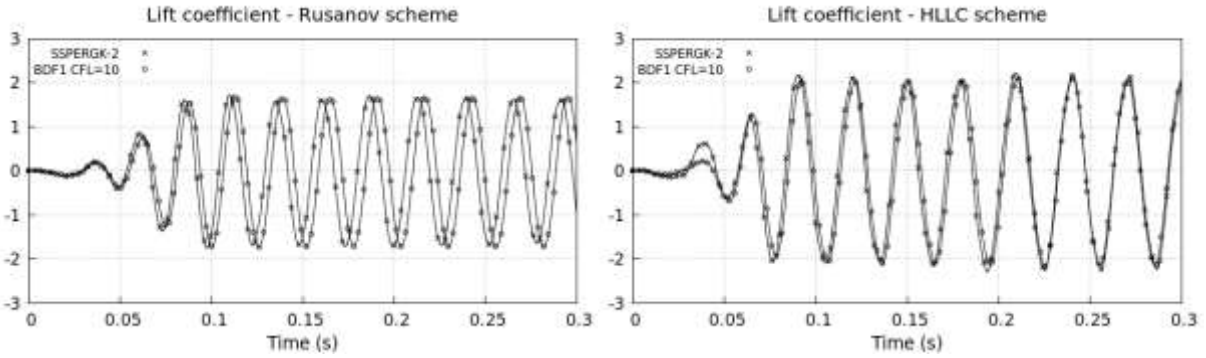

Figure 10: 2D transonic flow past a circular cylinder. Comparison of lift coefficient using explicit and implicit schemes based on Rusanov and HLLC spatial fluxes (left and right respectively).

All the tested implicit schemes are in good agreement with the explicit solution and results of the literature.

ADOO based implicit schemes are now considered for more sophisticated flow models for which derivation of numerical fluxes is a challenging task.

2. Two-phase flow models in full equilibrium and full disequilibrium

Two compressible two-phase flow models are presented in this section. The implicit discretization using ADOO is used and verified on the numerical simulation of various test problems.

a. Model in full disequilibrium

The symmetric variant (Saurel et al., 2003) of the compressible two-phase flow model of (Baer & Nunziato, 1986) in the absence of mass and heat transfer reads in the following 1D compact form:

$$\frac{\partial \mathbf{Q}}{\partial \mathrm{t}} + \frac{\partial \mathbf{F}(\mathbf{Q})}{\partial \mathrm{x}} + \mathbf{G}(\mathbf{Q}) \frac{\partial \mathbf{H}(\mathbf{Q})}{\partial \mathrm{x}} = \mathbf{S}(\mathbf{Q}) \qquad (8)$$

with

$$\mathbf{Q}=\begin{pmatrix}\alpha_1\\ \alpha_1\rho_1\\ \alpha_1\rho_1 u_1\\ \alpha_1\rho_1 E_1\\ \alpha_2\rho_2\\ \alpha_2\rho_2 u_2\\ \alpha_2\rho_2 E_2\end{pmatrix},\mathbf{F}(\mathbf{Q})=\begin{pmatrix}0\\ \alpha_1\rho_1 u_1\\ \alpha_1\left(\rho_1 u_1^2+p_1\right)\\ \alpha_1 u_1\left(\rho_1 E_1+p_1\right)\\ \alpha_2\rho_2 u_2\\ \alpha_2\left(\rho_2 u_2^2+p_2\right)\\ \alpha_2 u_2\left(\rho_2 E_2+p_2\right)\end{pmatrix},\mathbf{G}(\mathbf{Q})=\begin{pmatrix}u_I\\ 0\\ -p_I\\ -p_I u_I\\ 0\\ -p_I\\ -p_I u_I\end{pmatrix},\mathbf{H}(\mathbf{Q})=\begin{pmatrix}\alpha_1\\ 0\\ \alpha_1\\ \alpha_1\\ 0\\ \alpha_2\\ \alpha_2\end{pmatrix} \quad (9)$$

and

$$\mathbf{S}(\mathbf{Q})=\begin{pmatrix}\mu\left(p_1-p_2\right)\\ 0\\ \lambda\left(u_2-u_1\right)\\ \lambda\tilde{u}_I\left(u_2-u_1\right)-\mu\tilde{p}_I\left(p_1-p_2\right)\\ 0\\ -\lambda\left(u_2-u_1\right)\\ -\lambda\tilde{u}_I\left(u_2-u_1\right)+\mu\tilde{p}_I\left(p_1-p_2\right)\end{pmatrix} \quad (10)$$

In this system, the subscripts 1 and 2 refer to phase 1 and 2 respectively. $\alpha_k,\rho_k,u_k,p_k,E_k\ (k=1,2)$ represent the volume fraction, density, velocity, pressure and total energy of each phase. u_I,p_I represent the interfacial velocity and pressure modeled as in (Saurel et al., 2003):

$$u_I=\frac{Z_1u_1+Z_2u_2}{Z_1+Z_2}+\mathrm{sgn}\left(\frac{\partial\alpha_1}{\partial x}\right)\frac{p_2-p_1}{Z_1+Z_2},\ p_I=\frac{Z_2p_1+Z_1p_2}{Z_1+Z_2}+\mathrm{sgn}\left(\frac{\partial\alpha_1}{\partial x}\right)\frac{Z_1Z_2}{Z_1+Z_2}\left(u_2-u_1\right) \quad (11)$$

where $Z_k=\rho_k c_k$ is the acoustic impedance of phase k with c_k the associated sound speed. The homogeneous part of this model considers each phase as compressible, evolving with its own velocity, pressure and temperature. The source term vector $\mathbf{S}(\mathbf{Q})$ contains closure laws for pressure relaxation and drag effects, where μ and λ control the rate at which pressure and velocity equilibrium are reached respectively. For the present illustrations, there is no need to detail these relaxation parameters. Models with relevant physics may be found for example in (Saurel & Le Métayer, 2001) and (Saurel et al., 2017) in various contexts. The volume average interface pressure and velocity are given by:

$$\tilde{u}_I=\frac{Z_1u_1+Z_2u_2}{Z_1+Z_2},\ \tilde{p}_I=\frac{Z_2p_1+Z_1p_2}{Z_1+Z_2} \quad (12)$$

The system is closed by an EOS for each phase. The stiffened-gas EOS is used in the present work,

$$E_k=e_k\left(\rho_k,p_k\right)+\frac{1}{2}u_k^2=\frac{p_k+\gamma_k p_{SG,k}}{\rho_k\left(\gamma_k-1\right)}+\frac{1}{2}u_k^2, \quad (13)$$

The last closure relation corresponds to the saturation constraint,

$$\alpha_1 + \alpha_2 = 1. \qquad (14)$$

The compressible two-phase flow model (8)-(14) is a non-conservative hyperbolic system. The intercell flux functions arising from the spatial finite volume discretization are solved using a HLLC-type approximate Riemann solver detailed in (Furfaro & Saurel, 2015) and summarized in Appendix A.

Two 1D shock tube test problems are solved using explicit and implicit time integrators. The first test corresponds to a water-air shock tube without relaxation, and the second test corresponds to a shock tube with water-air mixture in the presence of instantaneous pressure relaxation and finite rate velocity relaxation.

Water-air shock tube without relaxation

Initially a 1m length tube is filled with liquid water on the left side of a membrane located at x=0.8m and pure air on the right side. Water pressure is set to 0.2GPa while air pressure is set to 0.1MPa. Densities of water and air are initially set to 1000kg/m^3 and 1 kg/m^3 respectively. Water is governed by the stiffened gas EOS with parameters $p_{SG,1} = 1\text{GPa}$ and $\gamma = 2.35$ while air is governed by the ideal gas law with $\gamma = 1.4$. A small volume fraction of air is present in the water and vice versa ($\alpha = 10^{-6}$). The mesh is composed of 2000 elements and first-order spatial discretization is used. The simulation time is set to 276µs.

This test case in the absence of pressure and velocity relaxation is very challenging. Indeed, interface conditions across the two-phase contact of equal pressure and normal velocity are not trivial to achieve as they are solely satisfied by the consistent discretization of the non-conservative terms present in the model. This problem is thus a hard benchmark for numerical schemes applied to the two-phase flow model out of equilibrium.

Explicit Euler temporal integration with CFL=0.5 and implicit methods with CFL=20 using Newton-BDF1, Newton-BDF2 and Newton-SSPSDIRK schemes have been compared. Single-step BDF methods failed on this test case for CFL numbers greater than 5. Newton method tolerance has been set to 10^{-6}, reached in less than 10 iterations thanks to the Jacobian accuracy yielding quadratic convergence.

Volume fraction of water is displayed in Figure 11. Explicit and implicit schemes solutions are merged and they all yield perfectly bounded solutions.

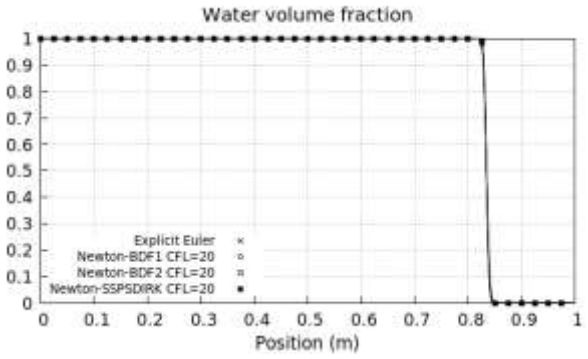

Figure 11: Water-air shock tube without relaxation. Volume fraction of water computed with both explicit and implicit schemes. Symbols are displayed every 50 elements.

Mixture pressure and velocity as well as pressures and velocities of both phases are plot in Figure 12. It can be seen on the top graphs that Newton-BDF1 is more diffusive than the explicit Euler solver but is strongly stable. Newton-BDF2 and Newton-SSPSDIRK which are second-order accurate give very similar solutions matching the explicit solution. A small oscillation is however present at the tail of the rarefaction wave using Newton-BDF2 scheme where Newton-SSPSDIRK remains oscillation-free. Second-order implicit methods are only conditionally SSP, SSPSDIRK with a less stringent time-step limit.

Bottom graphs of Figure 12 show both phases pressures and velocities. Only Newton-SSPSDIRK results are shown for the sake of conciseness, explicit solution can be found in (Furfaro & Saurel, 2015). Interface conditions are fulfilled: At the contact wave pressures and velocities of both phases match perfectly, as shown in the graph at bottom.

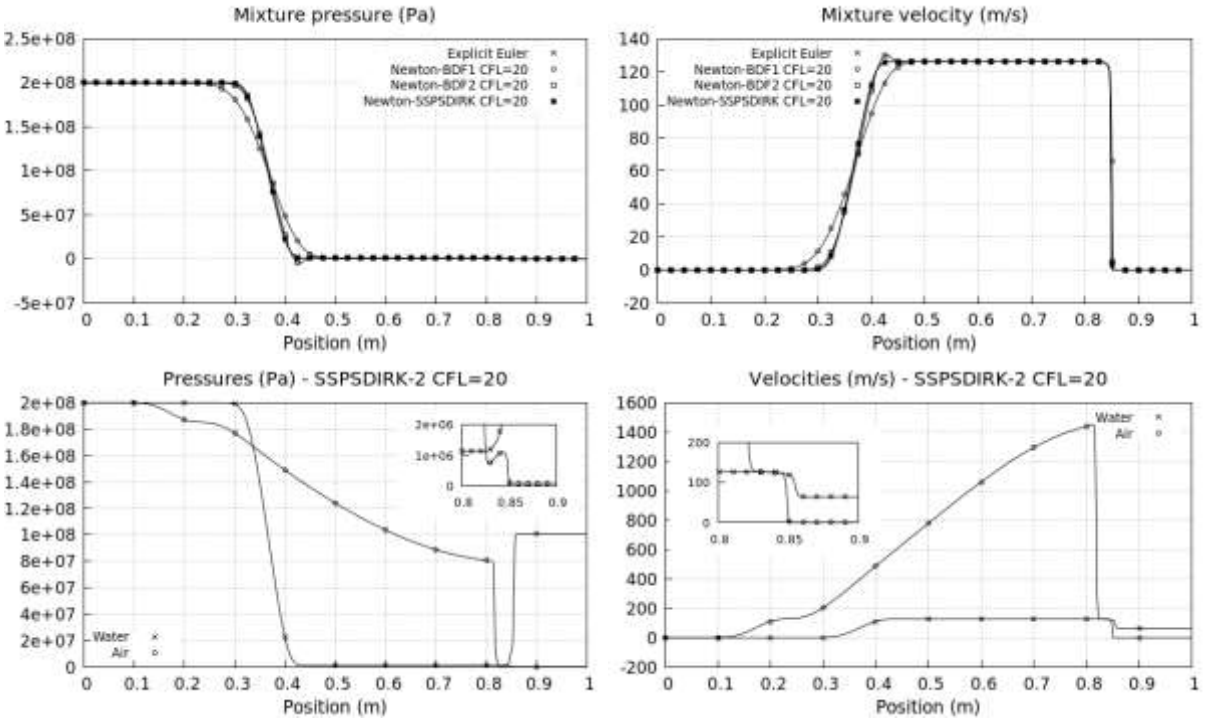

Figure 12: Water-air shock tube without relaxation. Comparison between explicit and implicit methods. Top: mixture pressure and mixture velocity. Bottom: pressures and velocities of both phases showing correct fulfillment of interface conditions.

Details regarding this non-trivial test problem may be found in (Abgrall & Saurel, 2003).

Water-air mixture shock tube with stiff pressure relaxation and finite drag relaxation

The second test case considered with the two-phase model out of equilibrium is a shock tube filled with a water-air mixture. A 1m length tube is initially filled with water and air in proportion $\alpha = 0.1$ and $\alpha = 0.9$ respectively. A membrane is placed at $x = 0.5\text{m}$ separating a high-pressure chamber at 100bar to the left and 1bar to the right. Densities of water and air are set to 1000kg/m^3 and 1kg/m^3 respectively. Water phase is governed by the stiffened gas EOS with parameters $p_{SG,1} = 1\text{GPa}$ and $\gamma = 2.35$ and air is governed by the ideal gas law with $\gamma = 1.4$. The mesh is composed of 2000 elements and first-order spatial discretization is used. The simulation time is set to 790µs.

Instantaneous pressure relaxation is used following a time splitting procedure detailed in (Furfaro & Saurel, 2015). Drag effects are accounted for considering constant water bubble radius $R_w = 5\text{mm}$. The exchange interfacial area per unit volume (A_I) dependence is thus a simplified one. It is assumed dependent only on water volume fraction (α_w) evolution which in turn is driven by pressure relaxation. The total friction coefficient λ is thus modelled as:

$$\lambda = \frac{Z_1 Z_2}{Z_1 + Z_2} A_I = \frac{Z_1 Z_2}{Z_1 + Z_2} \frac{3\alpha_w}{R_w},$$

This finite rate momentum exchange is integrated following a first-order explicit time-splitting method.

A comparison between explicit Euler temporal integration using CFL=0.5 and Newton-SSPSDIRK scheme using CFL=10,20 and 30 is carried out. Pressure is displayed in Figure 13, showing overall good agreement between explicit and implicit solutions. The dominant error is due to the time-splitting strategy. Indeed, the difference between the explicit solution and SSPSDIRK with CFL=10 is greater than the differences between the various implicit solutions using increasing CFL. This error may be lowered by inserting the finite drag relaxation inside the implicit temporal scheme.

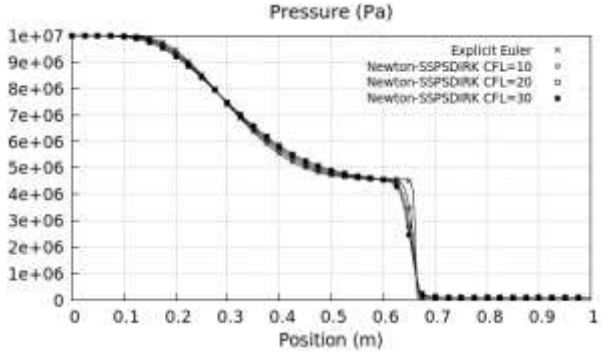

Figure 13: Water-air mixture shock tube with pressure relaxation and drag effects. Relaxed pressure using explicit and implicit time integrators.

Water and air velocities are displayed in Figure 14. Momentum exchanges driven by drag effects are quite stiff under these operating conditions as shown by the relatively small velocity drift. Overall good agreement between the explicit and the implicit solutions is visible.

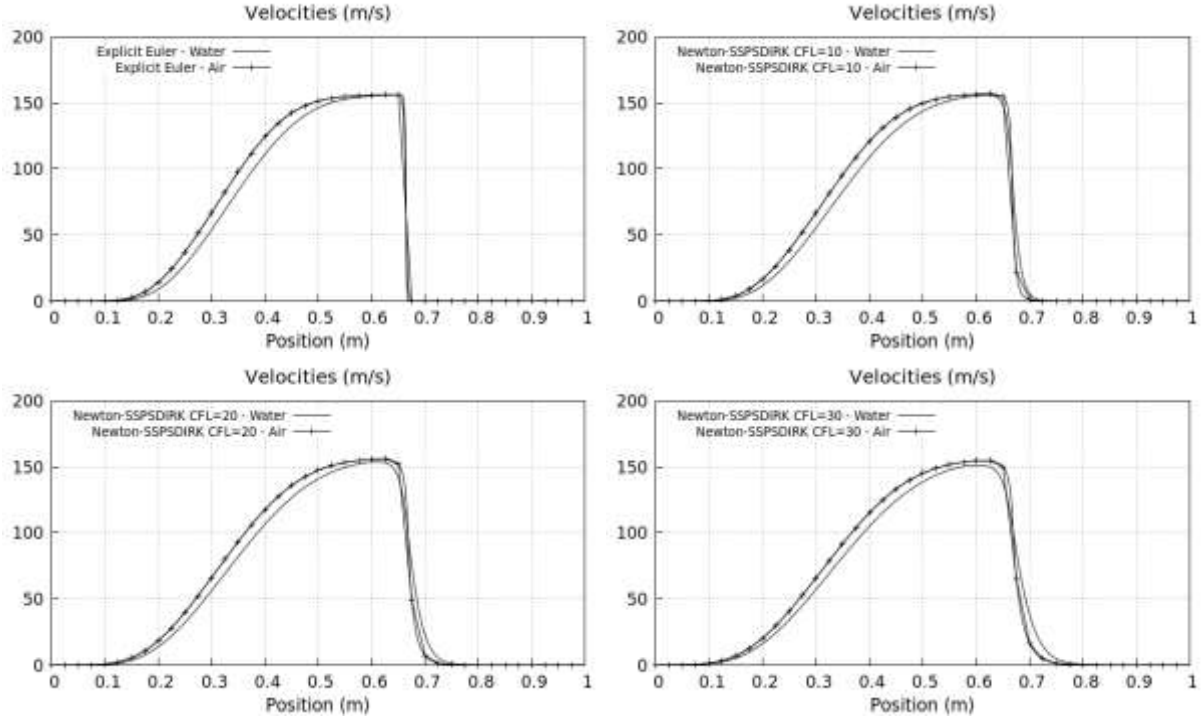

Figure 14: Water-air mixture shock tube with pressure relaxation and drag effects. Water and air velocities using explicit and implicit time integrators are in correct agreement.

b. Reduced model in full equilibrium

Another two-phase flow model is considered as well for numerical experiments. An asymptotic analysis in the stiff limit of velocity, pressure and temperature of system (8)-(14) yields the following two-phase model in full equilibrium (Le Martelot et al., 2014) (Chiapolino et al., 2016):

$$\frac{\partial \mathbf{Q}}{\partial t}+\frac{\partial \mathbf{F}(\mathbf{Q})}{\partial x}=0, \mathbf{Q}=\begin{pmatrix}\rho\\ \rho u\\ \rho E\\ \rho Y_1\end{pmatrix}, \mathbf{F}(\mathbf{Q})=\begin{pmatrix}\rho u\\ \rho u^2+p\\ u(\rho E+p)\\ \rho u Y_1\end{pmatrix} \qquad (15)$$

where ρ, u, p, E, Y_1 denote the mixture density, velocity, pressure, total energy and mass fraction of phase 1 respectively. The system is closed by the mixture mass definition,

$$Y_1+Y_2=1, \qquad (16)$$

and by a mixture EOS satisfying the following definitions for the mixture internal energy and mixture specific volume:

$$\begin{aligned} e &= Y_1 e_1(p,T)+Y_2 e_2(p,T)\\ v &= Y_1 v_1(p,T)+Y_2 v_2(p,T) \end{aligned} \qquad (17)$$

Considering stiffened-gas EOS for phase 1 and ideal gas EOS for phase 2, system (17) admits a unique physical solution for the mixture pressure:

$$\begin{aligned} p &= \frac{1}{2}\left(A_1+A_2-p_{SG,1}\right)+\sqrt{\frac{1}{4}\left(A_2-A_1+p_{SG,1}\right)^2+A_1A_2}\\ \text{with } A_1 &= \left(\rho e-p_{SG,1}\right)\frac{Y_1(\gamma_1-1)c_{v_1}}{Y_1c_{v_1}+Y_2c_{v_2}}, A_2=\rho e\frac{Y_2(\gamma_2-1)c_{v_2}}{Y_1c_{v_1}+Y_2c_{v_2}} \end{aligned} \qquad (18)$$

Equations (15),(16) and (18) form a conservative hyperbolic system. The sound speed associated to this system of equations is very well approximated by the simple Wood formula (Wood, 1930). The intercell flux function arising from the spatial finite volume discretization is solved using HLLC-type approximate Riemann solver (Saurel et al., 2006).

Two 1D problems are solved using explicit and implicit time integrators. The first test is a shock tube with a water-air mixture while the second configuration is a double expansion test. Multi-dimensional extension is addressed through a sonic jet problem and a gas-gas single mode Rayleigh-Taylor instability.

Water-air mixture shock tube

A 1m length tube is filled with a mixture of water and air. Water mass fraction is set to 2%. Initially, a membrane located at x=0.5m separates a high-pressure chamber to a low-pressure one. On the left side, pressure is set to 0.2MPa while it is set to 0.1MPa on the right side. Temperature is imposed to 293K in all the domain. Water phase is governed by the stiffened gas EOS with parameters $p_{SG,1}=1\text{GPa}$ and $\gamma=2.35$ and air is governed by the ideal gas law with $\gamma=1.4$. The mesh is composed of 10000 elements and first-order spatial discretization is used. The simulation time is set to 1ms.

Explicit and implicit integration is carried out. The explicit scheme uses first-order Euler integration with CFL number set to 0.5, while the implicit scheme uses single-step BDF1 with CFL=10,20 and 40. Mixture density is displayed at the final time in Figure 15. Under these mild initial conditions single-step BDF1 remains stable even for a CFL number of 40, yielding a computational gain of a factor 10 with respect to explicit integration.

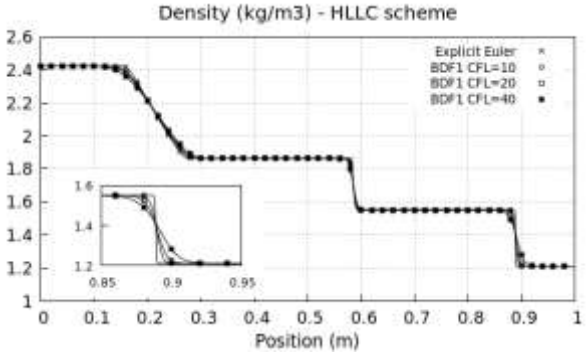

Figure 15: Water-air mixture shock tube. Mixture density at time t=1ms using explicit Euler and implicit BDF1 schemes.

Double expansion test

The second 1D test case considered using the two-phase model in mechanical and thermal equilibrium is a double expansion test problem. This test is particularly interesting to benchmark numerical schemes as vacuum conditions are reached at the center.

A 1m length tube is considered filled with almost pure water (air mass fraction equal to 10^{-6}) at atmospheric conditions p=0.1MPa and T=293K. Initially a membrane is located at x=0.5m. Velocity is set to -10m/s and 10m/s to the left and to the right of the membrane respectively. The domain is discretized with 10000 elements and the simulation time is set to 1.5ms. Explicit first-order Euler integration with CFL number equal to 0.5 and implicit single-step BDF1 scheme with CFL=10,20 and 40 are used with first-order spatial discretization.

Pressure and velocity at time t=1.5ms are displayed in Figure 16. Single-step BDF1 scheme remain stable in all cases without pressure positivity violation. At CFL 40, the gain in CPU time is about a factor 10 with respect to explicit Euler integration.

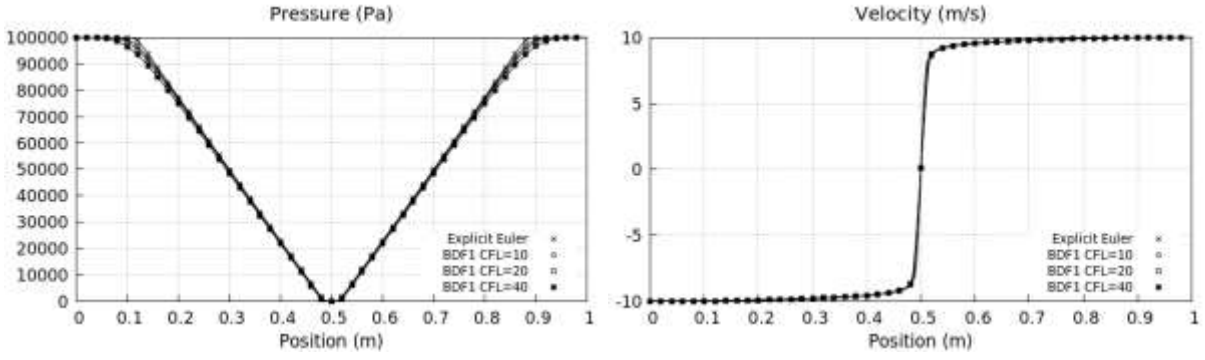

Figure 16: Double expansion test. Velocity (left) and pressure (right) using explicit Euler and implicit BDF1 schemes.

Multi-dimensional tests are now addressed. A 2D two-phase sonic jet simulation is carried out first, then a 2D two-phase single mode Rayleigh-Taylor instability is computed.

2D sonic jet

A 2D jet configuration is set up as described in Figure 17. Initially the domain is composed of a water-air mixture in equal mass proportion at atmospheric conditions. Water phase is governed by the stiffened gas EOS

with parameters $p_{SG,1} = 1\text{GPa}$ and $\gamma = 2.35$ and air is governed by the ideal gas law with $\gamma = 1.4$. At the left side, a two-phase pressure tank-inlet boundary condition is imposed (see Appendix B). Tank pressure is set to 0.2MPa yielding a chocked flow through the convergent-divergent nozzle and a sonic jet develops.

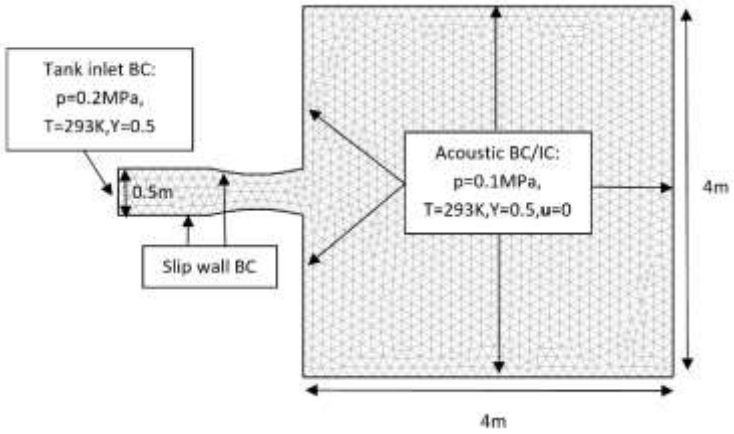

Figure 17: 2D sonic jet computational set-up.

The domain is discretized using an unstructured mesh composed of about 250000 triangles (the mesh edges length is multiplied by a factor 10 in Figure 17). Second-order spatial discretization is applied using Barth and Jespersen limiter and the simulation time is set to 21ms. Explicit temporal integration uses second-order SSPERGK scheme with CFL number equal to 0.5. Implicit schemes use single-step first-order BDF and second-order Newton-SSPSDIRK both using CFL equal to 20. In case of Newton-SSPSDIRK scheme, as quadratic convergence cannot be reached due to the first-order Jacobian simplification, a limit of five iterations to reach a non-linear residual magnitude of 10^{-6} has been imposed.

Two-phase tank-inlet condition implies the solution to a non-linear equation to approximate the boundary flux. The Jacobian boundary condition contribution is accounted for in a straightforward manner using ADOO as detailed in Appendix B.

Contours of density at time 14ms and 21ms for the three temporal schemes are displayed in Figure 18. No significant diffusion is observed on the jet contour using BDF1 scheme. In contrast the shock wave is slightly smeared. At this CFL number, the higher-order Newton-SSPSDIRK scheme predicts a sharper shock front but some non-amplifying oscillations are present.

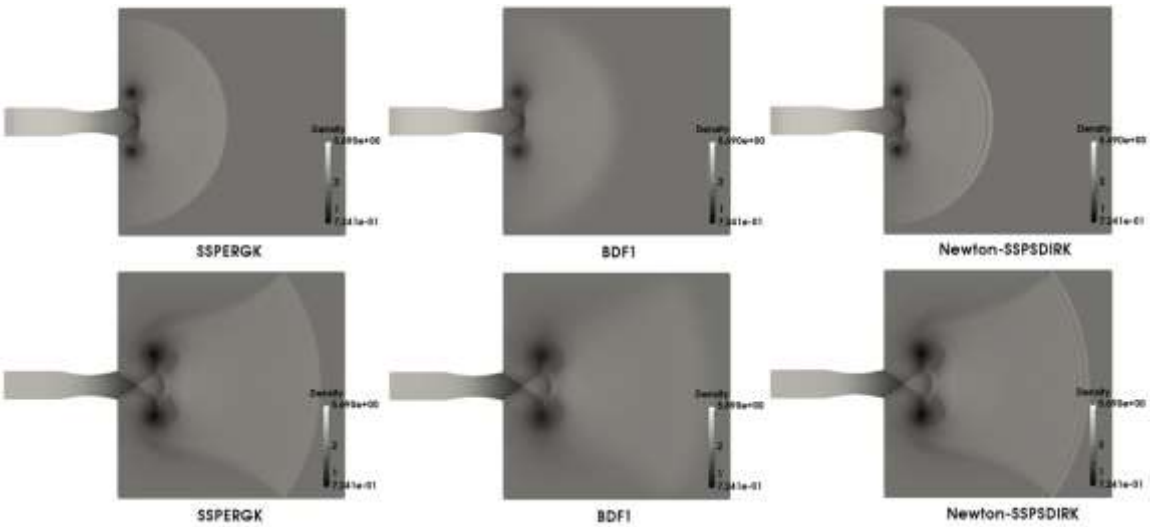

Figure 18: 2D jet at time t=14ms (top) and t=21ms (bottom). Mixture density contours.

Newton-SSPSDIRK scheme with imposed maximum of five non-linear iterations per time step requires about 10 linear systems to solve at each time step. At CFL number 20, the CPU gain with respect to two-step explicit SSPERGK scheme is about 17%. Single-step BDF scheme which only requires one linear system solution per time step yields CPU gain factor of about 5.

Air-Helium Rayleigh-Taylor instability

A 2D single-mode Rayleigh-Taylor configuration is set up as described in Figure 19. Initially the domain is composed of pure light gas (helium) at bottom and a denser gas on top (air). An initial perturbed interface separates the two gases to trigger a single-mode instability driven by gravity effects. Gravity magnitude has been set to 1000m/s^2. The computational domain is bounded by slip wall boundary conditions. Air and helium are governed by the ideal gas law with polytropic coefficients 1.4 and 1.67 respectively.

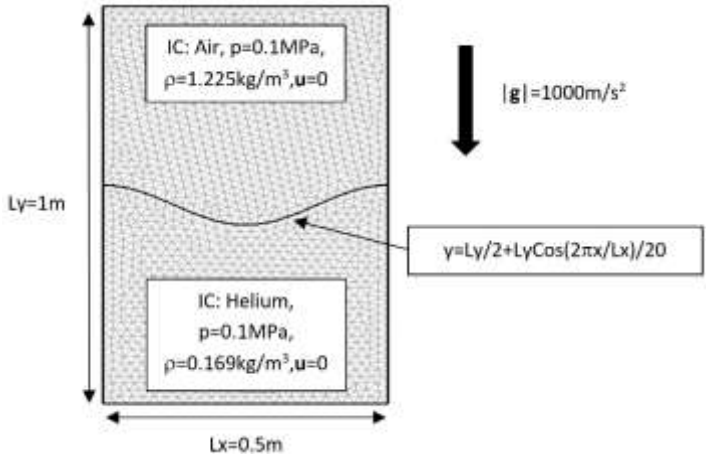

Figure 19: 2D single-mode Rayleigh-Taylor instability computational set up.

Final simulation time is set to 37.5ms. In the absence of viscous and surface tension effects, the interface is highly unstable and subject to chaotic behavior which can be triggered by small perturbations induced for example using distinct spatial numerical discretization (Liska & Wendroff, 2004). In the present context, the spatial discretization is fixed using second-order HLLC scheme on a mesh composed of about 300 000 triangles.

Explicit temporal integration uses second-order SSPERGK scheme with CFL number equal to 0.5. Implicit schemes use single-step first-order BDF and second-order Newton-SSPSDIRK both using CFL equal to 100. In case of Newton-SSPSDIRK scheme, a limit of five iterations to reach a non-linear residual magnitude of 10^{-6} has been imposed. Contours of mixture density are shown for the three temporal schemes at two time-intervals in Figure 20.

Single-step first-order BDF scheme is more diffusive than the explicit scheme resulting in a slightly altered mushroom shape. In contrast, second-order implicit SSPSDIRK scheme gives a remarkably similar solution compared to the explicit one.

The CPU speed-up using implicit SSPSDIRK scheme is about a factor 4 while it reaches a factor of about 27 using single-step first-order BDF method.

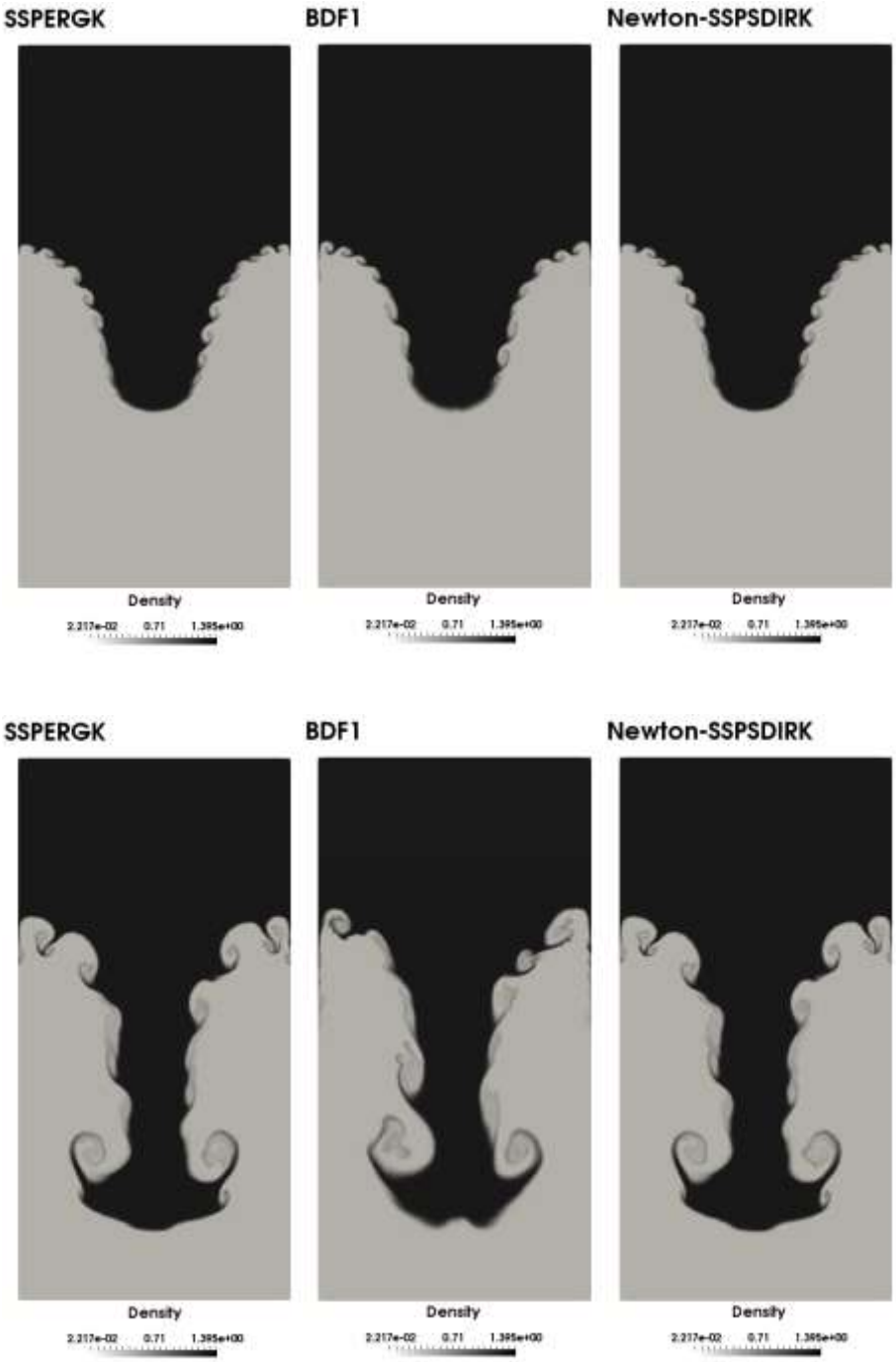

Figure 20: Single mode Rayleigh-Taylor instability at time t=22.5ms (top) and t=37.5ms (bottom). Mixture density contours using explicit and implicit time integrators.

V. Conclusion

A numerical procedure to evaluate the matrix Jacobian coming from the implicit discretization of hyperbolic systems has been presented. The approach is based on the forward mode of ADOO and showed several advantages:

- It is easily applicable to existing codes with languages compilers supporting user-defined DDT and operator overloading;

- It is highly flexible with flow models and associated numerical fluxes as the user do not need to worry about the implementation of the derivatives;
- It handles complex numerical algorithms such as root-finding methods or conditional branches differentiation in a straightforward manner;
- The overhead associated to operations on DDTs is negligible in the present context as most of the computational time is spent in the linear solver.

The ADOO method has been applied to the implicit discretization of various flow models, involving root-finding methods and complex equations of state. In the test-cases needing unsteady residual resolution, the accurate Jacobian evaluation allowed very fast convergence of Newton's method (always quadratic for first-order in space discretization).

ADOO allowed implicit integration of the compressible two-phase flow model of Baer and Nunziato. In the author's knowledge, the present work corresponds to the first successful attempt with a fully compressible and implicit approach.

Bibliography

Abgrall, R. & Saurel, R., 2003. Discrete equations for physical and numerical compressible multiphase mixtures. *Journal of Computational Physics,* pp. 361-396.

Baer, M. R. & Nunziato, J. W., 1986. A two-phase mixture theory for the deflagration-to-detonation transition (ddt) in reactive granular materials. *International Journal of Multiphase Flow,* pp. 861-889.

Balay , S. et al., 2018. *PETSc Users Manual,* s.l.: Argonne National Laboratory.

Barth, T. & Jespersen, D., 1989. *The design and application of upwind schemes on unstructured meshes.* s.l., AIAA.

Barth, T. J. & Linton, S. W., 1995. An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation. *AIAA Paper 95-0221.*

Bilanceri, M. et al., 2012. Linearized implicit time advancing and defect correction applied to sediment transport simulations. *Computers & Fluids,* Volume 63, pp. 82-104.

Bischof, C., Khademi, P., Mauer, A. & Carle, A., 1996. Adifor 2.0: Automatic Differentiation of Fortran 77 Programs. *IEEE Computational Science & Engineering,* pp. 18-32.

Bramkamp, F. D., Bücker, H. M. & Rasch, A., 2006. Using exact Jacobians in an implicit Newton–Krylov method. *Computers and Fluids,* Volume 35, p. 1063–1073.

Briley, W. R. & McDonald, H., 1977. Solution of the multi-dimensional compressible Navier-Stokes equations by a generalized implicit method. *Journal of Computational Physics,* pp. 372-397.

Brown, P. N. & Saad, Y., 1990. Hybrid Krylov Methods for Nonlinear Systems of Equations. *SIAM Journal on Scientific and Statistical Computing,* pp. 450-481.

Chiapolino, A., Boivin, P. & Saurel, R., 2016. A simple phase transition relaxation solver for liquid–vapor flows. *International Journal for Numerical Methods in Fluids,* 83(7), pp. 583-605.

Colonia, S., Steijl, R. & Barakos, G. N., 2014. Implicit implementation of the AUSMP and AUSMP-up schemes. *International Journal for Numerical Methods in Fluids,* p. 687–712.

Davis, S. F., 1988. Simplified second-order Godunov-type methods. *SIAM Journal of Scientific and Statistical Computing,* 9(3), p. 445–473.

Dennis, J. E. & Schnabel, R. B., 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Englewood Cliffs: Prentice-Hall.

Dubuc , L. et al., 1998. Solution of the Unsteady Euler Equations Using an Implicit Dual Time Method. *AIAA Journal, 36,* pp. 1417-1424.

Espath, L. F. R., Linn, R. V. & Awruch, A. M., 2011. Shape optimization of shell structures based on NURBS. *International Journal for Numerical Methods in Engineering,* Volume 88, pp. 613-636.

Furfaro, D. & Saurel, R., 2015. A simple HLLC-type Riemann solver for compressible non-equilibrium two-phase flows. *Computers & Fluids,* pp. 159-178.

Godunov, S., 1959. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematicheskii Sbornik, Steklov Mathematical Institute of Russian Academy of Sciences,* Volume 47, pp. 271-306.

Gottlieb, S., Shu, C.-W. & Tadmor, E., 2001. Strong Stability-Preserving High-Order Time Discretization Methods. *SIAM Review,* pp. 89-112.

Griewank, A. & Walther, A., 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* s.l.:SIAM.

Harten, A., Lax, P. D. & van Leer, B., 1983. On upstreaming differencing and Godunov-type. *SIAM Review,* 25(1), pp. 35-61.

Hascoet, L. & Pascual, V., 2013. The Tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software.*

Hovland, P. D. & McInnes, L. C., 2001. Parallel simulation of compressible flow using automatic differentiation and PETSc. *Parallel Computing,* Volume 27, pp. 503-519.

Jameson, A., 1991. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. *AIAA Paper 91-1596.*

Jameson, A. & Turkel, E., 1981. Implicit Scheme and LU-Decompositions. *Mathematics of Computation,* pp. 385-397.

Karypis, G. & Kumar, V., 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing,* 20(1), pp. 359-392.

Kennedy, C. A. & Carpenter, M. H., 2016. *Diagonally Implicit Runge-Kutta Methods for Ordinary Differential Equations. A Review,* NASA Langley Research Center; Hampton, VA, United States: NASA/TM-2016-219173, L-20470, L-20597, NF1676L-19716.

Le Martelot, S., Saurel, R. & Nkonga, B., 2014. Towards the direct numerical simulation of nucleate boiling flows. *International Journal of Multiphase Flow,* pp. 62-78.

Liou, M. S., 1996. A Sequel to AUSM: AUSMP. *Journal of Computational Physics,* 129(2), pp. 364-382.

Liska, R. & Wendroff, B., 2004. Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. *SIAM Journal on Scientific Computing,* pp. 995-1017.

Liu, X., Xia, Y., Luo, H. & Xuan, L., 2016. A Comparative Study of Rosenbrock-Type and Implicit Runge-Kutta Time Integration for Discontinuous Galerkin Method for Unsteady 3D Compressible Navier-Stokes equations. *Communications in Computational Physics,* pp. 1016-1044.

Martin, M. P. & Candler, G. V., 2006. A parallel implicit method for the direct numerical simulation of wall-bounded compressible turbulence. *Journal of Computational Physics,* pp. 153-171.

Mohammadi, B. & Pironneau, O., 2004. Shape optimization in fluid mechanics. *Annual Review of Fluid Mechanics,* Volume 36, pp. 255-279.

Mulder, W. A. & Van Leer, B., 1983. Implicit upwind methods for the Euler equations. *AIAA Paper 83-1930.*

Newman, J. C. et al., 1999. Overview of Sensitivity Analysis and Shape Optimization for Complex Aerodynamic Configurations. *Journal of Aircraft,* 36(1), pp. 87-96.

Pandolfi, M. & Larocca, F., 1989. Transonic flow about a circular cylinder. *Computers & Fluids,* 17(1*), pp. 205-220.

Park, J. S., Yoon, S.-H. & Kim, C., 2010. Multi-dimensional limiting process for hyperbolic conservation laws on unstructured grids. *Journal of Computational Physics,* pp. 788-812.

Pulliam, T. H., 1993. Time Accuracy and the Use of Implicit Methods. *AIAA Paper 93-3360.*

Reynolds, R., Samtaney, R. & Tiedeman, H. C., 2012. A fully implicit Newton–Krylov–Schwarz method for tokamak magnetohydrodynamics: Jacobian construction and preconditioner formulation. *Computational Science and Discovery,* 5(1).

Rinaldi, E., Pecnik, R. & Colonna, P., 2014. Exact Jacobians for implicit Navier–Stokes simulations of equilibrium real gas flows. *Journal of Computational Physics,* p. 459–477.

Rusanov, V. V., 1962. Calculation of interaction of non-steady shock waves with obstacles. *USSR Computational Mathematics and Mathematical Physics,* 1(2), pp. 304-320.

Saad, Y. & Schultz, M. H., 1986. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing,* pp. 856-869.

Salas, M. D., 1983. Recent developments in transonic Euler flow over a circular cylinder. *Mathematics and Computers in Simulation,* 25(3), pp. 232-236.

Saurel , R., Gavrilyuk, S. & Renaud, F., 2003. A multiphase model with internal degrees of freedom: application to shock–bubble interaction. *Journal of Fluid Mechanics,* pp. 283-321.

Saurel, R., Boivin, P. & Le Métayer, O., 2006. A general formulation for cavitating, boiling and evaporating flows. *Computers & Fluids,* Volume 128, pp. 53-64.

Saurel, R., Fraysse, F., Furfaro, D. & Lapébie, E., 2017. Multiscale multiphase modeling of detonations in condensed energetic materials. *Computers & Fluids,* pp. 95-111.

Saurel, R. & Le Métayer, O., 2001. A multiphase model for compressible flows with interfaces, shocks, detonation waves and cavitation. *Journal of Fluid Mechanics,* pp. 293-271.

Saurel, R., Petitpas, F. & Berry, R. A., 2009. Simple and efficient relaxation methods for interfaces separating compressible fluids, cavitating flows and shocks in multiphase mixtures. *Journal of Computational Physics,* pp. 1678-1712.

Sod, A. S., 1978. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics,* 27(1), pp. 1-31.

Squire, W. & Trapp, G., 1998. Using Complex Variables to Estimate Derivatives of Real Functions. *SIAM Review,* 40(1), pp. 110-112.

Toro, E. F., Spruce, M. & Speares, W., 1994. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves,* 4(1), p. 25–34.

Venkatakrishnan, V. & Barth, T., 1989. Application of direct solvers to unstructured meshes for the Euler and Navier-Stokes equations using upwind schemes. *AIAA Paper 89-0364.*

Walther, A. & Griewank, A., 2012. Getting started with ADOL-C. *In U. Naumann und O. Schenk, Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science,* pp. 181-202.

Wengert, R. E., 1964. A simple automatic derivative evaluation program. *Communications of the ACM,* pp. 463-464.

Wood, A., 1930. *A textbook of sound.* London: G. Bell and Sons Ltd.

Xia, Y., Luo, H., Frisbey, M. & Nourgaliev, R., 2014. A set of parallel, implicit methods for a reconstructed discontinuous. *Computers and Fluids,* Volume 98, pp. 134-151.

Appendix A: HLLC solver for the two-phase disequilibrium model

The two-phase model for disequilibrium mixtures expresses under the following compact form for a generic phase k:

$$\frac{\partial \mathbf{Q}_k}{\partial t}+\frac{\partial \mathbf{F}_k(\mathbf{Q}_k)}{\partial x}+\alpha_k\frac{\partial \mathbf{H}_k(\mathbf{Q}_k)}{\partial x}=0,\mathbf{Q}_k=\begin{pmatrix}\alpha_k\\(\alpha\rho)_k\\(\alpha\rho u)_k\\(\alpha\rho E)_k\end{pmatrix},\mathbf{F}_k(\mathbf{Q}_k)=\begin{pmatrix}\alpha_k u_I\\(\alpha\rho u)_k\\\alpha_k(\rho u^2+p)_k-\alpha_k p_I\\\alpha_k(\rho E+p)_k u_k-\alpha_k p_I u_I\end{pmatrix},\mathbf{H}_k(\mathbf{Q}_k)=\begin{pmatrix}-u_I\\0\\p_I\\p_I u_I\end{pmatrix} \quad (19)$$

The first step consists in approximating the interfacial terms u_I et p_I. This step takes its roots in the Discrete Equations Method (Abgrall & Saurel, 2003);(Saurel et al., 2003), given a 2D topology composed of independant fluid channels as illustrated in Figure 21.

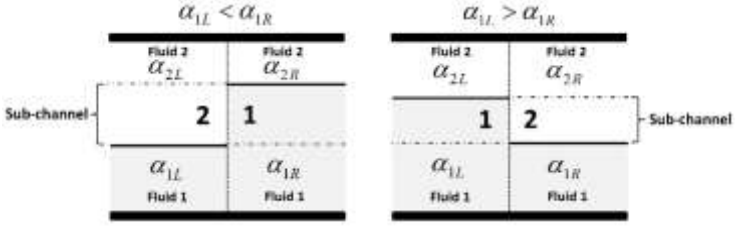

Figure 21: 2D topology showing two types of contacts at a given element boundary.

On a given cell boundary, three types of contacts are admissible : two mono-phasic contacts 1-1 and 2-2 and one contact 2-1 if the phase 1 volume fraction is higher on the right side or 1-2 if it is lower. Interfacial pressure and velocity are approximated as the solution of the Riemann problem associated to the two-fluid Euler

equations. As the Riemann solution depends only on the left and right states, u_I and p_I become local constants during a time-step.

Focusing on a 2-1 contact, the wave pattern is shown in Figure 22.

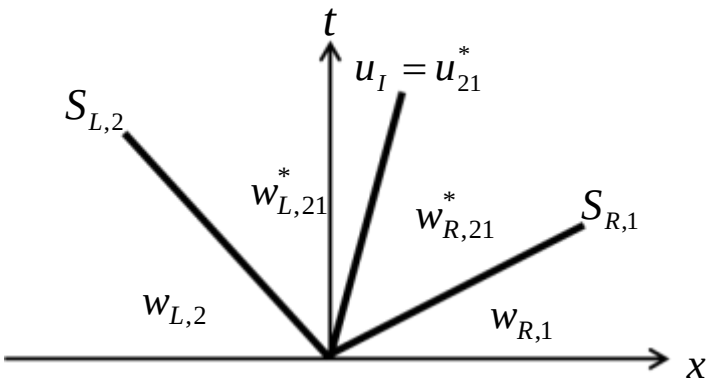

Figure 22: Riemann wave pattern associated to contact 2-1.

Wave speeds $S_{L,2}, S_{R,1}$ are computed following Davis approximation (Davis, 1988):

$$S_{R,1} = \mathrm{Max}\left(u_{L,1} + c_{L,1}, u_{R,1} + c_{R,1}\right)$$
$$S_{L,2} = \mathrm{Min}\left(u_{L,2} - c_{L,2}, u_{R,2} - c_{R,2}\right)$$

The interfacial variables u_I and p_I are computed HLL (Harten et al., 1983) and HLLC (Toro et al., 1994) approximations respectively:

$$u_I = u_{21}^{*,HLL} = \frac{\left(\rho u^2 + p\right)_{R,1} - \left(\rho u^2 + p\right)_{L,2} + S_{L,2}\left(\rho u\right)_{L,2} - S_{R,1}\left(\rho u\right)_{R,1}}{\left(\rho u\right)_{R,1} - \left(\rho u\right)_{L,2} + S_{L,2}\rho_{L,2} - S_{R,1}\rho_{R,1}}$$
$$p_I = p^*_{R,21} = \rho_{R,1}\left(u_{R,1} - S_{R,1}\right)\left(u_{R,1} - u_I\right) + p_{R,1}$$

The symetric situations, contact 1-2 is considered similarly (Furfaro & Saurel, 2015).

The interfacial vairbales being locally constant on an element boundary during a time-step, System (19) is rewritten under the following conservative form:

$$\frac{\partial \mathbf{Q}_k}{\partial t} + \frac{\partial \mathbf{F}_k\left(\mathbf{Q}_k\right)}{\partial x} = 0, \mathbf{Q}_k = \begin{pmatrix} \alpha_k \\ \left(\alpha\rho\right)_k \\ \left(\alpha\rho u\right)_k \\ \left(\alpha\rho E\right)_k \end{pmatrix}, \mathbf{F}_k\left(\mathbf{Q}_k\right) = \begin{pmatrix} \alpha_k u_I \\ \left(\alpha\rho u\right)_k \\ \alpha_k\left(\rho u^2 + P\right)_k - \alpha_k p_I \\ \alpha_k\left(\rho E + p\right)_k u_k - \alpha_k p_I u_I \end{pmatrix}$$

Local constancy of the interfacial variables also implies the decoupling of the full 7-wave Riemann problem in two 4-wave Riemann problems as shown in .

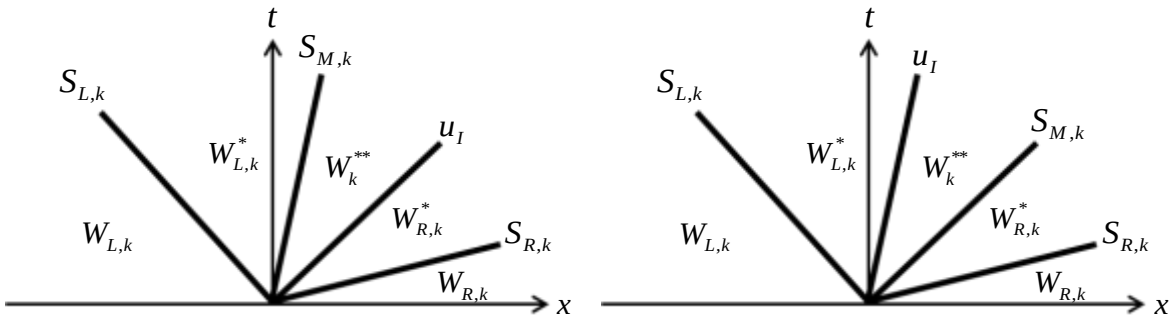

Figure 23: Main patterns of phase k Riemann problem: $S_{L,k} < S_{M,k} < u_I < S_{R,k}$ et $S_{L,k} < u_I < S_{M,k} < S_{R,k}$

The 4-wave Riemann solution is based on the Rankine-Hugoniot relations, approximating the wave speeds $S_{L,k}$ and $S_{R,k}$ by Davis (1988) estimates and intermediate contact speed $S_{M,k}$ by HLL. Rankine-Hugoniot relations across the various waves read :

$$\mathbf{F}^*_{L,k} = \mathbf{F}_{L,k} + S_{L,k}\left(\mathbf{Q}^*_{L,k} - \mathbf{Q}_{L,k}\right)$$
$$\mathbf{F}^*_{R,k} = \mathbf{F}_{R,k} + S_{R,k}\left(\mathbf{Q}^*_{R,k} - \mathbf{Q}_{R,k}\right)$$
$$\mathbf{F}^{**}_{k} = \mathbf{F}^*_{R,k} + u_I\left(\mathbf{Q}^{**}_{k} - \mathbf{Q}^*_{R,k}\right)$$
$$\mathbf{F}^*_{L,k} = \mathbf{F}^{**}_{k} + S_{M,k}\left(\mathbf{Q}^*_{L,k} - \mathbf{Q}^{**}_{k}\right)$$

Explicit relations for $\mathbf{Q}^*_{L,k}, \mathbf{Q}^*_{R,k}$ and $\mathbf{Q}^{**}_{k}$ are obtained as a functions of known states $\mathbf{Q}_{L,k}$ and $\mathbf{Q}_{R,k}$. Riemann solution state is obtained through a classical sampling. Once computed, these states and fluxes are used in the non-conservative explicit or implicit Godunov scheme associated to system (19).

The whole numerical flux function is algorithmically rather complex. Nevertheless, the ADOO procedure provides the exact Jacobian matrix in a straightfoward manner.

Appendix B: Tank-inlet boundary condition

The tank-inlet boundary condition is obtained though the solution of a semi-Riemann problem described in Figure 24.

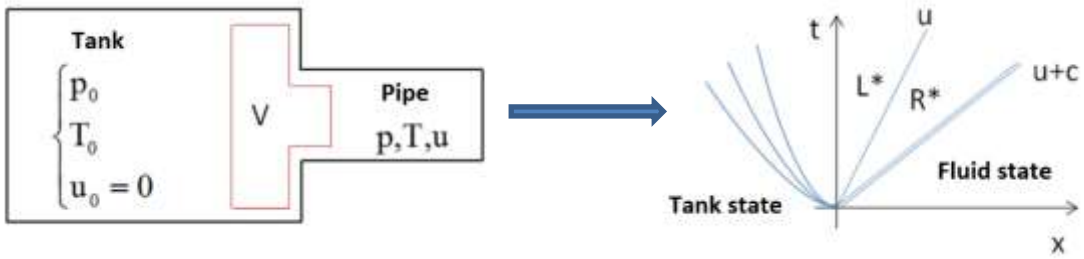

Figure 24: Tank-inlet boundary condition description.

Assuming a stationary flow between the tank and the inlet, the integration of the energy equation associated to the Euler equations on volume V yields:

$$\int_V \vec{\nabla}.\left(\rho H\vec{u}\right)dV=0 \quad \Leftrightarrow \quad \left(\rho uHS\right)_L^* - \left(\rho uHS\right)_0 = 0 \overset{\text{mass cons°}}{\Rightarrow} \quad H_L^* = H_0$$

which gives a first relation linking the tank state to the left Riemann state. Stagnation enthalpy is conserved across the left curved wave.

Similar calculation with the help of the entropy equation yields $s_L^* = s_0$. This relation is approximated through the sound speed definition $c^2 = \left(\frac{\partial p}{\partial \rho}\right)_s$ which is assumed constant between the tank and the inlet: $c_L^{*2} = c_0^2 \approx \frac{p_L^* - p_0}{\rho_L^* - \rho_0}$. This gives a second relation linking the tank state to the left Riemann state: $\rho_L^* = \rho_0 + \frac{p_L^* - p_0}{c_0^2}$. Acoustic relations are assumed through the right wave:

$$\begin{aligned} & p_R^* - \rho_R c_R u_R^* = p_R - \rho_R c_R u_R \\ & \Rightarrow p_R^*\left(u_R^*\right) = p_R + \rho_R c_R\left(u_R^* - u_R\right) \end{aligned}$$

The middle wave being necessarily a contact, equality of pressure and normal velocity is imposed yielding $p_L^* = p_R^*, u_L^* = u_R^*$. Putting all relations together yields to:

$$\left.\begin{aligned} & e\left(p_L^*, \rho_L^*\right) + \frac{p_L^*}{\rho_L^*} + \frac{1}{2}u_L^{*2} - H_0 = 0 \\ & \rho_L^* = f\left(p_L^*\right) = \rho_0 + \frac{p_L^* - p_0}{c_0^2} \\ & u_R^* = g\left(p_R^*\right) = \frac{p_R^* - p_R}{\rho_R c_R} + u_R \\ & p_L^* = p_R^* \\ & u_L^* = u_R^* \end{aligned}\right\} \Rightarrow e\left(p^*, f\left(p^*\right)\right) + \frac{p^*}{f\left(p^*\right)} + \frac{1}{2}g\left(p^*\right)^2 - H_0 = h\left(p^*\right) = 0$$

After application of the EOS, this non-linear relation is solved numerically using Newton's method. Classical sampling is then performed, and the flux is computed from the Riemann solution. ADOO is propagated to the Newton's method in a transparent way, however extra care must be taken to obtain an accurate flux derivative as detailed in the following.

Denoting $\mathbf{Q}_0$ the vector of known values in the tank and $\bar{\mathbf{Q}}_R$ the vector of conservative variables coming from the computational domain, $\mathbf{Q}^*$, solution to the semi-Riemann problem is obtained as the root of a function $\mathbf{h}(\mathbf{Q}^*, \bar{\mathbf{Q}}_R, \mathbf{Q}^0)$. The local contribution to the Jacobian matrix $\frac{\partial \mathbf{F}\left(\mathbf{Q}^*\right)}{\partial \bar{\mathbf{Q}}_R}$ s.t. $\mathbf{h}(\mathbf{Q}^*, \bar{\mathbf{Q}}_R, \mathbf{Q}^0) = 0$ needs to be evaluated. The analytic expression of this contribution is:

$$\frac{\partial \mathbf{F}\left(\mathbf{Q}^*\right)}{\partial \bar{\mathbf{Q}}_{\mathrm{R}}} = -\frac{\partial \mathbf{F}\left(\mathbf{Q}^*\right)}{\partial \mathbf{Q}^*}\left[\frac{\partial \mathbf{h}(\mathbf{Q}^*, \bar{\mathbf{Q}}_{\mathrm{R}}, \mathbf{Q}^0)}{\partial \mathbf{Q}^*}\right]^{-1} \frac{\partial \mathbf{h}(\mathbf{Q}^*, \bar{\mathbf{Q}}_{\mathrm{R}}, \mathbf{Q}^0)}{\partial \bar{\mathbf{Q}}_{\mathrm{R}}}.$$

The accuracy of the flux derivative, if computed analytically, depends only on the level of approximation of $\mathbf{Q}^*$. On the other hand, when ADOO method is applied the accuracy of the flux derivative evaluation not only depends on the convergence rate of $\mathbf{Q}^*$ but also on the convergence rate of $\frac{\partial \mathbf{h}(\mathbf{Q}^*, \bar{\mathbf{Q}}_{\mathrm{R}}, \mathbf{Q}^0)}{\partial \mathbf{Q}^*}$. To better understand this subtle point, let us take a scalar example.

Be $\mathrm{f}\left(\mathrm{y}(\mathrm{x})\right)$ s.t. $\mathrm{g}(\mathrm{y},\mathrm{x}) = 0$ a function whose derivative with respect to x is to be evaluated. Let be $\mathrm{g}(\mathrm{y},\mathrm{x}) = \mathrm{e}^{\mathrm{y}} - \mathrm{x}$ and $\mathrm{f}\left(\mathrm{y}(\mathrm{x})\right) = \mathrm{e}^{\mathrm{y}(\mathrm{x})}$. The analytic derivative of f with respect to x is obtained easily as $\frac{\partial \mathrm{f}\left(\mathrm{y}(\mathrm{x})\right)}{\partial \mathrm{x}} = 1$ and actually neither depends on y nor on x. However, ADOO computes $\frac{\partial \mathrm{f}\left(\mathrm{y}(\mathrm{x})\right)}{\partial \mathrm{x}}$ as the result of successive derivative updates in the Newton iterative scheme. Indeed, under ADOO framework, the $\mathrm{n}+1$ Newton iteration reads:

$$\left[\mathrm{y}^{\mathrm{n}+1}, \frac{\partial \mathrm{y}^{\mathrm{n}+1}}{\partial \mathrm{x}}\right] = \left[\mathrm{y}^{\mathrm{n}}, \frac{\partial \mathrm{y}^{\mathrm{n}}}{\partial \mathrm{x}}\right] - \left[\frac{\mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\frac{\partial \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\partial \mathrm{y}}}, \frac{\frac{\partial \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\partial \mathrm{x}}\frac{\partial \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\partial \mathrm{y}} - \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)\frac{\partial^2 \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\partial \mathrm{y}\partial \mathrm{x}}}{\left(\frac{\partial \mathrm{g}\left(\mathrm{y}^{\mathrm{n}},\mathrm{x}\right)}{\partial \mathrm{y}}\right)^2}\right]$$

Thus, the accuracy of $\frac{\partial \mathrm{f}\left(\mathrm{y}(\mathrm{x})\right)}{\partial \mathrm{x}}$ computed by ADOO depends on the convergence rates of both $\mathrm{y}^{\mathrm{n}+1}$ and $\frac{\partial \mathrm{y}^{\mathrm{n}+1}}{\partial \mathrm{x}}$. The graph in Figure 25 depicts this situation. The ADOO evaluation of $\frac{\partial \mathrm{f}\left(\mathrm{y}(\mathrm{x})\right)}{\partial \mathrm{x}}$ is compared to the exact value and displayed as a function of different convergence criteria: the magnitude of function $\mathrm{g}(\mathrm{y},\mathrm{x})$, the magnitude of Newton solution increment $\Delta \mathrm{y}$ and the magnitude of the derivative of function $\mathrm{g}(\mathrm{y},\mathrm{x})$. While the two formers show a very similar trend, the latter has slightly different convergence rate.

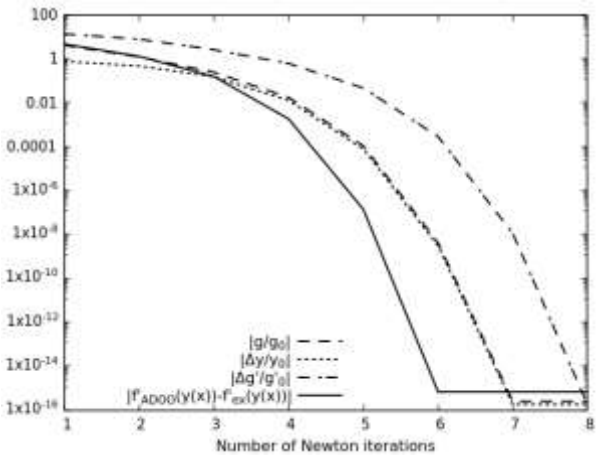

Figure 25: Convergence rate of the derivative of a non-explicit function using ADOO.

Stop criterion of the Newton algorithm should then be based on both function and function derivative.

Extension to the two-phase reduced model is straightforward considering the mixture EOS. A prototype subroutine is given as example in Figure 26.

```
PROGRAM TankInlet(p0,rho0,c0,Y0,H0,VcR,fluxbc)
REAL(KIND=RP),INTENT(IN)       :: p0,rho0,c0,Y0,H0
TYPE(AutoDiffType),INTENT(IN)  :: VcR(:)
TYPE(AutoDiffType),INTENT(OUT) :: fluxbc(:)
INTEGER                        :: var,nit
TYPE(AutoDiffType)             :: rhor,ur,Etotr,Yr,er,pr,cr
TYPE(AutoDiffType)             :: pstar,rhostarl,rhostarr,rhostar,ustar,Ystar,estar,fstar,dfstar

! Derivatives initialization
DO var=1,Ncons
  VcR(var)%dv(:)   = 0.0_RP
  VcR(var)%dv(var) = 1.0_RP
END DO

rhor  = VcR(VC_Rho)
ur    = VcR(VC_RhoU)/rhor
Etotr = VcR(VC_RhoE)/rhor
Yr    = VcR(VC_RhoY)/rhor
er    = Etotr-0.5_RP*ur**2
pr    = prey(rhor,er,Yr)
cr    = cpry(pr,rhor,Yr)

! Newton method for the solution of h(pstar)=0.
pstar=p0
DO nit=1,nitmax
  rhostarl = rho0+(pstar-p0)/c0**2
  ustar    = (pstar-pr)/(rhor*cr)+ur
  fstar    = FuncHP(H0,rhostarl,ustar,pstar)
  IF(ABS(fstar)<eps)EXIT
  dfstar   = dFuncHPdp(H0,rhostarl,ustar,pstar)
  pstar    = pstar-fstar/dfstar
END DO

IF(nit==nitmax+1) STOP "Newton convergence failed in TankInlet."

! Sampling
IF(ustar>0.0_RP)THEN
  Ystar    = Y0
  rhostar  = rhostarl
ELSE
  Ystar    = Yr
  rhostarr = rhor+(pstar-pr)/cr**2
  rhostar  = rhostarr
ENDIF

! Boundary flux. fluxbc(:)%dv contains derivative blocks w.r.t. VcR.
fluxbc(VC_Rho)  = rhostar*ustar
fluxbc(VC_RhoY) = rhostar*ustar*Ystar
fluxbc(VC_RhoU) = rhostar*ustar**2+pstar
fluxbc(VC_RhoE) = ustar*(rhostar*(epry(pstar,rhostar,Ystar)+0.5_RP*ustar**2)+pstar)

END SUBROUTINE TankInlet
```

Figure 26: Fortran routine for the computation of tank-inlet boundary flux. The resulting data structure fluxbc contains the derivative blocks of the flux with respect to the vector of conservative variables needed in the Jacobian matrix.