SYMCOMP 2023
Évora, 30-31 March 2023

A GENERAL METHODOLOGY FOR SYMBOLICALLY GENERATING MANUFACTURED SOLUTIONS SATISFYING PRESCRIBED CONDITIONS - APPLICATION TO TWO-PHASE FLOWS EQUATIONS

David Henneaux[1,2*], Pierre Schrooyen[3], Philippe Chatelain[2], Thierry Magin[1,4]

1: von Karman Institute for Fluid Dynamics,
Waterloosesteenweg 72, Sint-Genesius-Rode, B-1640, Belgium
e-mail: david.henneaux@vki.ac.be

2: Université catholique de Louvain,
Place du Levant 2, Louvain-la-Neuve, B-1348, Belgium

3: Cenaero,
Rue des Frères Wright 29, Gosselies, B-6041, Belgium

4: Université Libre de Bruxelles,
Avenue F.D. Roosevelt 50, Brussels, B-1050, Belgium
e-mail:
{pierre.schrooyen@cenaero.be,philippe.chatelain@uclouvain.be,thierry.magin@vki.ac.be}

Keywords: Manufactured solution, symbolic algebraic system, code verification, open-source Python software, computational fluid dynamics

Abstract. *The Method of Manufactured Solution (MMS) is a powerful technique for code verification. It provides a systematic procedure for generating analytical solutions to be discretized by a numerical solver. Usually, an arbitrary continuous solution is selected before being inserted into the governing equations. Although sufficient in many situations, this procedure may be inappropriate if the problem at hand imposes some constraints on the shape of the manufactured solutions. In such cases, some unknown parameters should be included in the continuous solution and computed to satisfy the imposed constraints. This limitation of the standard MMS has already been recognized in previous work. However, the way to handle it is most of the time case-dependent and based on ad-hoc strategies. In this work, we develop a generic framework based on the Sympy library to produce manufactured solutions complying with arbitrary complex Dirichlet and Neumann-type conditions. It is made available through an open-source Python software. As a challenging illustrative application, analytical solutions obeying the interface jumps conditions of a two-phase compressible Navier-Stokes system are built and are exploited to conduct a convergence study of a high-order Computational Fluid Dynamics (CFD) solver.*

1 INTRODUCTION

Scientific computing plays an ever-growing role in the prediction of natural and engineering systems. In response to the demands of simulations encompassing increasing levels of fidelity, the complexity of the models, the numerical methods and the algorithms in computational tools has risen significantly. Along with this software sophistication, it is becoming more and more challenging, yet essential, to design strategies assessing the implementation and the suitability of the physical models in order to demonstrate the predictive capabilities of the numerical tool. These assessment strategies fall into two complementary categories: verification and validation. Validation evaluates the accuracy of the physical models implemented in the code for a given application. It is typically performed through comparison with experimental data and answers the question "are we solving the right equations?". On the other hand, verification is a purely mathematical enterprise, which attempts to resolve the question "are we solving the equations right?". It is commonly further divided into code verification and solution verification. Solution verification estimates the numerical errors incurred by the discretization of the continuous model while code verification has for purpose to test that the numerical algorithms in the code are implemented correctly and that they are functioning as intended.
The present work focuses on code verification, which is the first mandatory action to be conducted in the verification and validation (V&V) process. The books [1, 2, 3] provide a comprehensive discussion about V&V. We propose to extend the capabilities of the most widely used code verification approach, namely the Method of Manufactured Solutions (MMS). This powerful and elegant method was first proposed in [4] and has since then been applied to many computational physics codes; a selection of them can be found in [5, 6]. The MMS is a general procedure to construct arbitrarily complex exact analytical solution to the governing equations. This reference solution can then be exploited to perform convergence and order of accuracy tests. Formal order verification not only allows to detect coding mistakes, it can also identify glitches in the implemented formulations and algorithms. Most of the time, the manufactured solution (MS) lacks physical realism and this is not a concern since verification is strictly a mathematical exercise. An artificial source term is added to the original interior equations such that the MS is the exact solution of the transformed system inside the domain. In [7], a boundary residual is considered in addition to the interior residual so that it is possible to add the MMS source term to both the interior and boundary equations. This approach necessitates a discretization method that employs boundary residual, which limits its generality. That is why in general, the auxiliary conditions like initial and boundary conditions (BC) are often considered *after* the solution to the interior equations has been manufactured. And it is precisely this flexibility that makes the MMS so attractive. However, in some situations, it might be required to incorporate *during* the solution design particular constraints dictated by the governing equations at hand. One important scenario has to do with the verification of the imposition of specialized BC in the software, as pointed out in [8]. In this case, it

is convenient to build the solution such that it complies with the physical/mathematical conditions enforced on a portion of the domain boundaries. Several previous work have addressed this aspect. In [8, 9], the baseline MS is multiplied by a function involving the equation of the concerned boundary geometry so that the solution or its gradient evaluated on the boundary is equal to the value corresponding to the exact physical BC to be tested. The same approach has been applied and extended in [10] to different BCs commonly used in compressible and incompressible CFD simulations on general curved boundaries, with a focus on identifying mathematically well-posed boundary conditions formulations. Although flexible thanks to the possibility of considering complex boundary geometries, this strategy is mostly applicable to classical wall BC. When it comes to the prescription of conditions on boundaries with more complex models or across discontinuities like shocks or material interfaces, a more suitable methodology seems desirable. The idea is to define the MS with some unknown free parameters that are determined based on the set of conditions. Some examples are given here. In [11], a MS consistent with the employed turbulence models, both in the interior of the domain and on the boundaries, is proposed. A physically-realistic MS is derived in [12, 6] for moving walls with ablating boundary conditions. In [13, 14], the parameters of the MS are computed such as to respect the interface jumps conditions associated to scalar two-phase equations. Finally, a least-squares regression to generate MS obeying the Rankine-Hugoniot jumps relations on curved shocks for the Euler equations is discussed in [15]. Those examples illustrate the case-by-case handling of MS assembly for complex models.

The aim of this work is to establish a generic numerical framework to build MS satisfying arbitrary complex conditions. It relies on the *Sympy Python* library for symbolic manipulations. It has been designed as a versatile tool that can accommodate any new governing equations and conditions. As such, it is proposed to the community as an open-source software and can be considered as an easy-to-use tool that complements the already available ones, like the *MASA* library [16]. The free parameters of the user-defined MS are computed with the available *Sympy* system of equations solver while the MMS source terms are automatically generated using symbolic differentiation. We name this new tool *PyManufSol* and it is available at `https://github.com/henneauxd/PyManufSol` [17].

This paper is structured as follows. The theoretical foundations are first given in Sec. 2, highlighting the various steps in the whole code verification process and discussing the limitations of the current proposed approach. In connection with this description, the structure and working principle of the numerical tool are detailed in Sec. 3. Its capabilities are then illustrated in Sec. 4 on two challenging test cases. Section 5 concludes the work and provides some perspectives.

2 BUILDING MANUFACTURED SOLUTIONS SATISFYING PRESCRIBED CONDITIONS

2.1 General manufacturing and verification procedure

We start by presenting the whole manufacturing and verification procedure. Suppose the solver we want to verify handles an initial-boundary value model problem of the form: find $\mathbf{u}\colon \Omega \to \mathcal{U}$, where Ω is the physical domain expressed in terms of the spatial coordinates vector $\mathbf{x}$ and the scalar temporal coordinate t , such that

$$\begin{cases} \mathbf{F}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x},t),\ \forall t > 0 \ , \\ \mathbf{B}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \partial\Omega(\mathbf{x},t),\ \forall t > 0 \ , \\ \mathbf{I}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x},t),\ t = 0 \ , \end{cases} \tag{1}$$

where $\mathbf{F}\colon \mathcal{U} \to \mathcal{V}$ is a differential operator mapping elements of vector space $\mathcal{U}$ to the vector space $\mathcal{V}$ and representing the implemented equations in the interior domain Ω, $\mathbf{B}\colon \mathcal{U} \to \mathcal{W}$ is an operator gathering all the conditions on the domain boundaries $\partial\Omega$, and $\mathbf{I}\colon \mathcal{U} \to \mathcal{V}$ is the operator for the initial conditions. Let us remark that in the most general case, the domain can be time-dependent.

Manufacturing The solution and source term manufacturing steps include:

1. the selection of an arbitrary solution expression, $\widetilde{\mathbf{u}}$, that may depend, in addition to the spatial and temporal coordinates, on a set of N_p free unknown parameters, noted $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_{N_p}\}$:

$$\widetilde{\mathbf{u}} = \widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda}) \ . \tag{2}$$

   The choice of the expression does not need to bear any resemblance to the one of the exact physical solution $\mathbf{u}$ of the system Eq. (1) since this latter is most of the time unknown and impossible to derive analytically. Both the spatial and temporal coordinates, $\mathbf{x}$ and t, as well as the free parameters are considered as symbolic variables in the computational framework.

2. the definition of the interior governing equations implemented in the software to be verified, i.e. the operator $\mathbf{F}$ in Eq. (1). The derivatives that appears in the equations are also handled in a symbolic way.

3. the definition of a set of N_c constraints/conditions that the manufactured solution $\widetilde{\mathbf{u}}$ will need to comply with and which is noted as

$$\mathbf{C}(\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda})) = \begin{Bmatrix} C_1(\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda})) = 0 \\ \vdots \\ C_{N_c}(\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda})) = 0 \end{Bmatrix} \ . \tag{3}$$

Those constraints can involve the manufactured solution itself and its derivatives and can be freely defined by the user. However, in most of the cases, the constraints will correspond to a subset or the entire set of the boundary conditions $\mathbf{B}$.

4. the computation of the unknown parameters $\boldsymbol{\lambda}$ from the set of constraints Eq. (3), i.e.

$$\text{find} \quad \boldsymbol{\lambda}^*(\mathbf{x}, t) \quad \text{s.t.} \quad \mathbf{C}(\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda}^*)) = \mathbf{0} \ . \tag{4}$$

   If the constraints have been defined taking into account some of the original problem boundary conditions, this step will reconcile $\widetilde{\mathbf{u}}$ with some constraints imposed by the physical model. In the most general case, the system Eq. (3) is a non-linear system of equations that is solved symbolically with appropriate functions provided by the *Sympy* library. Some important remarks about the construction and the solution of this system are given in Sec. 2.2.

5. the optional change of spatial coordinates system. To simplify the resolution of Eq. (3), it is sometimes advisable to define $\widetilde{\mathbf{u}}$ as being dependent of a single spatial coordinates while the constraints $\mathbf{C}$ should be defined on boundaries aligned with one of the coordinates (this is discussed in Sec. 2.2). In order to make the resulting solution more complex such that it checks all the terms involved in $\mathbf{F}$, the original spatial coordinates $\mathbf{x}$ can be translated to a new reference system to become $\overline{\mathbf{x}}$:

$$\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda}^*) \Rightarrow \widetilde{\mathbf{u}}(\overline{\mathbf{x}}, t, \boldsymbol{\lambda}^*(\overline{\mathbf{x}}, t)) \ . \tag{5}$$

   For the constraints to still hold after this transformation, the boundaries on which they have been defined should also be transformed in the new coordinates system for the verification step.

6. the assembly of the MMS source term. Because $\widetilde{\mathbf{u}}$ will in general not be the exact solution of the interior equations, inserting the generated manufactured solution into the previously defined operator $\mathbf{F}$ gives rise to an artifical residual:

$$\mathbf{F}(\widetilde{\mathbf{u}}(\overline{\mathbf{x}}, t, \boldsymbol{\lambda}^*)) = \mathbf{S}(\overline{\mathbf{x}}, t) \ , \tag{6}$$

   where $\mathbf{S}$ is referred to as the MMS source term. It is automatically generated by passing the manufactured solution which is now fully defined (the λ^* parameters have been computed before) to the the specified operator $\mathbf{F}$ thanks to symbolic manipulations.

Code verification After having generated the MS and the MMS source term in the proposed symbolic software, the verification of the code can be undertaken via the following steps:

1. if not already available, the implementation into the code of a source term discretization. Indeed, the original problem we want to solve is modified according to Eq. (6). In other words, we seek the solution of the interior equations determined by a new operator $\widetilde{\mathbf{F}}$:

$$\widetilde{\mathbf{F}}(\widetilde{\mathbf{u}}) = \mathbf{F}(\widetilde{\mathbf{u}}) - \mathbf{S} = \mathbf{0} \ . \tag{7}$$

   It is worth mentioning that some alternative non-intrusive approaches that do not require the addition of a source term to the governing equations have been developed [18, 6]. They require however the modification of the physical parameters involved in the equations.

2. the splitting of the boundary conditions between the ones to be verified and the "artificial" ones. It must be recalled that the main interest of the present methodology is to verify, in addition to the interior equations, the implementation of some specialized boundary conditions used in the code. It is therefore necessary to let those boundary conditions as such to be able to compute the errors made by the code during their imposition based on the exact reference value that is known if the boundary condition considered is part of the constraints Eq. (3). On the other hand, the other remaining boundary conditions can simply be specified as Dirichlet condition with the imposed value being equal to the value of the MS evaluated on the boundary. The modified boundary operator can then be written as

$$\begin{aligned} \widetilde{B}_i(\widetilde{\mathbf{u}}) &= B_i(\widetilde{\mathbf{u}}) \quad \text{if} \quad B_i \in \mathbf{C} \ , \\ \widetilde{B}_i(\widetilde{\mathbf{u}}) &= \widetilde{\mathbf{u}} - \widetilde{\mathbf{u}}(\mathbf{x}_{\mathrm{BC}}, t) \quad \text{if} \quad B_i \notin \mathbf{C} \ , \end{aligned} \tag{8}$$

   where $\mathbf{x}_{\mathrm{BC}}$ identifies the evaluation of the MS the corresponding boundary location.

3. the specification of the initial conditions depending on the problem type. In general, the initial solution will simply be set as the MS evaluated at $t = 0$,

$$\widetilde{\mathbf{I}}(\widetilde{\mathbf{u}}) = \widetilde{\mathbf{u}} - \widetilde{\mathbf{u}}(\mathbf{x}, t = 0) \ . \tag{9}$$

   For time-independent problem solved by an iterative steady-state procedure, it may be useful to slightly perturb the initial condition given by Eq. (9) in order to inspect the convergence behavior of the solver.

4. the discretization of the continuous problem. The numerical discretization of the modified continuous problem composed by the modified operators $\widetilde{\mathbf{F}}$, $\widetilde{\mathbf{B}}$ and $\widetilde{\mathbf{I}}$ described above writes: find $\widetilde{\mathbf{u}}^h \colon \Omega^h \to \mathcal{U}^h$ such that

$$\begin{cases} \mathbf{F}(\widetilde{\mathbf{u}}^h(\mathbf{x}, t)) = \mathbf{S}(\mathbf{x}, t) \ , & \forall \mathbf{x} \in \Omega^h(\mathbf{x}, t), \, \forall t > 0 \ , \\ \widetilde{\mathbf{B}}(\widetilde{\mathbf{u}}^h(\mathbf{x}, t)) = 0 \ , & \forall \mathbf{x} \in \partial\Omega^h(\mathbf{x}, t), \, \forall t > 0 \ , \\ \widetilde{\mathbf{I}}(\widetilde{\mathbf{u}}^h(\mathbf{x}, t)) = 0 \ , & \forall \mathbf{x} \in \Omega^h(\mathbf{x}, t), \, t = 0 \ , \end{cases} \tag{10}$$

   where Ω^h represents the geometrical discretization and $\mathcal{U}^h$ is the discrete space in which the approximate solution $\widetilde{\mathbf{u}}^h$ is searched.

5. the order of accuracy test. Among the different criteria that can be used to perform code verification (code-to-code comparisons, discretization error quantification, convergence/consistency tests), the order of accuracy test is the most rigorous one [19]. It evaluates if the discretization errors are reduced at the expected theoretical rate when refining the geometry discretization and/or the time step size. Any implementation or conceptual mistakes should prevent that rate to be achieved and indicate the presence of an error. If *a priori* bounds of the form

$$\|\widetilde{\mathbf{u}} - \widetilde{\mathbf{u}}^h\| \leq C\, h^p \ , \tag{11}$$

are available, for a constant C, the theoretical order of convergence p and the mesh spacing parameter or time step h, a systematic convergence study can be conducted for various levels of uniform h-refinement by evaluating the error in the appropriate norm between the exact and approximate MS.

2.2 Limitations concerning the constraints definition and the MS free parameters computation

This subsection provides more details about the steps 3 and 4 of the manufacturing process described above that are specific to the current approach compared to the usual MMS strategy. Although theoretically applicable to a wide range of constraints and solutions, the approach currently implemented in our software has some limitations regarding the steps 3 and 4:

- to be able to solve exactly the system Eq. (3), the number of free parameters introduced in the MS must be equal to the number of constraints specified on the MS,

$$N_p = N_c \ . \tag{12}$$

- if Neumann or Robin conditions are included in the set of constraints, we need to ensure that the free parameters appearing in those constraints will not depend on the spatial coordinates $\mathbf{x}$. Indeed, when solving Eq. (3) at step 4, we implicitly assume that all spatial derivatives of the free parameters are zero, i.e.

$$\nabla_{\mathbf{x}} \boldsymbol{\lambda}^* = \mathbf{0} \ . \tag{13}$$

  Without that assumption, the non-linear algebraic system would be transformed to a non-linear differential algebraic system of equations (DAE), which appears to be very complicated to solve with complex MS expressions. A good practice to ensure that condition is to (i) choose the expression of the MS to be dependent on a single spatial coordinate, (ii) impose the constraints on boundaries which are oriented parallel to the other spatial coordinate direction. Doing so, it will be possible to substitute the constant numerical value defining the boundary equation into the MS gradient expression before solving the system, hence removing all spatial coordinates dependencies.

- related to the previous remark, it is at the moment not possible to consider arbitrary boundary geometries. However, as explained in the manufacturing step 5, there is the possibility to operate a change of coordinates after having determined the MS, which allows to rotate and translate a boundary, or even to transform it to a curve by going e.g. from a polar to Cartesian reference frame.

3 DESCRIPTION OF THE MANUFACTURING SOLUTION SOFTWARE

An overview of the structure of the *PyManufSol* tool is given in Fig. 1 and the links between its constituent modules and the manufacturing steps described in Sec.2.1 are established below:

- *ManufSolution*: the user starts by defining the spatial coordinates $\mathbf{x}$, the temporal coordinate t and the set of free parameters $\boldsymbol{\lambda}$ as symbolic variables (using the `symbols` function of *Sympy*). The user then chooses expressions depending on those symbolic variables for each of the field f of the solution vector. The tool has been designed with the will to treat not only single phase situations but also two-phase configurations. That is why it is possible to assemble a solution vector for each phase k considered in the system.

- *PhysicalModels*: the MS is then fed to the second module which handles the physical models, i.e. the interior governing equations represented by the operator $\mathbf{F}$. Each phase will have its own model. The type of reference coordinates system considered is also specified. It must be noted that this module can easily be extended with new models by overriding the functions defining the convective and diffusive fluxes.

- *BoundaryConditions*: the third module might not be necessary if the assembled MS does not contain any unknown parameters. In this case, the fourth module can directly be used. If there are some free parameters to determine, a set of constraints/ boundary conditions need to be compiled by the user. First, the geometry of each boundary on which a constraint is enforced is stipulated. The user has then the choice between two main boundary conditions types, i.e. Dirichlet and Neumann. The first kind only involves the solution vector and other quantities that can be obtained from it (e.g. the density from the pressure and temperature). The second kind of BC requires quantities depending on the spatial gradients of the solution. The physical model $\mathbf{F}$ is used at this stage to express the quantities required in the constraint definition. Let us remark that a constraint can be a condition concerning the MS of a specific phase, $\widetilde{\mathbf{u}}_k$, or a condition connecting the MS in both phases, which is useful for interface jumps conditions (as it will be illustrated in Sec. 4) but which could also be applied to the case of shock discontinuities e.g.

- *ProblemSolving*: the fourth module has a central role as it takes as inputs the physical model and the set of constraints. From there, it can be used to straightforwardly

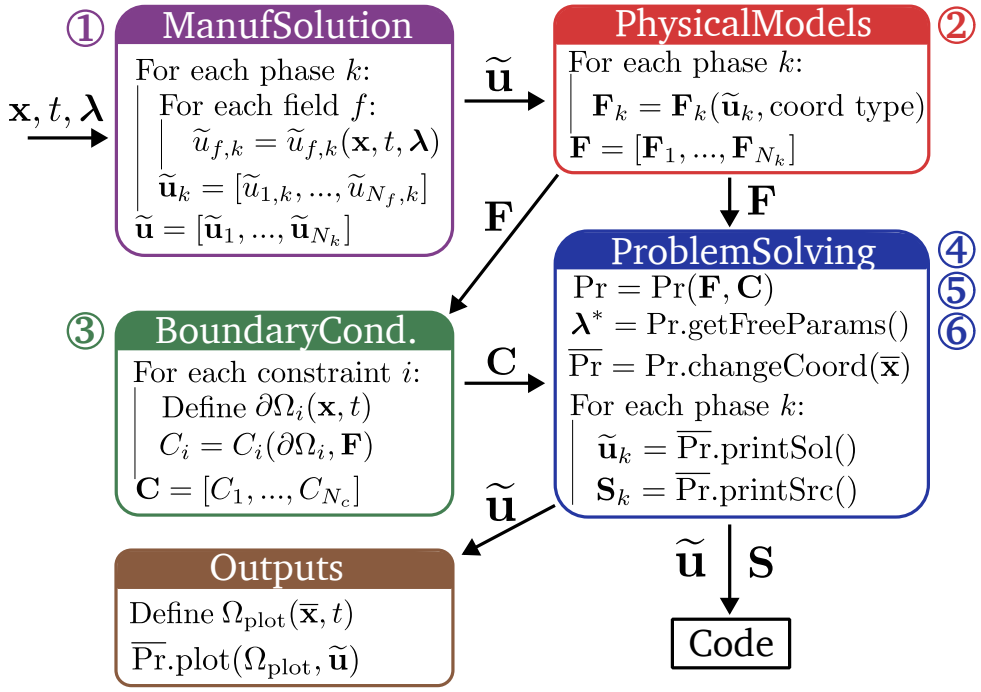


Figure 1: Modules constituting the *pyManufSol* numerical framework with their main functionalities, the way they are related and their link with the manufacturing steps described in Sec.2.1 (via the circled numbers).

compute the free parameters (by solving Eq. (3) calling the `solvers.solve` *Sympy* method) and possibly change the initial coordinates system. Functions to print in readable format files the final expressions of the MS and the source terms in all phases are also available. Those files can then be passed to the code to be verified which only needs to be able to extract the expressions contained in them and replace the (x, y, t) variables by the appropriate numerical values.

- *Outputs*: there is a last module dealing with the visual representation of the MS. One-dimensional plots along user-specified directions or two-dimensional plots over a predefined zone are possible. This can be useful to evaluate the suitability of the MS before using them in the verification process.

4 APPLICATION EXAMPLES

4.1 Governing equations

To illustrate the capabilities of the software, we consider two steady test cases about the complex situation of a two-phase flow system governed by the compressible Navier-Stokes equations. Each phase k occupying a portion of the domain Ω_k obeys the two-dimensional

system of equations given by

$$\mathbf{F}_k(\mathbf{u}_k(\mathbf{x})) = \nabla \cdot \left(\underline{\boldsymbol{F}}_k^c(\mathbf{u}_k) \; - \; \underline{\boldsymbol{F}}_k^d(\mathbf{u}_k, \, \nabla \mathbf{u}_k) \right) = \mathbf{0} \quad \forall \, \mathbf{x} \in \Omega_k \; , \tag{14}$$

such that the two phases occupy the entire domain and are separated by an interface Γ,

$$\Omega \coloneqq \Omega_1 \cup \Omega_2 \cup \Gamma \; , \quad \text{with } \Gamma \coloneqq \overline{\Omega}_1 \cap \overline{\Omega}_2 \; . \tag{15}$$

The conservative variables, the convective fluxes and the diffusive fluxes for phase k are given respectively by

$$\begin{aligned} \mathbf{u}_k = \begin{bmatrix} \rho_k \\ \rho_k \mathbf{v}_k \\ \rho_k E_k \end{bmatrix} \; , \quad \underline{\boldsymbol{F}}_k^c(\mathbf{u}_k) = \begin{bmatrix} \rho_k \mathbf{v}_k \\ \rho_k \mathbf{v}_k \otimes \mathbf{v}_k + p_k \underline{\boldsymbol{I}} \\ \rho_k \mathbf{v}_k H_k \end{bmatrix} \; , \\ \underline{\boldsymbol{F}}_k^d(\mathbf{u}_k, \, \nabla \mathbf{u}_k) = \begin{bmatrix} 0 \\ \underline{\boldsymbol{\tau}}_k \\ \underline{\boldsymbol{\tau}}_k \cdot \mathbf{v}_k - \mathbf{q}_k \end{bmatrix} \; , \end{aligned} \tag{16}$$

in which the density of phase k is noted ρ_k, the velocity vector in (x, y) Cartesian coordinates system $\mathbf{v}_k = [v_{x,k}, v_{y,k}]$, the static pressure p_k and the identify matrix $\underline{\boldsymbol{I}}$. The total energy E_k and the total enthalpy H_k per unit mass both include thermal and kinetic contributions

$$E_k = e_k + \frac{\mathbf{v}_k \cdot \mathbf{v}_k}{2} \; , \quad H_k = E_k + \frac{p_k}{\rho_k} \; , \tag{17}$$

with the specific internal energy e_k. The constitutive equations for the shear stress tensor $\underline{\boldsymbol{\tau}}_k$,

$$\underline{\boldsymbol{\tau}}_k = \mu_k \left[\nabla \mathbf{v}_k + (\nabla \mathbf{v}_k)^T - \frac{2}{3} \nabla \cdot \mathbf{v}_k \, \underline{\boldsymbol{I}} \right] = \begin{bmatrix} \tau_{xx,k} & \tau_{xy,k} \\ \tau_{yx,k} & \tau_{yy,k} \end{bmatrix} \; , \tag{18}$$

and the heat flux $\mathbf{q}_k$,

$$\mathbf{q}_k = -\lambda_k \, \nabla T_k = \begin{bmatrix} q_{x,k} \\ q_{y,k} \end{bmatrix} \; , \tag{19}$$

follow the assumption of a Newtonian fluid and the Fourier law, respectively, with the dynamic viscosity μ_k, the thermal conductivity λ_k, and the temperature T_k. To close the system of Eqs. (14), a stiffened gas model is used for the volumetric and caloric equations of state that unequivocally determine the state of the fluid inside each phase k:

$$\begin{aligned} p_k(\rho_k, T_k) &= \rho_k (\gamma_k - 1) e_k - \gamma_k p_{\infty,k} \, , \\ e_k(\rho_k, T_k) &= C_{v,k} T_k + \frac{p_{\infty,k}}{\rho_k} \, , \end{aligned} \tag{20}$$

where γ_k is the heat capacity ratio, $C_{v,k}$ is the heat capacity at constant volume and $p_{\infty,k}$ models molecular attractive effects.

The bulk equations of each phase are coupled by jumps conditions specifying the exchanges of mass, momentum, and energy through the interface. Their projection along the interface normal $\mathbf{n}_\Gamma$ and tangent $\mathbf{t}_\Gamma$ writes:

$$\begin{array}{lllll} & & \llbracket \dot{m}_\Gamma \rrbracket & = & 0 \\ \dot{m}_\Gamma \llbracket v_n \rrbracket + \llbracket p \rrbracket - \Delta p & & & = & \llbracket \tau_{nn} \rrbracket \\ \dot{m}_\Gamma \llbracket v_t \rrbracket & & - \Delta \tau_{nt} & = & \llbracket \tau_{nt} \rrbracket \\ \dot{m}_\Gamma \llbracket E \rrbracket + \llbracket p\, v_n \rrbracket - \Delta p\, v_{n,\Gamma} & & - \Delta \tau_{nt}\, v_{t,\Gamma} - \Delta q & = & \llbracket \tau_{nn}\, v_n \rrbracket + \llbracket \tau_{nt}\, v_t \rrbracket \\ & & & - & \llbracket q_n \rrbracket \end{array} \tag{21}$$

with the jump operator at a position $\mathbf{x} \in \Gamma$ for some quantity η defined in the bulk of each phase given by

$$\llbracket \eta \rrbracket = (\eta_{1|\Gamma} - \eta_{2|\Gamma}) \ , \quad \text{with} \quad \eta_{k|\Gamma} = \lim_{\epsilon \to 0, \epsilon > 0} \eta(\mathbf{x} \pm \epsilon \mathbf{n}_\Gamma) \ , \tag{22}$$

In practice, the normal momentum jump is splitted in two contributions such that the second equation in Eq. (21) becomes:

$$\begin{aligned} \llbracket p \rrbracket &= \Delta p - \dot{m}_\Gamma \llbracket v_n \rrbracket \ , \\ \llbracket \tau_{nn} \rrbracket &= 0 \ . \end{aligned} \tag{23}$$

The normal and tangent bulk fluid velocities at the interface are noted respectively $v_n = \mathbf{v} \cdot \mathbf{n}_\Gamma$ and $v_t = \mathbf{v} \cdot \mathbf{t}_\Gamma$, the scalar normal and tangential projections of the shear stress tensor with respect to the normal are $\tau_{nn} = \underline{\boldsymbol{\tau}} \cdot \mathbf{n}_\Gamma \cdot \mathbf{n}_\Gamma$ and $\tau_{nt} = \underline{\boldsymbol{\tau}} \cdot \mathbf{n}_\Gamma \cdot \mathbf{t}_\Gamma$, and the normal heat flux is $q_n = \mathbf{q} \cdot \mathbf{n}_\Gamma$. The jumps Eq. (21) do not constitute sufficient matching conditions to define the problem uniquely. Consequently, they should be supplemented by additional relations. Two common kinematic conditions that are imposed in addition to the jumps concern the jumps on temperature and tangential velocity across the interface:

$$\llbracket v_t \rrbracket = \Delta v_t \, , \tag{24}$$

$$\llbracket T \rrbracket = \Delta T \, . \tag{25}$$

Moreover, for problems involving mass transfer between the phases, a phase transition model $\dot{m}^{\text{model}}$ is provided to express the net interface mass flux $\dot{m}_\Gamma$:

$$\dot{m}_\Gamma \coloneqq \rho_k (v_{n,k} - v_{n,\Gamma}) = \dot{m}_\Gamma^{\text{model}} \ . \tag{26}$$

The Δ jump quantities appearing in Eq. (21) are related to physically relevant phenomena, like e.g. surface tension (Δp) or partial slip condition (Δv_t). They are taken as constant values in this work for the purpose of verification even though in reality, they may depend on the solution variables. The mass flux model, $\dot{m}_\Gamma^{\text{model}}$, is also set at a constant value.

The Navier-Stokes model, Eq. (14), as well as the equations of state, Eq. (20), are implemented as a derived class in the *PhysicalModels* module of our software. In the two test cases that follows, we are only interested to verify the most challenging aspect our CFD solver has to deal with, namely the enforcement of the interface jumps conditions (and not the classical wall boundary conditions). Those are specified in our manufacturing software via the routines implemented in the *BoundaryConditions* module. More precisely, the two last equations in the system Eq. (21) and the second equation in Eq. (23) are added as Neumann interface conditions using the normal and tangent vectors computed from the provided interface geometry while the jumps in Eqs. (24, 25, 26) and the first equation in Eq. (23) are considered as Dirichlet interface conditions. The values chosen for the bulk properties and the jumps for both test cases are listed in Tab. 1.

		Inclined interface	Circular interface
Bulk properties	$\gamma_1\,,\ \gamma_2$	1.4 , 1.6	1.911 , 1.290
	$C_{v,1}\,,\ C_{v,2}$	2.5 , 2.0	977.704, 991.20
	$p_{\infty,1}\,,\ p_{\infty,2}$	0.0 , 0.0	7.16e9 , 0.0
	$\mu_1\,,\ \mu_2$	1.0 , 0.25	4.45e-3 , 6.98e-5
	$\lambda_1\,,\ \lambda_2$	1.0 , 0.25	35.24 , 0.127
Jumps values	$\dot{m}_\Gamma$	0.0	-0.3
	Δp	-0.2	200.0
	$\Delta\tau_{nt}$	0.2	0.2
	Δq	1.0	100.0
	ΔT	0.5	10.0
	Δv_t	-0.5	-0.5

Table 1: Physical parameters selected for the test cases of Sec. 4.2 and Sec. 4.3.

The reader interested by getting information about our sharp interface discontinuous Galerkin (DG) solver might consult [20, 21, 22].

4.2 Test case 1 - Inclined interface

The two-dimensional domain considered for this test case is $\Omega = [0,1] \times [0,1]$. The MS determination is realized on a preliminary configuration with an horizontal interface located at $y_\Gamma = 1/3$ using (x, y) Cartesian coordinates. Phase $k = 1$ is situated above the interface and phase $k = 2$ below. Because this problem involves Neumann-type conditions, we choose the initial forms of the MS to be solely dependent on the y-coordinate following the remarks formulated in Sec. 2.2. We manufacture the solutions for both phases in terms

of the primitive "p-$\mathbf{v}$-T" variables set so that:

$$\begin{aligned}
\widetilde{p}_1(y) &= -0.11\,\frac{\cos(\pi\,y/2)}{\cos(\pi/6)} + 1.21\ , \\
\widetilde{v}_{x,1}(y) &= 1.0\cos(0.75\,\pi\,y) + 2.45\ , \\
\widetilde{v}_{y,1}(y) &= 1.0\cos(0.75\,\pi\,y) + 2.45\ , \\
\widetilde{T}_1(y) &= 1.0\cos(0.75\,\pi\,y) + 2.45\ , \\
\widetilde{p}_2(y) &= \lambda_{p,1} - 2.36178756984203\cos(y) + 3.53178756984203\ , \\
\widetilde{v}_{x,2}(y) &= \lambda_{v_x,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{v_x,2}\ , \\
\widetilde{v}_{y,2}(y) &= \lambda_{v_y,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{v_y,2}\ , \\
\widetilde{T}_2(y) &= \lambda_{T,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{T,2}\ ,
\end{aligned} \tag{27}$$

with the set of unknown free parameters

$$\boldsymbol{\lambda} = \{\lambda_{p,1}, \lambda_{v_x,1}, \lambda_{v_x,2}, \lambda_{v_y,1}, \lambda_{v_y,2}, \lambda_{T,1}, \lambda_{T,2}\}$$

comprising seven components, which corresponds to the number of constraints to be satisfied. The parameters are computed from the available routines in the *ProblemSolving* module after having assembled the constraints based on the jumps conditions described earlier. The MS are then rotated by 15° in the anti-clockwise direction by means of a change of reference system, which is also provided by *ProblemSolving*. The MS is now dependent on both the x and y coordinates and can exercise all the terms in the governing equations. In addition to the MS in both phases, the source terms in both phases are also computed and printed in a file to be read by the CFD solver. The surface plot of the density field is shown in Fig. 2 while the one-dimensional profiles of the temperature and normal-tangential projection of the shear stress tensor along the y-axis for several x coordinates values are plotted in Fig. 3. We can observe that the MS indeed satisfy the jumps we have prescribed all along the interface and is therefore appropriate to be used as the reference solution for our order-of-accuracy study. This study helped us to detect flaws both in the implementation of the models as well as the numerical formulation. After correcting them, we retrieved the theoretical order of convergence of the DG method, i.e. $p + 1$ with the polynomial order p. Those results will be presented in an upcoming publication.

4.3 Test case 2 - Circular interface

This second test case follows the same procedure as the first one, except that we determine the MS in polar coordinates (r, θ) before converting it to Cartesian coordinates. The two-dimensional domain is (in Cartesian coordinates) $\Omega = [0.5, 1.5] \times [0.5, 1.5]$. We specify Dirichlet boundary conditions on velocities and temperature on the boundaries of the domain, in addition to the interface jumps conditions. The interface is a circle given by

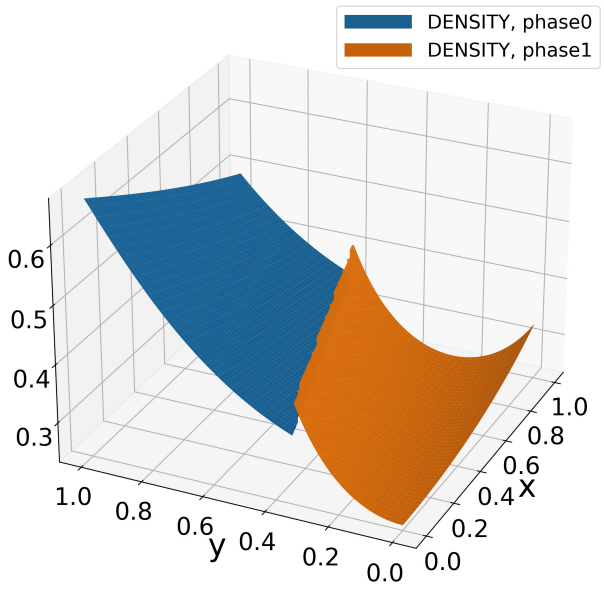
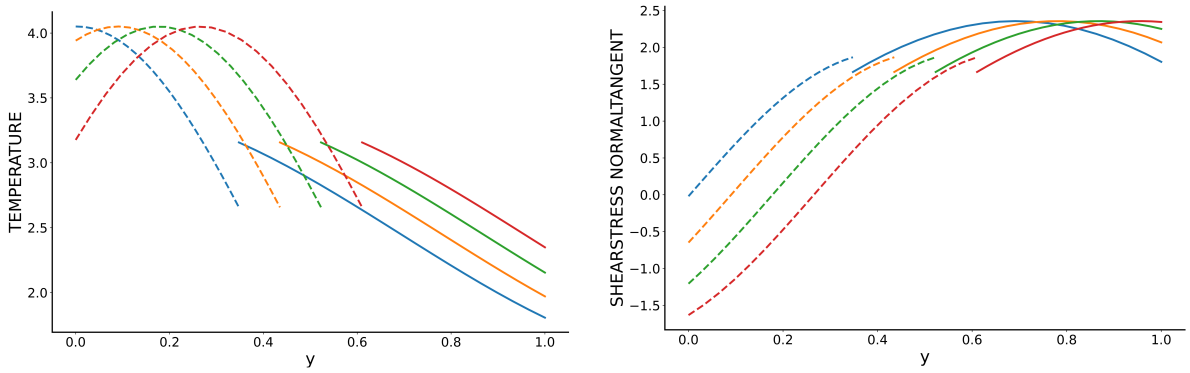


Figure 2: Surface plot of the density field over the two phases for the inclined interface test case.



(a) T field (b) τ_{nt} field

Figure 3: One-dimensional profiles along the y-axis for four different x coordinates values of manufactured fields for the inclined interface test case. Plain line is for phase 1, dashed line for phase 2.

the equation $\Gamma \equiv r = 0.25625$. Phase $k = 1$ is inside the circle, phase $k = 2$ is outside. We again choose the expressions of the MS to be uniquely dependent on a single coordinate,

r in this case. We also manufacture the solutions in the primitive variables set such that:

$$\begin{aligned}
\widetilde{p}_1(r) &= 3380.84473527662\, r^2 + 21978.0 \ , \\
\widetilde{v}_{r,1}(r) &= 59.4303622988639\, r^3 \ , \\
\widetilde{v}_{\theta,1}(r) &= 59.4303622988639\, r^3 \ , \\
\widetilde{T}_1(r) &= 1942.94288270104 - 115.466222827242\, r^3 \ , \\
\widetilde{p}_2(r) &= \lambda_{p,1} - 16751.9333729923\, r^2 + 23300.0 \ , \\
\widetilde{v}_{r,2}(r) &= \lambda_{v_r,1} + \lambda_{v_r,2}\, r + \lambda_{v_r,3}\, r^2 \ , \\
\widetilde{v}_{\theta,2}(r) &= \lambda_{v_\theta,1} + \lambda_{v_\theta,2}\, r + \lambda_{v_\theta,3}\, r^2 \ , \\
\widetilde{T}_2(r) &= \lambda_{T,1} + \lambda_{T,2}\, r + \lambda_{T,3}\, r^2 \ ,
\end{aligned} \tag{28}$$

with the set of unknown free parameters

$$\boldsymbol{\lambda} = \{\lambda_{p,1}, \lambda_{v_r,1}, \lambda_{v_r,2}, \lambda_{v_r,3}, \lambda_{v_\theta,1}, \lambda_{v_\theta,2}, \lambda_{v_\theta,3}, \lambda_{T,1}, \lambda_{T,2}, \lambda_{T,3}\}$$

comprising ten components, which corresponds to the number of constraints to be satisfied (7 jumps + 3 wall BC). After having determined the parameters, the problem is transferred to a Cartesian reference frame such that the MS gets significantly more complex to challenge the CFD solver (which solves the equations in Cartesian coordinates). The manufactured velocity field is represented in Fig. 4 and the surface plot of the temperature is shown in Fig. 5. Both figures have been generated by *PyManufSol* (from the *Outputs* module). This test case allowed us to confront our code with a non-trivial interface geometry while at the same time considering the whole set of non-linear Dirichlet and Neumann interface jumps conditions. The results of our order-of-accuracy study on this test case will also be published in a future work.

5 CONCLUSIONS AND PERSPECTIVES

We have presented the first version of a flexible open-source numerical framework using symbolic manipulations to generate solutions in the context of the Method of Manufactured Solutions (MMS) method. Compared to previous work, the general approach on which it relies offers the possibility to design potentially any complicated solutions subjected to some user-defined conditions in a convenient and efficient way. This feature makes it an ideal tool to build reference solutions aiming at verifying the treatment of boundaries,interfaces or discontinuities inside a PDE numerical solver. This has been demonstrated on two challenging test cases about a two-phase compressible Navier-Stokes equations system.

As discussed in this article, the current approach has some limitations requiring further developments that should be inspired by previous work. In order to relax the constraint on the boundaries geometry, we could investigate the regression approach proposed in [15] or the use of multiplicative functions encompassing the geometry as done in [8]. For

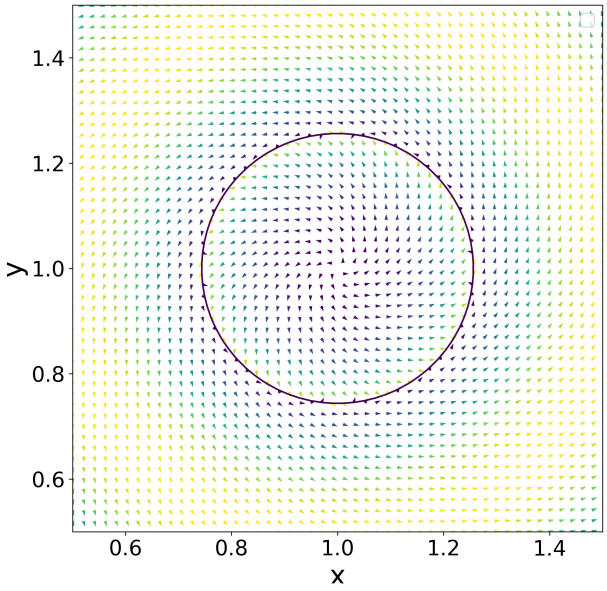


Figure 4: Velocity field over the two phases for the circular interface test case.

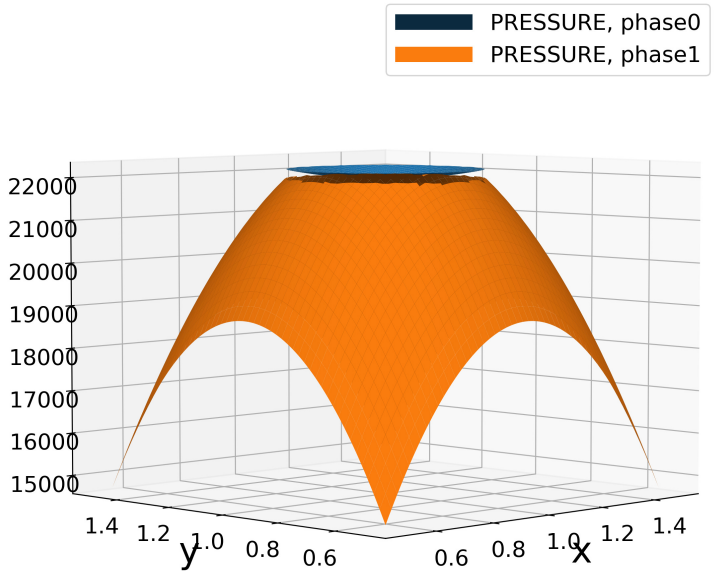


Figure 5: Surface plot of the pressure field over the two phases for the circular interface test case.

what concerns the limitations about the use of Neumann-type conditions, the approaches in [18] and [6] that manufacture some parameters instead of the solution directly could be explored.

REFERENCES

[1] Patrick J Roache. *Verification and validation in computational science and engineering*, volume 895. Hermosa Albuquerque, NM, 1998.

[2] Patrick Knupp and Kambiz Salari. *Verification of computer codes in computational science and engineering*. CRC Press, 2002.

[3] William L Oberkampf and Christopher J Roy. *Verification and validation in scientific computing*. Cambridge University Press, 2010.

[4] Stanly Steinberg and Patrick J Roache. Symbolic manipulation and computational fluid dynamics. *Journal of Computational Physics*, 57(2):251–284, 1985.

[5] Patrick J. Roache. *The Method of Manufactured Solutions for Code Verification*, pages 295–318. Springer International Publishing, Cham, 2019.

[6] Brian A Freno, Brian R Carnes, Victor E Brunini, and Neil R Matula. Nonintrusive manufactured solutions for non-decomposing ablation in two dimensions. *Journal of Computational Physics*, 463:111237, 2022.

[7] David Folkner, Aaron Katz, and Venke Sankaran. Design and verification methodology of boundary conditions for finite volume schemes. *Computers & Fluids*, 96:264–275, 2014.

[8] Ryan B Bond, Curtis C Ober, Patrick M Knupp, and Steven W Bova. Manufactured solution for computational fluid dynamics boundary condition verification. *AIAA journal*, 45(9):2224–2236, 2007.

[9] Subrahmanya P Veluri, Christopher J Roy, and Edward A Luke. Comprehensive code verification techniques for finite volume cfd codes. *Computers & fluids*, 70:59–72, 2012.

[10] Aniruddha Choudhary, Christopher J Roy, Edward A Luke, and Subrahmanya P Veluri. Code verification of boundary conditions for compressible and incompressible computational fluid dynamics codes. *Computers & Fluids*, 126:153–169, 2016.

[11] L Eça, M Hoekstra, and G Vaz. Manufactured solutions for steady-flow reynolds-averaged navier–stokes solvers. *International Journal of Computational Fluid Dynamics*, 26(5):313–332, 2012.

[12] Brian A Freno, Brian R Carnes, and Neil R Matula. Nonintrusive manufactured solutions for ablation. *Physics of Fluids*, 33(1):017104, 2021.

[13] RK Crockett, Phillip Colella, and Daniel T Graves. A cartesian grid embedded boundary method for solving the poisson and heat equations with discontinuous coefficients in three dimensions. *Journal of Computational Physics*, 230(7):2451–2469, 2011.

[14] PT Brady, Marcus Herrmann, and JM Lopez. Code verification for finite volume multiphase scalar equations using the method of manufactured solutions. *Journal of Computational Physics*, 231(7):2924–2944, 2012.

[15] Benjamin Grier, Richard Figliola, Edward Alyanak, and José Camberos. Discontinuous solutions using the method of manufactured solutions on finite volume solvers. *AIAA Journal*, 53(8):2369–2378, 2015.

[16] Nicholas Malaya, Kemelli C Estacio-Hiroms, Roy H Stogner, Karl W Schulz, Paul T Bauman, and Graham F Carey. Masa: a library for verification using manufactured and analytical solutions. *Engineering with Computers*, 29:487–496, 2013.

[17] David Henneaux. PyManufSol. `https://github.com/henneauxd/PyManufSol`, 2023. v1.0.0.

[18] Špela Brglez. Code verification for governing equations with arbitrary functions using adjusted method of manufactured solutions. *Engineering with Computers*, 30(4):669–678, 2014.

[19] Kambiz Salari and Patrick Knupp. Code verification by the method of manufactured solutions. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2000.

[20] David Henneaux, Pierre Schrooyen, Bruno Ricardo Barros Dias, Alessandro Turchi, Philippe Chatelain, and Thierry Magin. Extended discontinuous galerkin method for solving gas-liquid compressible flows with phase transition. In *AIAA AVIATION 2020 FORUM*, page 2971, 2020.

[21] David Henneaux, Pierre Schrooyen, Larbi Arbaoui, Philippe Chatelain, and Thierry E Magin. A high-order level-set method coupled with an extended discontinuous galerkin method for simulating moving interface problems. In *AIAA AVIATION 2021 FORUM*, page 2742, 2021.

[22] David Henneaux, Pierre Schrooyen, Philippe Chatelain, and Thierry E Magin. High-order enforcement of jumps conditions between compressible viscous phases: an extended interior penalty discontinuous galerkin method for sharp interface simulation. submitted.