Machine Learning design of Volume of Fluid schemes for compressible flows

Bruno Després, Hervé Jourdren

Machine Learning design of Volume of Fluid schemes for compressible flows

Bruno Després
Sorbonne-Université, CNRS, Université de Paris, Laboratoire Jacques-Louis Lions (LJLL), F-75005 Paris, France,
Institut Universitaire de France,
and
Hervé Jourdren
CEA, DAM, DIF, F-91297 Arpajon, France
Université Paris-Saclay, CEA DAM DIF, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation,
F-91297 Arpajon , France

Abstract

Our aim is to establish the feasibility of Machine-Learning-designed Volume of Fluid algorithms for compressible flows. We detail the incremental steps of the construction of a new family of Volume of Fluid-Machine Learning (VOF-ML) schemes adapted to bi-material compressible Euler calculations on Cartesian grids. An additivity principle is formulated for the Machine Learning datasets. We explain a key feature of this approach which is how to adapt the compressible solver to the preservation of natural symmetries. The VOF-ML schemes show good accuracy for advection of a variety of interfaces, including regular interfaces (straight lines and arcs of circle), Lipschitz interfaces (corners) and non Lipschitz triple point (the Trifolium test problem). Basic comparisons with a SLIC/Downwind scheme are presented together with elementary bi-material calculations with shocks.

Keywords: VOF, CFD, ML

1. Introduction

Machine Learning (ML) [19] for the construction of numerical fluxes adapted to Finite Volume (FV) discretizations is becoming a research subject of its own, see recent contributions for the discretization of hyperbolic equations by Hesthaven et al [30, 34]. For viscous incompressible flows, like bubble flows where the curvature of the interface controls the dynamics, it seems that ML techniques are established techniques now [37]: we refer to Zaleski et al. [28, 2] for the reconstruction of the curvature of interfaces and to [16] for an extension to compressible effects. The exact curvature is function of the second derivative of the function that defines the interface, and a comprehensive review centered on incompressible flows with surface tension is [17]. More general references can be found in [29]. On the contrary, for compressible non viscous flows, the interfaces are more related to contact discontinuities and material discontinuities: the dynamics of interfaces is more similar to the one of passive scalars; so it is needed to address interfaces with low regularity. In this context, we refer to [10, 36] for historical references on VOF methods (KRAKEN code, YOUNGS method) for compressible flows and [31, 27] for a more comprehensive presentation of the topic (algorithms LVIRA, ELVIRA and GRAD). Another name in the field is the PLIC method (Piecewise linear interface calculation) [32], the Youngs algorithm being an example. Recent developments on the MOF method which adresses high order extensions with different ideas are described in [1]. The objective of this work is to explain that an avenue completely different from PLIC, YOUNGS, LVIRA, ELVIRA, GRAD or MOF can be walked through for compressible solvers, which is the development of a ML strategy. ML techniques have their own philosophy and techniques [19, 6] since they are not based on analytical formulas, but on the construction of large datasets and on algorithmic learning of essential features encoded in these datasets. This class of methods has proved to be efficient for image identification and image comparison, so we believe it makes sense to consider that interface reconstruction and VOF procedures can also be addressed within the ML paradigm. Also it is reasonable to state that the performances of standard VOF methods for triple point problems often encountered for multi-material problems (3 phases and more, 2 phases near a wall, 2 materials sliding on a 3rd one, ...) suffer restrictions, with the notable exception of the new family of MOF methods [1]. This class of problems, which is our longterm objective, is another reason why the desire to establish the feasibility of ML techniques in the context of VOF. In this work, we do not make an extensive comparison of the quality of our new solvers with respect to the literature, even if some simple comparisons will be proposed with the SLIC

method of Noh and Woodward [26] implemented as the anti-diffusive scheme of Lagoutière [23] (it will be referred to as the SLIC/Downwind scheme in the core of this article). We concentrate hereafter on the feasibility of this new family of methods.

The kind of ML algorithms that we use corresponds to supervised training. We refer to [19, 6] for a state of art description of this approach. In more numerical words, it corresponds to the construction of an approximate/interpolated function defined through given points, the ensemble of these points is the dataset. The two main steps are the construction of one (or more) dedicated dataset(s), then the construction of the approximate function (this is called the training session). Both steps are crucial for the quality of the final results. This methodology is very classical, however the recent progresses of publicly available dedicated softwares make this task easy and powerful in terms of the size of the dataset which can be large and in terms of the quality of the training which is based in our case on dense Convolution Neural Network with many layers. A recent observation is that Neural Networks with high number of layers are quite efficient (we reach the same conclusion) and that the ReLU activation function provides enough accuracy (this is confirmed by the recent theoretical works [9]). It explains why we do not base our training session on any high order activation functions within a 2-layer structure (contrary to [28]), but as already mentioned on the ReLU function within a higher number of layers (up to 5 in our case).

Another key ingredient in our approach is the use of the Lagrange+remap strategy [4] on a Cartesian grid for the compressible flow solver (our implementation is based on [23, 14]). The main virtue of Lagrange+remap schemes is the natural decoupling between a Lagrange step where the acoustic part is treated and the interfaces are considered as fixed, and a remap (or projection) step where only the transport part of the equations is treated and the interfaces move. Lagrange+remap schemes are usually deployed as Finite Volume schemes, so there are naturally conservative in masses, total momentum and total energy: this is a valuable property for an accurate calculation of shocks. Also it has been proved in [23] that the stability of the first order scheme (in terms of the preservation of the bounds for the volume fractions and of the entropy inequality) is independent of the flux for the volume fractions in the remap stage. The Cartesian grid provides a simplification of the data structure which is amenable to reduce the computational burden for the description of the local geometry of the mesh. In practice the quality of the numerical treatment of the interfaces depends of the interface reconstruction technique in the remap stage, ultimately only the flux of mass fractions or volume fractions matters. These features are very specific to Lagrange+remap compressible schemes.

This brief tour of ML features and Lagrange+remap features motivates the development of a ML flux function which aims at an accurate transport/remap of a reconstructed interface. Following Hesthaven's approach [30, 34], we focus on the numerical construction of a flux function where the inputs contain the local volume fractions and output is the flux. That is the reconstruction of the interface features (in terms of curvature, angle at the corners, ...) is not the main goal, only the remap Finite Volume flux. One advantage of this approach is that it is versatile with respect to the type of different interfaces: we will consider straight lines, arcs of circle, corners with different angles (right angles, acute angles, ...) and even a non Lipschitz profile to emulate a simple triple point geometry. The new VOF-ML schemes are restricted to the dimension $d = 2$ on uniform Cartesian grids, however the proposed methodology can be used a priori for the reconstruction of internal boundaries in many fields, like in references [3, 5] and references therein.

In summary, the main steps of the development of our VOF-ML schemes are firstly the design of good datasets and accurate training, and secondly the incorporation of new fluxes in Lagrange+remap schemes. We will adopt an incremental presentation of our methods. The contributions of this work can be summarized as follows.

• We construct new VOF-ML schemes adapted to bi-material compressible Euler calculations and describe the calculation chain which is made of (1) construction of good datasets, (2) training session and (3) modification of a Lagrange+remap solver. The accuracy of the critical steps is controlled uniformly over the chain (here around 1%). In the training, we used up to 5 layers of neurons (4 dense hidden layers of neurons).

• We show that VOF-ML has a good ability to recover regular interfaces, details of Lipschitz interfaces, such as the corners of the Zalezak test problem [27], and even non Lipschitz interfaces (a new Trifolium test problem is proposed). Since the methodology is quite general in terms of the features of the interface, it is an improvement with respect to curvature reconstruction only [28] and to straight line interface reconstruction only [27].

• The implementation proposed in this work preserves the natural symmetries on a Cartesian mesh.

• The cost of the new VOF-ML schemes scales as

$$\mathsf{T}_{\mathsf{total}} = \mathsf{T}_{\mathsf{solver}} + \frac{\mathsf{C}^{\mathsf{ML}}_{\mathsf{interface}}}{N_{\mathsf{1D}}} \tag{1}$$

where $\mathsf{T}_{\mathsf{solver}}$ is the cost of the Finite Volume solver, $\mathsf{C}^{\mathsf{ML}}_{\mathsf{interface}}$ is a constant which depends of the geometry of the interface and of the new VOF-ML schemes and N_{1D} is the number of cells in one direction. This feature was expected and it is not an original one with respect to the literature. A proof is provided in 2D at the end of this work, the same scaling holds in 3D. Numerical measurements show that $\mathsf{C}^{\mathsf{ML}}_{\mathsf{interface}}$ can be high with respect to the unit cost of the FV solver. However, asymptotically for large N_{1D}, the cost of the new VOF-ML schemes is negligible. Our contribution is the numerical observation that, for Lagrange+remap calculations, $\mathsf{C}^{\mathsf{ML}}_{\mathsf{interface}}$ is bounded uniformly with respect to the mesh size.

The organization is as follows. Section 2 explains the algorithmic structure. We introduce the key ideas of our approach for straight line interfaces in Section 3. In next Section 4, we generalize the material to other types of interfaces and construct a dimensionless flux. Section 5 is dedicated to technical details about the inference in a C++ code and Section 6 presents an implementation of dimensionless flux which respects natural symmetries and the maximum principle for the mass and volume fractions. The bi-material compressible model which serves in the numerical section is given in Section 7. The numerical results are presented and discussed in Section 8. We end with a conclusion and perspectives. More mathematical comments are in the appendix section, with the description of a new Trifolium test problem and a test problem with vorticity beyond pure solid body rotation.

Notation. Depending on the context, the volume fraction (which is between 0 and 1) will be denoted as α or a_1. For similar reasons, the interpretation of the indices will be provided by the context. This abuse of notation has the advantage of never using the heavy notation $(a_1)_{ij}$ which is the value of the volume fraction a_1 in cell (i, j).

2. Description of the calculation chain

As stated in the introduction, this work is based on an incremental calculation chain where one step is performed after the other. Since this incremental algorithmic structure has a great influence on the organization of the calculations, we describe below some details and the software stack. A software may change without changing the chain structure.

• **First step** is the description of the geometry and construction of datasets. These datasets contain vectors of inputs of size less or equal to 26 in our case in 2D. In some of our tests, the number of vectors for the training dataset runs up to $1.4\ 10^6$ and up to $3.5\ 10^5$ for the validation dataset. The numerical values which determine the vectors are obtained by computing simple integrals which correspond to areas. We use Python code to perform these calculations.

• **Second step** is the training session which is performed with the Keras-Tensorflow suite in this work. It consists in the construction with a stochastic gradient algorithm of a function which interpolates the training dataset at best. The structure of this function depends on the number of layers and neurons per layer. The validation dataset is used to assess that the level of overfitting or underfitting is under control (which is the case in our calculations).

• **Third step** consists in passing the function to a C++ code for the inference (which means calling the function). Various possibilities exist so far. One can built an API but it has been considered as too heavy so far. Fortunately simple softwares exist, for example frugally-deep [15], kerasify [21] and keras2cpp [20], which all three can be used to call the function from C++. Preliminary tests show that frugally-deep is a possibility, but the unitary cost of calling the function is too high (≈ 0.03 s). Since it is needed to call the function in all cells near the interface and at all time steps, the CPU cost in a C++ code has been considered too heavy with frugally-deep. The software keras2cpp is used here.

• **Fourth step.** It concerns the implementation of the new VOF-ML scheme in our finite volume Lagrange+remap solver. In particular, attention is paid to the preservation of natural symmetries since it is not immediate with a VOF-ML scheme (here with accuracy around 1%).

3. Straight lines: a case of study

Straight-line interfaces serve as a case of study to present the basic features of the construction of the datasets and of the training session. It is generalized to more general types of interfaces in the next sections.

3.1. Parametrization of the geometry

For our CFD calculations, we use a Two Dimensional (2D) Cartesian mesh with a mesh size $\Delta x > 0$. As required in Learning textbooks [19, 6], it is better to normalize the data. Normalization is also a standard technique in CFD, so we believe it is a good idea to systematically use normalized data in our ML procedure. This is why we consider a normalized Cartesian mesh

$$C_{ij} = \left\{ \mathbf{x} = (x_1, x_2) \in \mathbb{R}^2 \mid -\frac{1}{2} + i < x_1 < \frac{1}{2} + i \text{ and } -\frac{1}{2} + j < x_2 < \frac{1}{2} + j \right\}.$$

By construction the area of all square cells is 1. The numeration is such that the central point is the center of mass of the central square C_{00}. To have notations compatible with the ones needed for the description of ML algorithms, the description of the local geometry needed for VOF methods starts from the data of two functions

$$D : \mathbb{R}^{\mathsf{par}} \longrightarrow \mathbb{R}^{\mathsf{in}} \text{ and } E : \mathbb{R}^{\mathsf{par}} \longrightarrow \mathbb{R}^{\mathsf{out}},$$

where

• $\mathsf{par} \in \mathbb{N}^*$ is the dimension of the space of parameters which describe the interfaces between the fluids,

• $\mathsf{in} \in \mathbb{N}^*$ is the dimension of the space of inputs,
• $\mathsf{out} \in \mathbb{N}^*$ is the dimension of the space of outputs.

The goal is to construct with ML algorithms a third function

$$F : \mathbb{R}^{\mathsf{in}} \longrightarrow \mathbb{R}^{\mathsf{out}} \tag{2}$$

such that

$$F(D(z)) \approx E(z) \text{ for all } z \in \mathbb{R}^{\mathsf{par}}. \tag{3}$$

If the function D has a left inverse $D^{-1} : \mathbb{R}^{\mathsf{in}} \longrightarrow \mathbb{R}^{\mathsf{par}}$, then the best solution is to take $F = EoD^{-1}$ (the situation can be thought as similar to the one of auto-encoders, in ML language [19]). In what follows, we will use a ML software to construct a function F which realizes (3) to the best, without even questioning about the invertibility of D. We detail hereafter the functions D and E used in this work.

3.2. Straight lines

The first and main example directly comes from the VOF literature [10, 36, 31, 27]. For small mesh sizes, regular interfaces between fluids are asymptotically straight lines: that is why we consider only the latter limit case in this section. This example has a pedagogical virtue, and other cases will be variations around this theme. Straight lines

$$I_{\theta,r}(x_1, x_2) = \cos\theta x_1 + \sin\theta x_2 - r, \tag{4}$$

are described by 2 parameters $(\theta, r) \in [0, 2\pi) \times \mathbb{R}^+$, as illustrated in Figure 1.

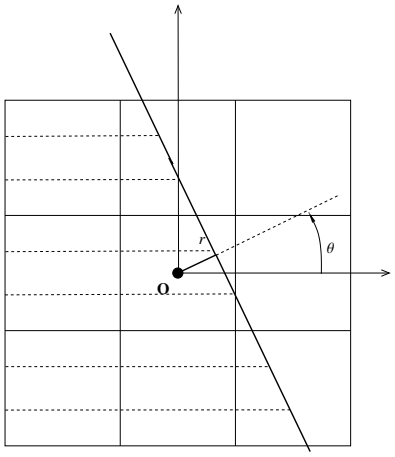

Figure 1: Description of the parameters (θ, r) for straight lines.

The straight line delimits two half planes

$$P^- = \{I_{\theta,r}(x_1, x_2) < 0\} \text{ and } P^+ = \{I_{\theta,r}(x_1, x_2) > 0\}.$$

The intersection of square cells $C_{i,j}$ with the first half plane $\{I_{\theta,r}(x_1, x_2)\} < 0$ yields the volume fractions

$$\alpha_{ij} = \int_{C_{ij} \cap P^-} dx_1 dx_2$$

which depend on the parameters θ and r. By definition the volume fractions are normalized between 0 and 1, that is $\alpha_{ij} \in [0, 1]$. Considering the other half plane P^+ would result in calculating $1 - \alpha_{ij}$ instead of α_{ij}. To construct the inputs, we decide of a certain block $(2N + 1) \times (2N + 1)$ of cells, symmetric with respect to the central cell, and gather the volume fractions in one vector

$$\alpha = (\alpha_{i,j})_{-N \leqslant i,j \leqslant N} \in [0, 1]^{(2N+1)^2} \subset \mathbb{R}^{(2N+1)^2}.$$

With these notations, the function D is

$$D(\theta, r) = \alpha. \tag{5}$$

Then one has to decide the output which might be the periodic angle $\theta \in [0, 2\pi)$ in this example. However forcing periodicity is not in the default implementation in Keras-TensorFlow. Therefore it is better for implementation purposes to work with 2 outputs which are the components $(\cos\theta, \sin\theta)$ of the direction, because the 2π periodicity of the angle is naturally taken into account and this procedure is easy to generalize in 3D. So our function E is

$$E(\theta, r) = (\cos\theta, \sin\theta)\,. \tag{6}$$

By construction the outputs are also normalized. For this example $(\mathsf{par}, \mathsf{in}, \mathsf{out}) = (2, (2N+1)^2, 2)$.

3.3. Basic properties of ML methods

To have a self-contained presentation, we summarize hereafter the basic mathematical features which are used to construct the function F defined in (2-3). More theoretical material is proposed in the appendix to explain the good properties of ML algorithms for our purposes.

3.3.1. Functions

ML frameworks make use of high level recursive numerical functions, and three basic functions are sufficient to construct the function F.

Linear algebra. Functions like $y = WX + b$ are implemented for $X \in \mathbb{R}^n$, $W \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

Non linearity. Some non linear functions are implemented for $x \in \mathbb{R}$, such as the sigmoid $\sigma(x) = (1 + \exp(-x))^{-1}$ or the rectified linear unit (ReLU) function $R(x) = \frac{1}{2}(x + |x|) = \max(0, x)$.

Recursivity. Recursive calculations like $W^2R(W^1X + b^1) + b^2$ or $R\left(W^2R(W^1X + b^1) + b^2\right)$ are implemented.

These functions can be used to construct other functions like $\max(a, b) = a + R(b - a)$ and $\min(a, b) = -\max(-a, -b) = a - R(a - b)$. Many more linear and non linear functions (convolution neural network, maxpool, ...) are also implemented in Keras-TensorFlow [6, 33]. Once a combination is chosen, the structure of the function F is known. For example a function with one hidden layer (parameters W^1, b^1) is $F(X) = W^2R(W^1X + b^1) + b^2$, a function with two hidden layers is $F(X) = W^3(R\left(W^2R(W^1X + b^1) + b^2\right) + b^3$, and so on and so forth. A key feature of modern ML software is the calculation of the coefficients $W^1, b^1, \ldots$ of a function F which is represented or approximated within the above structure. This is performed by optimization of a functional which encodes (2-3). This optimization uses stochastic gradient descent with automatic differentiation for the calculation of the gradient. The ReLU function R is piecewise differentiable, its derivative is the Heaviside function almost everywhere: it allows symbolic differentiation with the chain rule to calculate the first gradient of functions defined recursively. Comprehensive references can be found in [6, 33].

3.4. Elementary VOF advection and ML

We detail in this Section two simple reasons why the non linear ReLU function R is one of the best, better than a sigmoid σ, for application to VOF reconstruction on Cartesian grids.

The first reason is that many Finite Volume schemes with second-order accuracy use limiters as

$$\mathsf{minmod}(a, b) = \begin{cases} 0 & \text{for } ab \leqslant 0, \\ \min(|a|, |b|)\mathrm{sign}(a) & \text{for } ab > 0. \end{cases}$$

One has the formula $\mathsf{minmod}(a, b) = R(a - R(a - b)) - R(-a - R(b - a))$ which shows that limiting techniques are also prone to be rewritten recursively with the ReLU function R. In view of the mathematical theory of hyperbolic equations [18], it is also instructing to remark that the ReLU function R is deeply connected to Kruzkov entropies $\eta_k(x) = |x-k| = R(x-k) + R(k-x)$.

The second reason is based on the following geometric observation. Consider Figure 2 where, on the left and central parts, an interface splits a 2D cell with area Δx^2 in 2 pieces. In terms of a reconstructed volume fraction, it corresponds to a function

$$\alpha(x, y) = \begin{cases} 1 & \text{for } 0 < x < \gamma\Delta x, \quad 0 \leqslant \gamma \leqslant 1, \\ 0 & \text{for } \gamma\Delta x < x < \Delta x. \end{cases}$$

Consider that a velocity $u^* > 0$ is given and compute the flux at the right boundary during a time $0 < \Delta t \leqslant \frac{\Delta x}{u^*}$

$$f(\Delta t) = \int_{y=0}^{\Delta x}\int_{x=(1-\beta)\Delta x}^{\Delta x} \alpha(x, y)dxdy, \qquad \beta = \frac{u^*\Delta t}{\Delta x}.$$

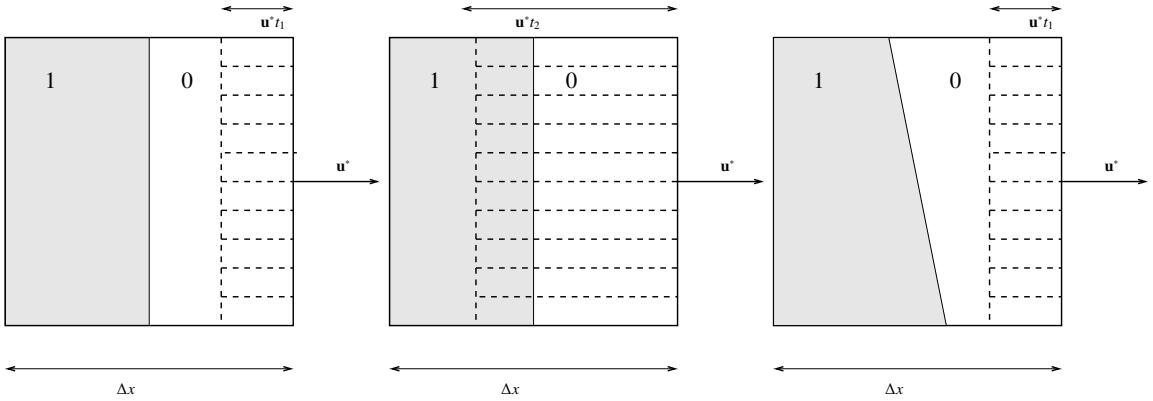

Figure 2: Description in a square cell of the swept region delimited by a moving interface. The swept region (in dashed) depends on the time t. In gray the value of the indicatrix function is 1, in white the value is 0. On the left part, the time is $t_1 = \frac{\Delta x}{4u^*}$: the flux is $f_1 = 0$ and the mean volume fraction is $\frac{f_1}{\Delta x^2/4} = 0$. On the center part, $t_2 = \frac{3\Delta x}{4u^*}$: the flux is $f_2 = \frac{\Delta x^2}{4}$ and the mean volume fraction is $\frac{f_2}{3\Delta x^2/4} = \frac{1}{3}$.

An exact calculation yields $f(\Delta t) = \Delta x^2 R(\beta - \gamma)$ that is

$$f(\Delta t) = \Delta x R\left(u^* \Delta t - \gamma \Delta x\right). \tag{7}$$

One recognizes the SLIC method [26, 27] or the antidiffusive scheme [23]. The formula (7) shows that the rectified linear unit function has the ability to be exact for some elementary VOF procedure. On the right part of the Figure, the interface has an angle and a reasonable approximation is $f(\Delta t) \approx \Delta x \alpha_1 R\left(u^* \Delta t - \gamma_1 \Delta x\right) - \Delta x \alpha_2 R\left(u^* \Delta t - \gamma_2 \Delta x\right)$ with $\alpha_1, \alpha_2, \gamma_1$ and γ_1 conveniently chosen.

More arguments in favor of using the function R are in the recent works [9, 35]. These basic observations are the reason why we use only the ReLU function R in our tests, because we believe it is more adapted.

3.5. Application to ML angle reconstruction for straight lines

We address the interpolation features of Keras-TensorFlow and measure its ability to construct the function F for the reconstruction of the angle. A reference, but with a completely different method, is the ELVIRA algorithm [27] which is *exact* for the reconstruction of the angle for straight line interfaces.

This section is a continuation of the pedagogical example Section 3.2. The vectors in the ML dataset $\mathcal{B}$ are constructed as follows. The distance to the origin, parameter r, is sampled uniformly in the range $(-0.5\sqrt{2}, 0.5\sqrt{2})$ (here 41 values): so the central cell is always crossed by the interface (even if at a corner of the cell in the extreme case). We sample the angle θ uniformly in $[0, 2\pi)$ every degree (that is 360 values). It yields $n_\theta \times n_r$ vectors in $\mathbb{R}^9$ for 3×3 blocks, or in $\mathbb{R}^{25}$ for 5×5 blocks. The volume fractions are calculated with numerical integration in 2D ($N_{\text{quadra}} \times N_{\text{quadra}}$ points with $N_{\text{quadra}} = 100$). The parameter N_{quadra} is the number of integration point per dimension and it yields a quadrature error

$$\epsilon_{\text{quadra}} \approx \frac{1}{N_{\text{quadra}}} = 1\%. \tag{8}$$

It produces $\text{card}(\mathcal{B})$ vectors $z_i \in \mathbb{R}^{\mathsf{par}}$. Then we apply the function D in (5) which calculates volume fractions: it yields $\text{card}(\mathcal{B})$ vectors

$$(X_i)_{1 \leqslant i \leqslant \text{card}(\mathcal{B})}, \qquad X_i = D(z_i) \in \mathbb{R}^{\mathsf{in}}, \quad \mathsf{in} = (2N+1)^2.$$

We also apply the function E in (6) which calculates the normal to the interface: it yields $\text{card}(\mathcal{B})$ vectors

$$(y_i)_{1 \leqslant i \leqslant \text{card}(\mathcal{B})}, \qquad y_i = E(z_i) \in \mathbb{R}^{\mathsf{out}}, \quad \mathsf{par} = 2.$$

Systematically, 80% of the data belongs to the training dataset while 20% of the data belongs to the test dataset: this choice is made at random. At the end of this stage we have a dataset for training

$$\mathcal{B}_{\mathsf{train}} = \left\{ (X_i, y_i),\ 1 \leqslant i \lesssim \frac{4\ \text{card}(\mathcal{B})}{5} \right\}$$

and a dataset for testing

$$\mathcal{B}_{\text{test}} = \left\{ (X_i, y_i), 1 \leqslant i \lesssim \frac{\text{card}(\mathcal{B})}{5} \right\}.$$

We use a dense two-layer ML reconstruction: the number of neurons of the dense hidden layer is systematically equal to the size of the inputs (parameter in), for example in=9 neurons for the first line of Table 1. Theoretically we postulate that an approximation formula (see Appendix A) holds for the function

$$\begin{aligned} F: \quad & \mathbb{R}^{\text{in}} \longrightarrow \mathbb{R}^{\text{par}} \\ & X \longrightarrow y = F(X) \end{aligned}$$

and we let Keras-TensorFlow find weights such that the least square error is quasi-minimized, in the context of implementation with batches,

$$\frac{1}{\text{card}(\mathcal{B}_{\text{train}})} \sum_{i=1}^{\text{card}(\mathcal{B}_{\text{train}})} |y_i - F(X_i)|^2 . \tag{9}$$

In all our tests, we use the default parameters. The stochastic gradient descent runs with the Adam-optimizer [22]. Here the batch size is 128. The number of epochs is 200.

It yields Table 1 where the third column is the number of training/validation data. The loss error defined by (9) is in column 5. The validation error (equal to (9) but with the dataset for testing instead of the dataset for training) is in column 6. We observe that they are of the same order. Actually in all training sessions, we have observed that the loss error and the validation error are systematically of the same order, which is hopefully the sign that no overfitting or underfitting is attached to the ML treatment of our data. We will not report anymore on this feature in the rest of the paper, since it is not a restriction.

The last column is the mean value of the L^2 error on the test data (equal to the square root of the error-validation). Clearly, the direction vector is recovered with an error around 2%. The results are promising.

dim	par	in	out	$\mathcal{B}_{\text{train}}/\mathcal{B}_{\text{test}}$	loss	val	mean L^2 error
2	2	9	2	11807/2953	5.4 e-4	5.5 e-4	2.3 e-2
2	2	25	2	11869/2891	1.8 e-5	1.7 e-5	4.2 e-3

Table 1: Accuracy of the direction vector $(\cos\theta, \sin\theta)$ for straight lines in 2D.

Many tests have been performed with the 2D data[1]: it has been observed that the L^2 error can be decreased by increasing the number of dense hidden layers (that is by increasing the quality of the interpolator). But here the gain is marginal. Indeed making the additional hypothesis that the predicted direction has norm equal to one, the mean L^2 error, when it is small, scales like

$$\text{mean } L^2 \text{ error} \quad \approx \sqrt{\frac{1}{\text{card}(\mathcal{B}_{\text{test}})} \sum \left(\begin{vmatrix} \cos\theta_{\text{true}} \\ \sin\theta_{\text{true}} \end{vmatrix} - \begin{vmatrix} \cos\theta_{\text{predic}} \\ \sin\theta_{\text{predic}} \end{vmatrix} \right)^2} \quad \approx \sqrt{\frac{1}{\text{card}(\mathcal{B}_{\text{test}})} \sum \left(\theta_{\text{true}} - \theta_{\text{predic}}\right)^2} = \epsilon_\theta .$$

In 2D, it means ϵ_θ is around 1% in relative unit

$$\epsilon_\theta \approx 1\%. \tag{10}$$

Considering that the numerical integration (8) has itself a maximal error ϵ_{quadra} of the same order, it is probably meaningless to increase the accuracy, without increasing the accuracy of the ML dataset (by increasing the accuracy of the numerical integration).

[1] A similar test has been performed in 3D. It is a confirmation in our framework of the 3D results in [28]. Now blocks are 27 cells or 125 cells, and the angles $(\varphi, \theta) \in [0, 2\pi) \times [0, \pi]$ which determine the planes are sampled quasi-uniformly to avoid over sampling near the poles. The numerical integration is uniform ($N_{\text{quadra}} \times N_{\text{quadra}} \times N_{\text{quadra}}$ points with $N_{\text{quadra}} = 50$). The results for the recovery of the direction vector $(\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$ for planes in 3D are displayed below and they are quantitatively the same as in 2D.

dim	par	in	out	$\mathcal{B}_{\text{train}}/\mathcal{B}_{\text{test}}$	loss	val	mean L^2 error
3	4	27	3	11313/2781	1.e-3	1.e-3	3.2e-2
3	4	125	3	11272/2822	1.3e-4	2.9e-4	1.7e-2

4. More general geometries and definition of a dimensionless VOF-ML flux

In the following, we parametrize the geometry by arcs of circle and corners which are assembled to describe interfaces more general than just straight lines, even if the principles are exactly the same. We also explain how to define and train the normalized original VOF-ML flux which will be used in the remap stage of the compressible solver.

4.1. Arcs of circle

We refer in [28] for the ML modeling of interfaces between incompressible fluids. Our implementation is slightly different because we ask that the modeling of curved lines degenerates to the one of straight lines if applied to a mesh with smaller and smaller mesh size (it allows natural mesh convergence tests).

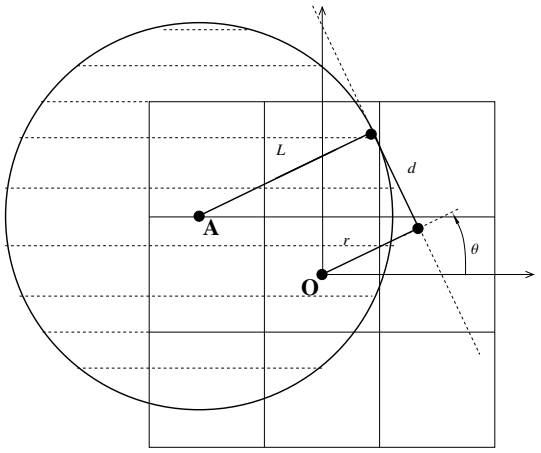

Figure 3: Description of the parameters (θ, r, d, L) for arcs of circle.

Consider the circle of Figure 3: the center is

$$\mathbf{A} = r(\cos\theta, \sin\theta) + d(-\sin\theta, \cos\theta) - L(\cos\theta, \sin\theta),$$

the radius is $L \geqslant 0$ and the offset in the direction perpendicular to $(\cos\theta, \sin\theta)$ is $d \in \mathbb{R}$. Take a point $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ inside the disk of radius L, that is $|\mathbf{x} - \mathbf{A}|^2 < L^2$. The expansion is

$$(x_1 - r\cos\theta + d\sin\theta + L\cos\theta)^2 + (x_2 - r\sin\theta - d\cos\theta + L\sin\theta)^2 < L^2.$$

Cancellation of some terms yields $2L\,(\cos\theta x_1 + \sin\theta x_2 - r) + (x_1 - r\cos\theta + d\sin\theta)^2 + (x_2 - r\sin\theta - d\cos\theta)^2 < 0$, that is

$$I_{\theta,r}(x_1, x_2) + \frac{1}{2L}\left[(x_1 - r\cos\theta + d\sin\theta)^2 + (x_2 - r\sin\theta - d\cos\theta)^2\right] < 0.$$

Using a rescaling of all lengths x_1, x_2, r, d, L by a factor which corresponds to the mesh size Δx, one gets

$$I_{\theta,r}(x_1, x_2) + \frac{1}{2}\nu\left[(x_1 - r\cos\theta + d\sin\theta)^2 + (x_2 - r\sin\theta - d\cos\theta)^2\right] < 0, \tag{11}$$

where the factor is $\nu = \frac{\Delta x}{L} > 0$. For small mesh size Δx, the continuous limit $\nu \to 0$ recovers the equation (4) of straight lines[2]. Taking $\mathsf{par} = 4$ and $\mathsf{in} = (2N+1)^2$ as before, the function $D : \mathbb{R}^{\mathsf{par}} \to \mathbb{R}^{\mathsf{in}}$ is defined by $D(\theta, r, d, R) = \alpha$. The function E will be described in Section 4.3 in formula (12).

[2]If one desires to capture instead the small radius limit, that is $L \to 0^+$, it is better to rescale the equations. Then take $\sigma = \frac{1}{\nu} = \frac{L}{\Delta x}$ and the parametrization $2\sigma I_{\theta,r}(x_1, x_2) + (x_1 - r\cos\theta + d\sin\theta)^2 + (x_2 - r\sin\theta - d\cos\theta)^2 < 0$.

4.2. Corners

The interfaces described above are smooth and so are not satisfactory for the local modeling of interfaces which are not C^1 but only Lipschitz, such as the cross test problem and the Zalezak test problem which will be considered in the numerical section. For such test problems, normal vectors at the interfaces have points of discontinuities: these points are called corners or corner points in this work. Consider the illustration in Figure 4. Straight lines correspond to $\mu = \pi$ (it is a degenerate case), right angles correspond to $\mu = \frac{\pi}{2}$ and $\mu = 3\frac{\pi}{2}$ and acute angles correspond to the other values of μ. The corner is denoted as $\mathbf{A} = (x_1^A, x_2^A)$. Actually this modeling of corners allows also to model straight line interfaces, it is sufficient either to take $\mu = \pi$ or to sample $\mathbf{A}$ outside of the block with an additional condition on the aperture of the corner.

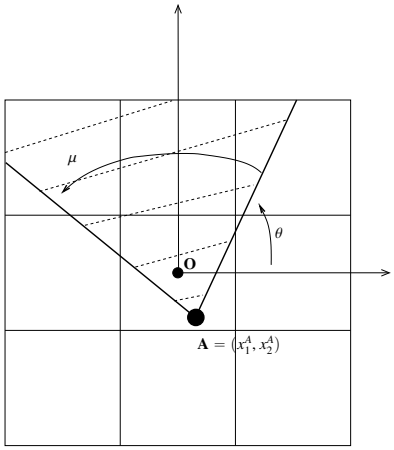

Figure 4: Description of the parameters $(\theta, \mu, x_1^A, x_2^A)$ for corners.

With these notations, one has $\mathsf{par} = 4$ and $\mathsf{in} = (2N+1)^2$. The function $D : \mathbb{R}^{\mathsf{par}} \to \mathbb{R}^{\mathsf{in}}$ is defined by $D(\theta, \mu, x_1^A, x_2^A) = \alpha$. The function E is described in the next Section 4.3 in formula (12).

4.3. Definition of a dimensionless VOF flux and adapted inputs

The function f in (7) used for the pedagogical case of straight lines is not normalized. For normalization reasons, it is better to consider the average flux which is the flux divided by the area of the swept region

$$g(\Delta t) = \frac{f(\Delta t)}{(1-\beta)\Delta x^2}. \tag{12}$$

This quantity is normalized in the sense that, on the one hand $g(\Delta t)$ can be evaluated in function of the normalized quantities $\beta \in [0,1]$ and $\gamma \in [0,1]$, and on the other hand $g(\Delta t) \in [0,1]$ under the CFL condition $\frac{u^*\Delta t}{\Delta x} \in [0,1]$. Other tests performed for the ML reconstruction of $f(\Delta t)$ show a loss of accuracy for small velocity or for small Courant number: it was expected because $f(\Delta t) \in [0, \Delta x u^* \Delta t]$ is systematically small for small Δt; on the contrary $g(\Delta t)$ is much less sensitive to small Δt. For all these reasons, we advocate using g instead of f as output for VOF flux reconstruction. This gives the function $E = g(\Delta t)$ that will be used from now on with $\mathsf{out} = 1$.

The reconstruction of the parameters of the different types of interfaces is not sufficient for the remap stage of CFD calculations. Indeed, even with an accurate reconstruction of these values, one must still construct the flux at the cell face. Therefore, to have a satisfactory ML modeling of the flux, we enlarge by one the size of the inputs. The new input is the rescaled velocity which scales like a non dimensional Courant number

$$\beta = u^* \Delta t / \Delta x \in [0,1].$$

We add it to the volume fractions, so the function D maps the parameters to a vector made of the volume fractions plus β

$$D(\text{interface parameters}) = (\alpha, \beta) \in \mathbb{R}^{(2N+1)^2+1}. \tag{13}$$

For example for a 3×3 block, the size of the inputs is now $\mathsf{in} = 1 + 9 = 10$. The output size is $\mathsf{out} = 1$: the output is the dimensionless outgoing flux

$$g = \frac{1}{\beta} \int_{\frac{1}{2}-\beta}^{\frac{1}{2}} \int_{-\frac{1}{2}}^{\frac{1}{2}} \alpha(x,y)\,dx\,dy \in [0,1] \tag{14}$$

where α is the indicatrix function, as described in (12). It yields the function E

$$E(\text{interface parameters}) = \mathrm{g} \in \mathbb{R}. \tag{15}$$

4.4. Validation of the VOF-ML flux for straight lines

Redoing the tests of Table 1 for $N = 3$ and $N = 5$, the dimensions are now $\mathsf{par} = 2$, $\mathsf{in} = 1 + (2N+1)^2$ and $\mathsf{out} = 1$. The size of the ML dataset is $100 \times 360 \times 41$. The batch size is 16×1028. The number of epochs is still 200. We use 2 dense hidden layers: a first one with $4 \times \mathsf{in}$ neurons, a second one with $2 \times \mathsf{in}$ neurons. The results are in Table 2. One observes that the results in the last column are even slightly better than in Table 1. It validates the use of the mean flux as the output.

dim	par	in	out	$\mathcal{B}_{\text{train}}/\mathcal{B}_{\text{test}}$	loss	val	mean L^2 error
2	2	10	1	1192451/298309	7e-5	7.1 e-5	9.8e-3
2	2	26	1	1193738/297022	8.8e-5	9.4e-5	6.5e-3

Table 2: Accuracy of the VOF-ML flux for straight lines.

At the end of this procedure, all the geometry (interface reconstruction, numerical integration of indicatrix functions in normalized cells, numerical integration of the flux) is offline. That is, it is considered only in the Python-Keras-TensorFlow stage of the calculation chain.

4.5. Construction of ML datasets and accuracy of the VOF-ML flux for general geometries

The construction of the ML datasets relies so far on uniform sampling of all parameters. This was possible because the number of parameters was small, equal to 2 in this example.

But we need more parameters to describe complex interfaces like arcs of circles and corners, so uniform sampling is no more possible because the number of configurations would blow up. That is why we decided, quite arbitrarily, to keep the uniform sampling of the angle θ and of the rescaled velocity β, and to take random values for all other parameters. Now the size of the ML dataset is $n_\theta \times n_\beta \times 2 \times P$: the factor 2 is because we systematically take the contrasted values ($\widetilde{\alpha} = 1 - \alpha$) and the extra factor P guarantees that the same value of the parameters θ and β are used P times while the other parameters are randomized. The sampling of the rescaled velocity β is equal to the sampling for the numerical integration, because it is convenient for implementation purposes, that is $n_\beta = N_{\text{quadra}}$.

The result for the reconstruction of the flux for arcs of circle (Section 4.1) are in Table 3, where $n_\theta = 8 \times 360$, $n_\beta = N_{\text{quadra}} = 100$, $P = 3$ and we randomize the parameters $r \leqslant 2\sqrt{2}$, $d \leqslant \sqrt{N + \frac{1}{2}}$ and $L \leqslant 2\sqrt{2}$. The error (last column) is still considered small enough.

dim	par	in	out	$\mathcal{B}_{\text{train}}/\mathcal{B}_{\text{test}}$	loss	val	mean L^2 error
2	5	10	1	1395790/349490	4.2e-5	4e-5	6.3e-3
2	5	26	1	1395422/349858	3.1e-5	4.2e-5	6.5e-3

Table 3: Accuracy of the VOF-ML flux for arcs of circle.

Finally we consider the corner configuration of Section 4.2. We take $n_\theta = 24 \times 360$ uniformly distributed angles. We take $n_\beta = 100$ values of β which are uniformly distributed. We double the results (taking the negative value $\alpha \leftarrow 1-\alpha$ and $\beta \leftarrow 1-\beta$). The parameters x_A, y_A are taken random in the square $[-3/4, 3/4]$ for $\mathsf{in} = 10$ and in the square $[-5/4, 5/4]$ for $\mathsf{in} = 26$. The parameter μ takes two different values $\frac{\pi}{2}$ and $\frac{3\pi}{2}$ at random, it models only right angles. The results are displayed in Table 4. It seems the 3×3 block has more difficulty to recover the solution, even if the accuracy is still of the same order as previous accuracies. It is probably related to some lack of invertibility of the function D for 3×3 blocks (a topic already evoked at the end of Section 3.1). It is also probably related to the lower regularity of this type of Lipschitz interfaces.

5. Inference of the model in a C++ code

Now that ML datasets are built with various parameters, and that the training and testing are performed, it remains to pass the model to a C++ code for the inference, i.e. the F function call.

dim	par	in	out	$\mathcal{B}_{\text{train}}/\mathcal{B}_{\text{test}}$	loss	val	mean L^2 error
2	5	10	1	1396604/348676	1.9e-3	2.1e-3	4.5e-2
2	5	26	1	1396744/348536	1.6e-4	2.2e-4	1.5e-2

Table 4: Accuracy of the VOF-ML flux for corners (right angles).

The software keras2cpp offers the ability of calling the function many times on tensors (which are arrays)

$$\mathcal{T}_{\mathsf{input}} \in \mathbb{R}^{N_{\mathsf{tensor}} \times \mathsf{in}}. \tag{16}$$

The size of the rectangular tensor $\mathcal{T}$ is $N_{\mathsf{tensor}} \times \mathsf{in}$ where N_{tensor} is the number of fluxes that one desires to calculate and in is the size of the inputs. The function F then returns a tensor

$$\mathcal{T}_{\mathsf{output}} = F\left(\mathcal{T}_{\mathsf{input}}\right) \in \mathbb{R}^{N_{\mathsf{tensor}} \times \mathsf{out}} \tag{17}$$

where out is the size of the outputs (actually equal to 1 in our case). This procedure allows a natural acceleration which is highly profitable for performance: a typical mean value of a call to F is between 10^{-5}s and 10^{-4}s, see the end of Section 8.

At the end of this stage, one has a function F callable in C++ which realizes the task (2-3) for the considered dataset: the output is the dimensionless flux (12).

6. Implementation in a Finite Volume solver

We provide some important details of the implementation of the method within a Finite Volume solver.

6.1. Preservation of natural symmetries

Calculations on a Cartesian mesh should in principle respect natural symmetry principles, when they are available. Moreover the accuracy of the ML interface reconstruction is achieved with an accuracy $\approx 1\%$ for the reason explained above. So any loss of symmetry can induce important discrepancies. The situation is different for incompressible fluid flow where the curvature of bubbles is recovered within an error of 10^{-6} with ML algorithms [37]. We particularize 3 different symmetries, and the consequences for the implementation of the VOF-ML scheme.

Rotational symmetry on a Cartesian mesh. The scheme must be invariant when a rotation of angle $\pi/2$ is applied. This is easy to obtain with a convenient ordering of the volume fractions. This principle is already used in standard Finite Volume implementations [24][Chapter 24], so there is no need to describe further. See Figure 5 for an illustration.

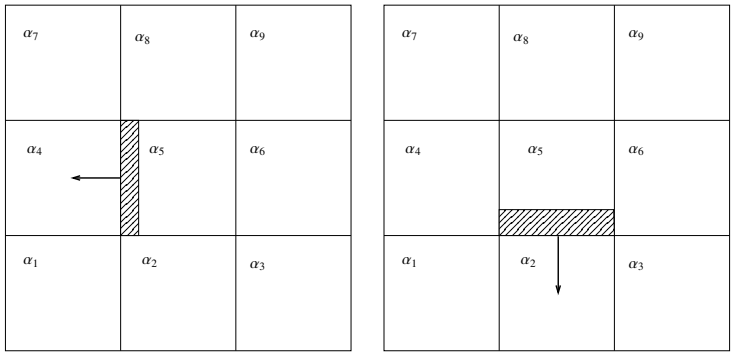

Figure 5: A rotational symmetry is described for a 3×3 block. On the left the call is $F(\beta, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$. On the right it is $F(\beta', \alpha_3, \alpha_6, \alpha_9, \alpha_2, \alpha_5, \alpha_8, \alpha_1, \alpha_4, \alpha_7)$ where the ordering of the volume fractions is the same up to a rotation. A similar procedure is used for calculation of the flux across the right boundary and the top boundary.

Mirror symmetry. Consider transport in the left direction, as in Figure 5. Let $\mathbf{Y} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$ and consider its rearrangement $\Pi\mathbf{Y} = (\alpha_7, \alpha_8, \alpha_9, \alpha_4, \alpha_5, \alpha_6, \alpha_1, \alpha_2, \alpha_3)$, where the bottom line components $\alpha_1, \alpha_2, \alpha_3$ are exchanged with the top line components $\alpha_7, \alpha_8, \alpha_9$. The function should satisfy the mirror symmetry $F(\beta, \mathbf{Y}) = F(\beta, \Pi\mathbf{Y})$, but such a symmetry is not guaranteed by our construction. So we force it by implementing

$$F_{\text{mir}}(\beta, \mathbf{Y}) = \frac{1}{2}\left(F(\beta, \mathbf{Y}) + F(\beta, \Pi\mathbf{Y})\right).$$

A similar treatment is applied for the left-right mirror symmetry.

Numbering symmetry. It means that the result should be the same if we exchange the role of material 1 and material 2 in the model, that is if we exchange a_1 and a_2. Mathematically, it can be described by considering the vector $e = (1, \ldots, 1)$ and the function $\Lambda\mathbf{Y} = \mathbf{e} - \mathbf{Y}$. We note the commutation $\Pi\Lambda = \Lambda\Pi$. The numbering symmetry corresponds to the property

$$F(\beta, \Lambda\mathbf{Y}) = 1 - F(\beta, \mathbf{Y}).$$

Since the construction does not guarantee this property, we force it by implementing

$$F_{\text{numb}}(\beta, \mathbf{Y}) = \frac{1}{2}\left(F(\beta, \mathbf{Y}) + 1 - F(\beta, \Lambda\mathbf{Y})\right).$$

The scheme which is implemented satisfies all symmetries by using

$$F(\beta, \mathbf{Y})^{\mathsf{ML}}_{\mathsf{VOF}} = \frac{1}{4}\left(F(\beta, \mathbf{Y}) + F(\beta, \Pi\mathbf{Y}) + 1 - F(\beta, \Lambda\mathbf{Y}) + 1 - F(\beta, \Lambda\Pi\mathbf{Y})\right). \tag{18}$$

This flux is called the VOF-ML flux in the rest of this article[3].

6.2. Preservation of natural bounds

It is expected that the VOF-ML flux (18) can violate natural bounds like $0 \leqslant a_1 \leqslant 1$, since no such considerations have been used so far. Of course, any kind of a posteriori method can be used to recover the maximum principle. In this work we use a simple a priori bound which is well adapted to directional splitting implementations. Essentially we constraint the flux so that no negative volume or mass is created.

Consider a square cell $(0, \Delta x)^2$ with initial volumes $0 \leqslant V_1 = a_1\Delta x^2$ and $0 \leqslant V_2 = a_2\Delta x^2$ with $a_1 + a_2 = 1$. Assume a flux a_1^* is applied on one side for a velocity $u^* > 0$, and disregard the flux on the other side. The remaining volumes at the end of the time step will be $\overline{V_1} = V_1 - \Delta x(a_1^*\Delta x u^*)$ and $\overline{V_2} = V_2 - \Delta x(a_2^*\Delta x u^*)$ where $a_2^* = 1 - a_1^*$. Note that $\overline{V_1} + \overline{V_2} < \Delta x^2$, but this is not the important point. The conditions $\overline{V_1} \geqslant 0$ and $\overline{V_2} \geqslant 0$ yield two inequalities $a_1^* \leqslant \frac{\Delta x}{u^*\Delta t}a_1$ and $1 - a_1^* \leqslant \frac{\Delta x}{u^*\Delta t}(1 - a_1)$ which are summarized in

$$1 - \frac{\Delta x}{u^*\Delta t}(1 - a_1) \leqslant a_1^* \leqslant \frac{\Delta x}{u^*\Delta t}a_1.$$

This double bound is systematically added in our implementation of the VOF-ML flux. It yields a non negativity principle, which transfers into a maximum principle for the volume fractions (and mass fractions as well). In practice we check if the VOF-ML flux (18) belongs to the interval $\left[1 - \frac{\Delta x}{u^*\Delta t}(1 - a_1), \frac{\Delta x}{u^*\Delta t}a_1\right]$: if it is in the interval we do nothing; if it is outside the interval, we replace the value of the flux with the nearest extreme value in the interval, which is equal either to the lowest value $1 - \frac{\Delta x}{u^*\Delta t}(1 - a_1)$ or to the highest one $\frac{\Delta x}{u^*\Delta t}a_1$.

6.3. Hybridation of the VOF flux

VOF schemes are often used in combination with other numerical techniques [27, 4]. For example a VOF interface reconstruction technique can be used near the interface, while a more traditional Finite Volume scheme may be used away from the interface. This is why it is probably better to conceive that VOF-ML schemes must be hybrid ones. Hybrid schemes have also the virtue that, in principle, the higher numerical cost of sophisticated VOF schemes contributes to the total cost only for cells near the interface.

The schemes developed in this article for test purposes are hybrid schemes. Various sensors have been tested, but the simplest one is retained. A small threshold value $\epsilon_{\text{flag}} > 0$ is taken and one checks at the beginning of every iteration and for all cells if a_1 is in between ϵ_{flag} and $1 - \epsilon_{\text{flag}}$:

$$\text{if } \epsilon_{\text{flag}} \leqslant a_1 \leqslant 1 - \epsilon_{\text{flag}} \Longrightarrow \text{ the cell is flagged.}$$

If the cell is flagged, one uses the VOF-ML flux (18): in the other case, we use another much cheaper method; in the Euler2D code, the anti-dissipative SLIC/Downwind flux is available, it is cheap and with strong anti-dissipative properties [23]. Considering that the quadrature error and average accuracy at the end of the training session are of the order of 1%, we take a slightly greater value

$$\epsilon_{\text{flag}} = 2\ 10^{-2}. \tag{19}$$

With this value, all thresholds (8), (10) and (19) have the same magnitude. It reflects a kind of compatibility of the whole chain of calculation.

[3] The cost of this procedure is 4 times the cost of one individual call to the function F. On a Cartesian grid in dimension d with m different materials, the cost of a similar procedure would be equal to $2^{d-1}m!$ times the cost of one individual call.

6.4. An additivity principle for the ML datasets

The additivity principle states that the ML dataset used for computing a given profile must be the union (or the addition) of ML datasets dedicated to different local features of the profile. In mathematical words, it means the ML dataset uses different functions $D_1, D_2, \ldots$, but the same function E, and that the function F is the best that ML can construct such that

$$F(D_i(z)) \approx E(z) \text{ for all } z \in \mathbb{R}^{\mathsf{par}} \text{ and for all } i = 1, 2, \ldots \tag{20}$$

This additivity principle is more a "good practice" requirement, and no theory is available so far. However its efficiency is

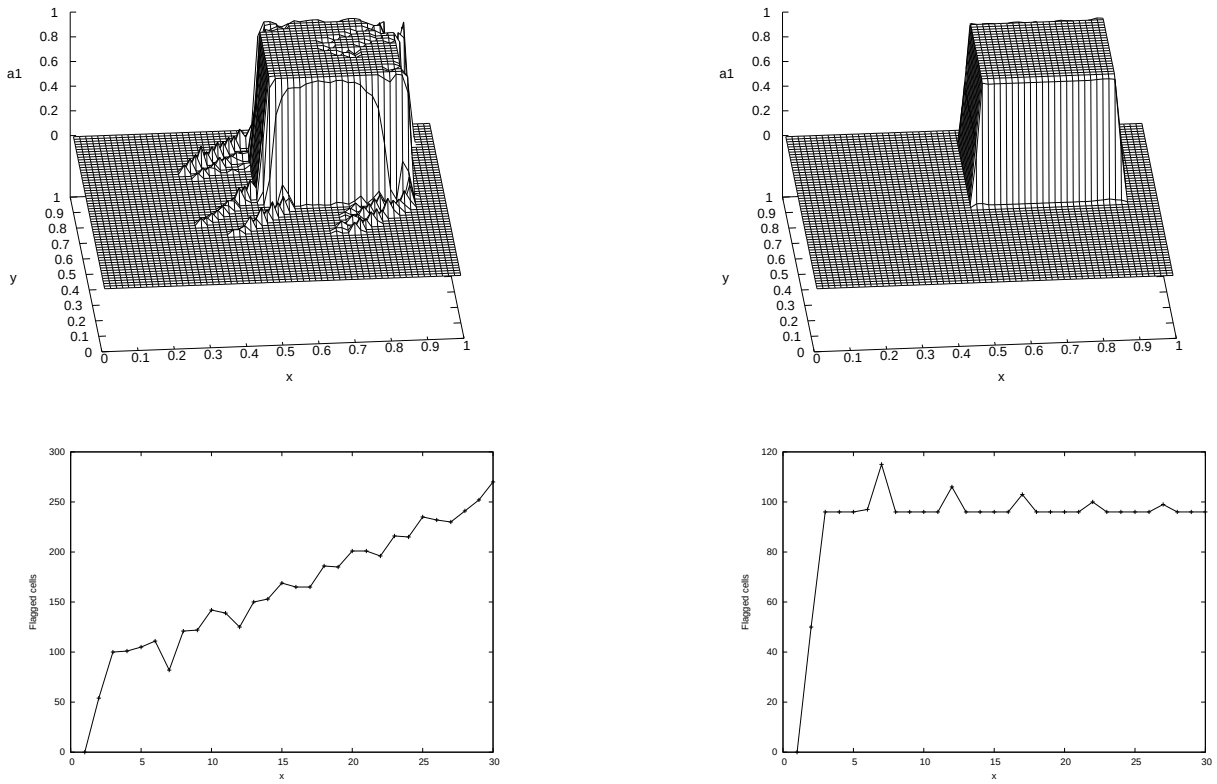

Figure 6: Value of volume fraction a_1 at time $t = 0.1$. Top left: the ML dataset has only straight lines. Top right: the ML dataset is made of straight lines and corners, following the addition principle. Bottom: the number of flagged cells versus number of time steps.

illustrated by the following simple numerical experiment. Consider the advection of a square. Locally the square is made of straight lines and corners. In the top left part of Figure 6, a simulation is displayed where the ML training is done with a dataset with straight line profiles on 5×5 blocks. It is clear that the numerical procedure is efficient for the capturing of straight lines, but the accuracy is very poor at corners. On the contrary, on the top right part of the Figure, the ML dataset is the addition of two ML datasets dedicated to straight lines and corners. Clearly, the overall accuracy is correct.

On the bottom of Figure 6, the number of flagged cells is plotted in function of the number of iterations in time. On the left part, the number of flagged cells increases linearly. It renders the computation inefficient and inaccurate. On the right part of the Figure, with the good resolution ML dataset (that is with straight lines and corners), the number of flagged cells is bounded uniformly with respect to the iterations.

Not only the ML algorithm is able to find good parameters for the advection of straight lines and corners separately, but also it is able to interpolate smoothly from case to the other. More generally we have observed in all our calculations that this principle is true:

$$\text{Additivity principle} \implies \text{Flagged cells number is bounded} \implies \text{Computational efficiency.}$$

7. A simple bi-material hydrodynamics model

Numerical tests of the VOF-ML flux have been performed in the Euler2D code developed by F. Lagoutière [23] that is described below.

In its basic setting, it uses a first-order accurate Lagrange+remap Finite Volume scheme with directional splitting [4] on a 2D Cartesian grid. The global scheme is conservative for masses, total momentum and total energy, and is endowed with strong stability properties based on entropy inequalities [23]. It solves a bi-material compressible Euler model

$$\left\{\begin{array}{l} \partial_t(\rho c_1) + \nabla\cdot(\rho c_1 \mathbf{u}) = 0, \\ \partial_t(\rho c_2) + \nabla\cdot(\rho c_2 \mathbf{u}) = 0, \\ \partial_t(\rho \mathbf{u}) + \nabla\cdot(\rho \mathbf{u}\otimes\mathbf{u}) + \nabla p = 0, \\ \partial_t(\rho e) + \nabla\cdot(\rho \mathbf{u} e + p\mathbf{u}) = 0. \end{array}\right.$$

The additivity of mass fractions, specific volumes ($1/\rho = \tau$) and internal energies is assumed

$$\left\{\begin{array}{lll} 1 = & c_1 + c_2, \\ \tau = & c_1\tau_1 + c_2\tau_2, \\ e = & c_1\epsilon_1 + c_2\epsilon_2 + \frac{1}{2}|\mathbf{u}|^2. \end{array}\right.$$

We use perfect gas pressure laws

$$p_1 = (\gamma_1 - 1)\rho_1\epsilon_1 \text{ and } p_2 = (\gamma_2 - 1)\rho_2\epsilon_2.$$

For $\gamma_1 = \gamma_2$, the model degenerates to a mono-fluid or mono-material model. Truly bi-fluid or bi-material configurations correspond to $\gamma_1 \neq \gamma_2$. The closure (that is the global pressure p) is calculated in accordance with, either the iso-pressure/iso-temperature model ($p_1 = p_2$, $T_1 = T_2$, $\varepsilon_1 = C_{v1}T_1$ and $\varepsilon_2 = C_{v2}T_2$), or the iso-pressure/iso-δQ model

$$p_1 = p_2, \quad D_t\epsilon_1 + p_1 D_t\tau_1 = D_t\epsilon_2 + p_2 D_t\tau_2.$$

The iso-pressure/iso-δQ model is our preference because the quality of the results is often better, but its implementation is non trivial because this closure model is non conservative. On the other hand the mathematical analysis of the iso-pressure/iso-temperature model is simpler, moreover its implementation is easier and the code is more robust: these are valuable features for comparisons. We will show results with both closures. If the initial data is constant pressure $p(t = 0, \mathbf{x}) = p_{\text{ref}}$ and constant velocity $\mathbf{u}(0, \mathbf{x}) = \mathbf{u}_{\text{ref}}$, then the exact solution of the model (whatever the closure) is provided by the solution of pure advection at velocity $\mathbf{u}_{\text{ref}}$. The volume fractions are related to the mass fractions and specific volumes

$$a_1 = \frac{c_1\tau_1}{\tau} \text{ and } a_2 = \frac{c_2\tau_2}{\tau} = 1 - a_1.$$

The boundary conditions can be periodic, neutral or wall conditions. Periodic boundary conditions are required for pure advection test cases. Two advection schemes are available in the Euler 2D for the remap stage. The first one is the first-order accurate upwind scheme which has poor accuracy for interfaces. The second one is the downwind scheme under upwind constraints [23]: it is similar to the SLIC algorithm [26] and will be referred to as the SLIC/Downwind scheme. The other details of such a code are standard, we refer to [23, 4].

8. Numerical results

Here we report on the accuracy of the new VOF-ML procedure, by considering basic test problems of the literature calculated within Euler2D. We distinguish pure advection and shock problems.

In order to explain what is needed for the calculation of a given type of interface, we consider VOF-ML fluxes issued from different datasets. For the Zalezak notched circle problem shown in Figure 9, the dataset is complete in the sense that it contains arcs of circles, right corners and corners with angle. In the appendix, a dedicated and somewhat artificial dataset is specially designed for the Trifolium test problem. An accurate model callable with keras2cpp for a 5×5 block trained with a ML dataset with arcs of circles, right and acute angles can be downloaded at [13]. It can be used to reproduce the results.

8.1. Advection test cases

The initial data is pressure equilibrium and the vectorial velocities are the same in all cells $\mathbf{u} = \mathbf{u}_{\text{ref}} = (a, a)$ with $a = 1$. We use periodic boundary conditions. The domain of calculation is a square with characteristic length equal to 1. The exact solution at time $t = 1$ is also the initial solution. The schemes use directional splitting. With an appropriate choice of the initial data, the sound speed c is smaller than the material velocity in both dimension, that is $c < 1$. The indicated Courant number is $CFL = a\frac{\Delta t}{\Delta x}$. In our tests, the Courant number varies between 0.1 to 0.6 in order to explore the sensibility of the algorithm. It must be noticed that Courant numbers too close to 1 are of less interests since our advection scheme becomes perfect at the limit. Also the Courant number is one parameter in the vector of inputs (parameter β): since the training session can be interpreted as an interpolation procedure, it is expected that the quality of the interpolation is smaller at the boundary of the domain of inputs (in phase space), so for Courant numbers close to 0 or to 1. For Courant numbers close to 1, this phenomenon probably mitigates the fact that an upwind advection scheme is perfect at the limit.

8.1.1. The cross

This is an extension of the square test problem displayed in Figure 6. The Courant number is 0.4. The ML dataset is made of curved lines and corners. Actually when $\nu = 0$, the curved lines (arcs of circle) dataset contains the straight lines dataset. So the accuracy for straight lines description is a priori as good as it is with only straight lines in the dataset.

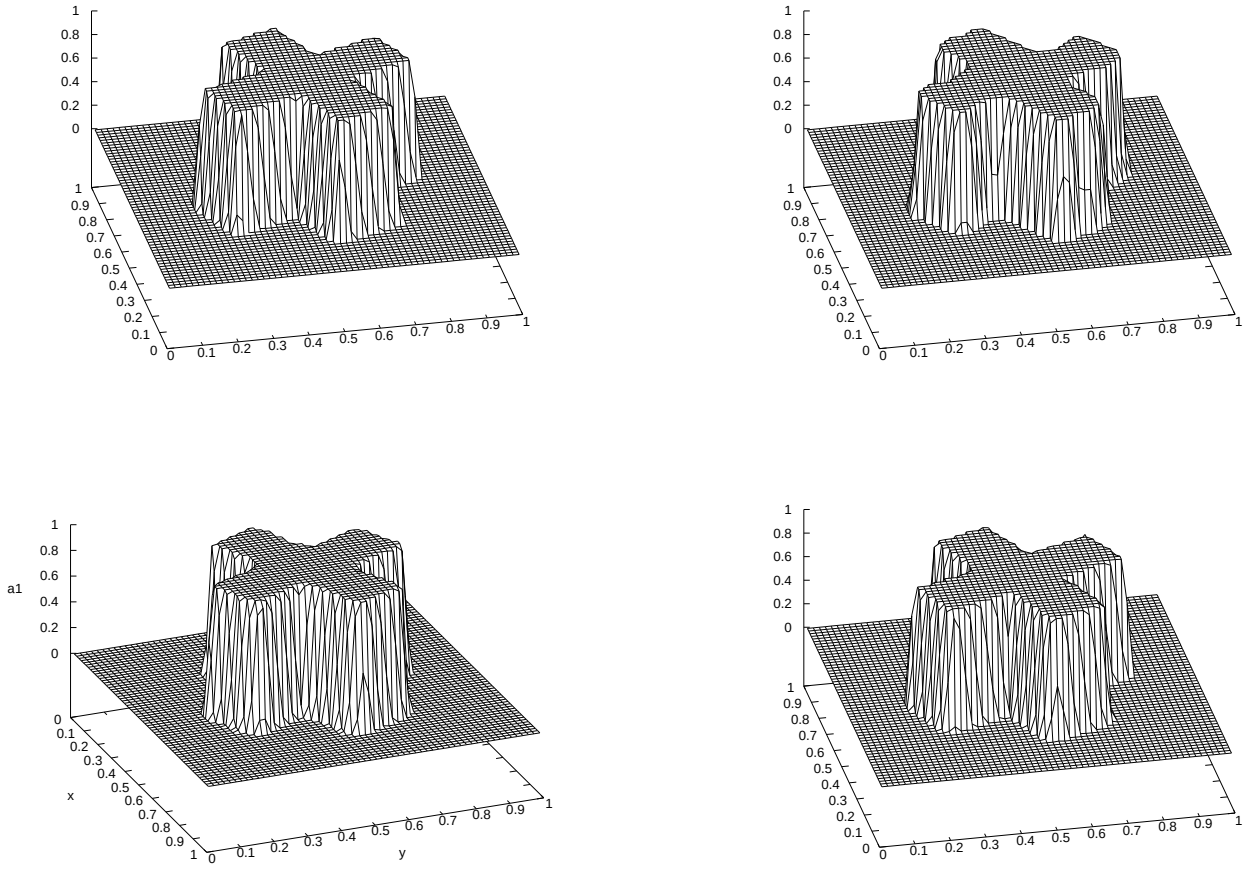

Figure 7: Value of volume fraction a_1 at time $t = 1$. Top left initial data (also equal to final data). Top right, SLIC/Downwind scheme. Bottom left, blocks are 3×3. Bottom right, blocks are 5×5.

In the ML dataset we oversample the corners (1163196 items, obtained with $4 \times 18 \times 20$ angles, only right angle corners that is $\mu = \pi/2$ or $\mu = 3\pi/2$ only, and 10 different values of the corners positions) with respect to arcs of circle (348754 items, obtained with $4 \times 360 \times 2$ angles, other parameters are randomized). In this case, the oversampling of corners is a good way to

obtain a satisfactory accuracy. The interpretation is that it counteracts the lower accuracy of corner reconstruction with respect to straight lines reconstruction (just compare the last column of Table 2 and 4). Here we use 5 dense layers in the neural network (meaning we use 5 levels of recursivity as described in Section 3.3.1).

The results are displayed in Figure 7. The results are good, in particular for the 5×5 blocks. Looking in more detail at the top right part of the Figure where the result is computed with the SLIC/Downwind scheme available in the code [23], one sees large errors because the branches of the cross are not aligned. This is not observed with the VOF-ML scheme.

8.1.2. The disk

The disk is discretized on a 40×40 cells grid. The Courant number is 0.4. The results with 3×3 blocks and 5×5 blocks are displayed in Figure 8. We also display the result obtained with the anti-dissipative scheme [23] which is over-compressive: it quickly captures an octagon instead of a disk. We use of course the ML dataset with arcs of circle. The results with the VOF-ML schemes are qualitatively accurate.

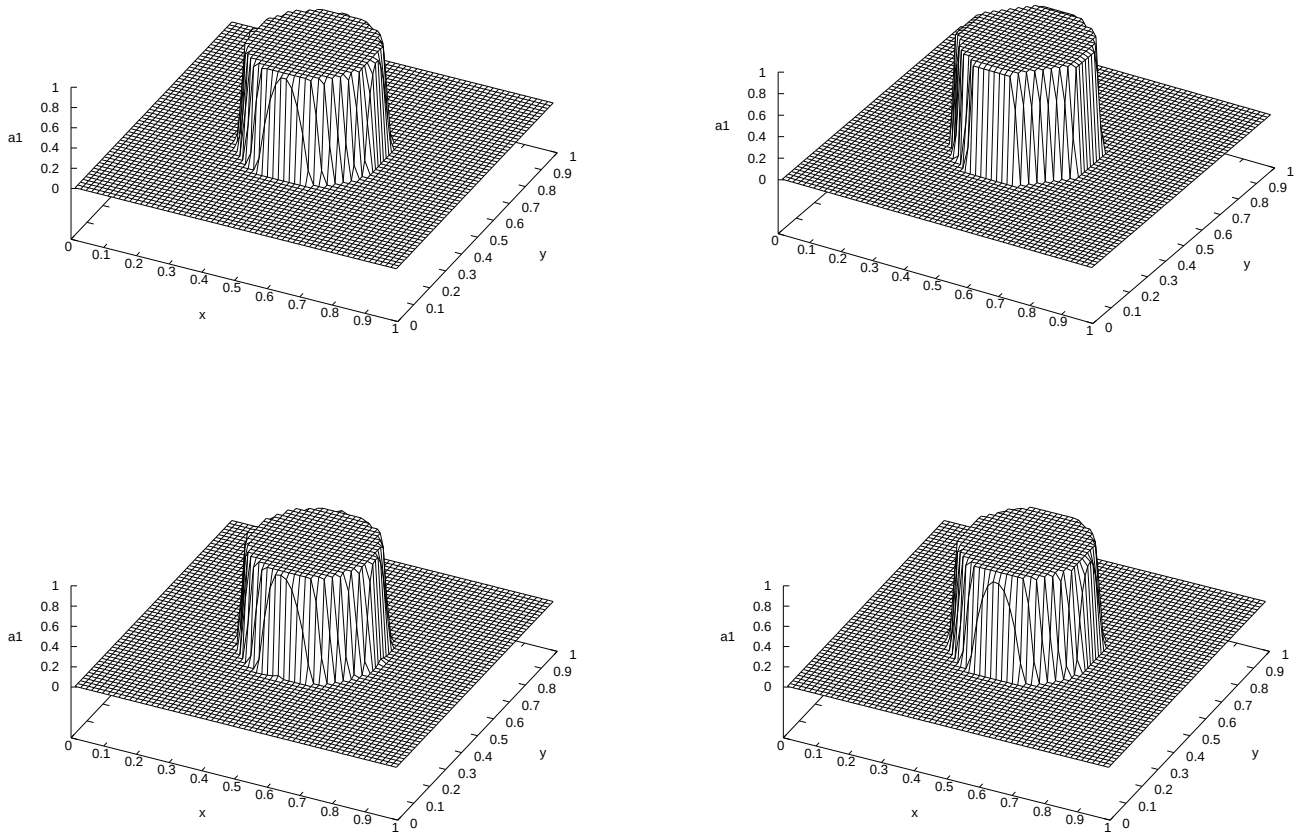

Figure 8: Value of volume fraction a_1 at time $t = 1$ after one round turn. Top left, initial and final data. Top right, SLIC/Downwind scheme. Bottom left, blocks are 3×3. Bottom right, blocks are 5×5.

8.1.3. The Zalezak notched circle

We consider the advection of the Zalezak notched circle [27] tilted $\frac{\pi}{7}$ with respect to the mesh axis. The elevations are displayed in Figure 9. The results with the SLIC/Downwind scheme are not accurate since the disk is transformed into a polygon (as for the disk). The results calculated with a 5×5 block are better but a discrepancy is visible at one of the acute angles. But, following the additivity principle, with right angles ($\mu = \frac{\pi}{2}$ or $\frac{3\pi}{2}$) and acute angles (parameter $0 \leqslant \mu \leqslant \pi/8$) in the dataset, the results are very accurate once again.

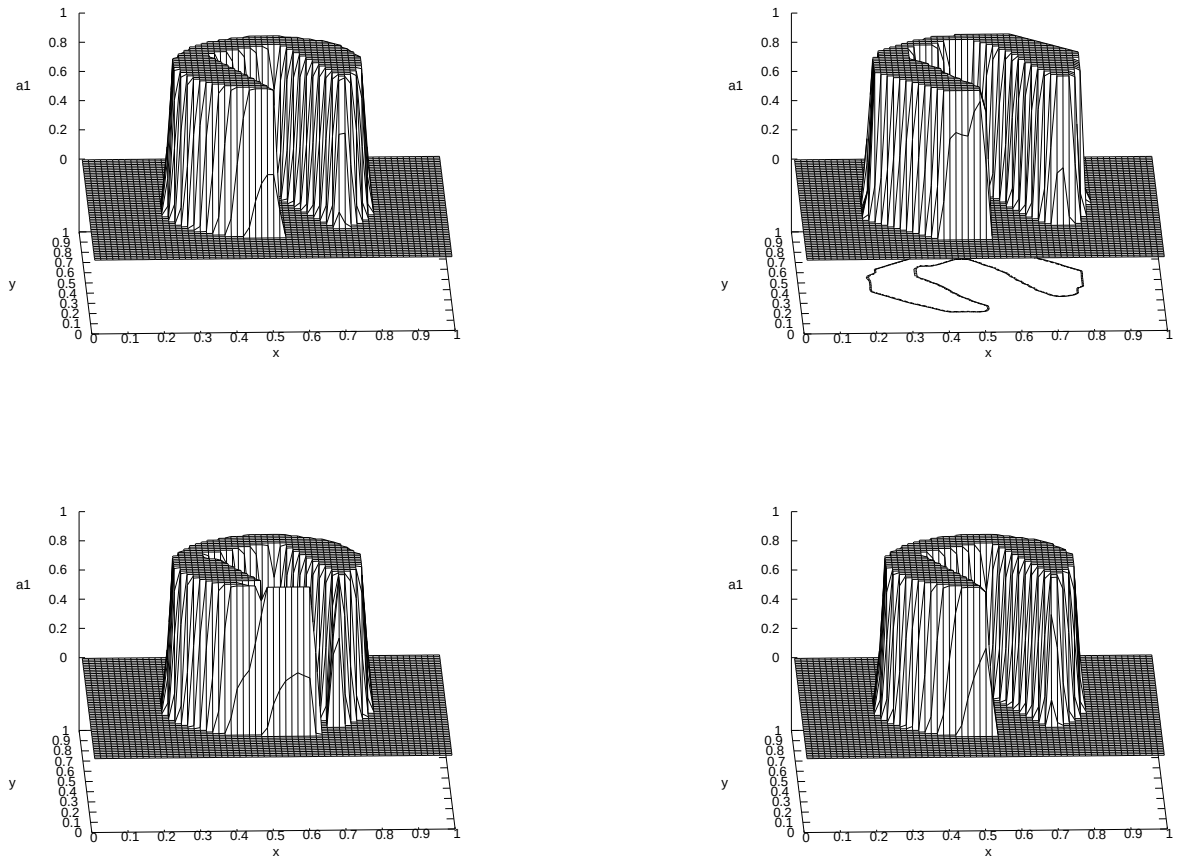

Figure 9: Value of volume fraction a_1 at time $t = 1$ after one round turn. Top left, initial and final data. Top right, SLIC/Downwind scheme: the notched circle is transformed in a polygonal line as shown by the contour lines. Bottom left, blocks are 5×5, but the ML dataset does not have acute angles. Bottom right, blocks are 5×5, now the ML dataset has acute angles. The level set of the bottom right result is presented in Figure 10.

Below in Figure 10 we reproduce the contour 0.5 of the Zalesak profile at time $t = 1$ and $t = 3$. The same setting is used (directional splitting, 15 cells in one radius). One observes that the VOF-ML scheme provides a good numerical rendering of the corners and acute angles at $t = 1$ even if a slight degradation is visible at $t = 3$.

8.1.4. Convergence tests

It is instructive to perform convergence tests to get some guarantee of the good quality of these schemes with very fine meshes. It must be mentioned that the schemes being written in a Finite Volume framework, they are conservative for local masses, total impulse and total energy. We consider the advection of 2 smooth interfaces which are a disk and an ellipse. The disk has a radius $L = 0.4$. The equation of the ellipse is $2x^2 + y^2 = L^2$. In the dataset the parameter r is carefully interpolated in the interval $[0, 1.5\sqrt{2}]$.

The results are displayed in Table 5 where we measure the difference between the initial data and the final data at time $t = 1$ in the L^1 norm. One observes a low rate of convergence, however it is remarkable that, even on a coarse mesh, the level of error is quite small in L^1 norm. Then, by refining the grid, the L^1 norm diminishes at a rate which is approximatively 1/2: that is doubling the number of cells in every direction decreases the error by approximatively a factor $\sqrt{2}$. This feature is clear on Figure 11. Therefore one can consider that the new VOF-ML schemes have the attributes of a Finite Volume scheme: they are conservative and converge in L^1 norm at rate 1/2 for bounded variation (BV) data, that is the error scales like $\approx C_{\mathrm{VOF-ML}}\Delta x^{\frac{1}{2}}$. We refer to [7, 12, 25] for estimates obtained in the context of standard numerical analysis and to [11] for an optimal estimate obtained with probabilistic tools which shows convergence of Finite Volume schemes at a rate $h^{\frac{1}{2}}$in L^1 for initial data in BV. For the completeness of this work, Table 6 provides the L^1 error norm computed with the Upwind scheme and with the SLIC/Downwind

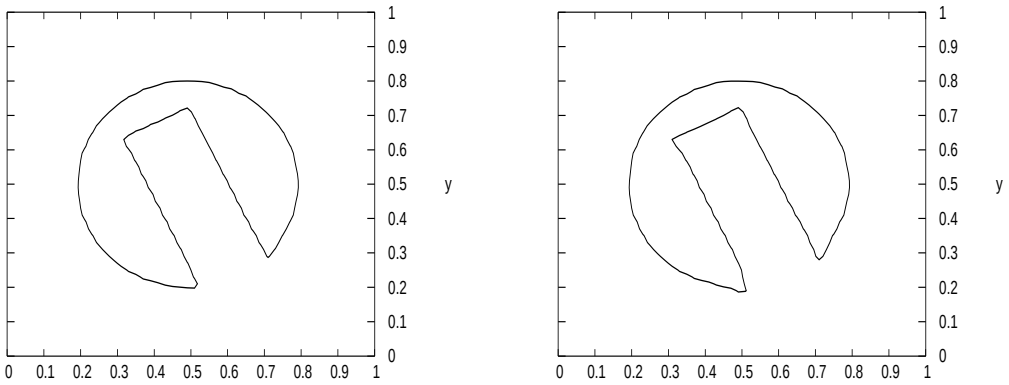

Figure 10: Level set 0.5 of the advected Zalezak profile. Value of a_1 at time $t = 1$ on the left and $t = 3$ on the right. A small degradation of the quality of the isoline at one internal corner is visible at $t = 3$.

scheme. For the Upwind scheme, one observes convergence rate $\approx C_{\text{Upwind}}\Delta x^{\frac{1}{2}}$, with a constant $C_{\text{Upwind}} \approx 600 \times C_{\text{VOF-ML}}$ which is much larger than the one of the VOF-ML scheme. For the SLIC/Downwind scheme, one observes a first order convergence rate $\approx C_{\text{SLIC}}\Delta x$: what is remarkable is that the constant $C_{\text{VOF-ML}}$ is so small so that the error of the SLIC/Downwind scheme for the 480×480 very fine mesh is still greater than the error of the VOF-ML scheme on the coarse 30×30 mesh.

In these tests, a rigorous protocol has been adopted. That is the parameters for the construction of the dataset for the 5×5 blocks are exactly the same as the parameters for the construction of the dataset for the 3×3 blocks. Also the training is done with 5 000 epochs. As a result, on the Table and the Figure, the accuracy is very similar for 3×3 blocks and 5×5 blocks. For the finer mesh, the accuracy is slightly in favor of 5×5 blocks. With less rigorous protocols for the construction of datasets, the comparison between 3×3 blocks and 5×5 blocks is less clear. Here we see that the results are of equal quality between 3×3 and 5×5 blocks. A simple explanation is that the boundaries are smooth and smooth curves tend to straight lines asymptotically for small mesh size. Since straight lines are already well captured with 3×3 blocks it is not surprising that the gain offered by 5×5 is marginal here. On the contrary, if the interfaces are less regular as in Table 4, it must be reminded that 5×5 blocks are more accurate than 3×3 blocks.

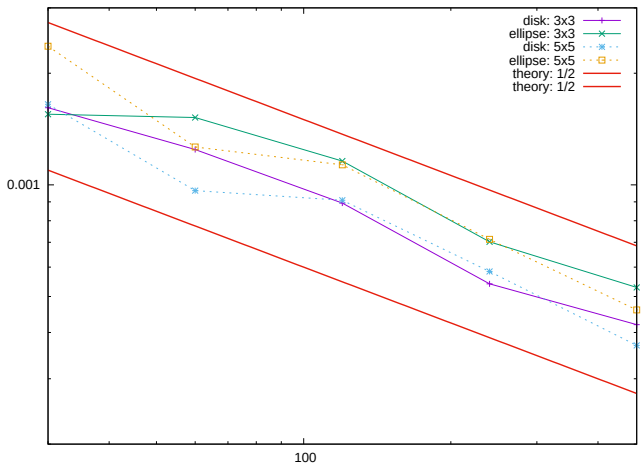

Figure 11: Logarithmic plot of the L^1 error versus the number of cells. The data are from Table 5 and L^1 convergence at a rate $\approx 1/2$ is satisfied asymptotically.

8.2. Shock test problems

Test problems with shocks are often simpler than pure advection, since the interfaces coincide with contact discontinuities which are constrained by the dynamics of the equations (in particular they are often less singular). The time step is calculated

object	block	N_{1D}	L^1	order	order
disk	3×3	30	0.001614		
disk	3×3	60	0.001245	(0.3745)	
disk	3×3	120	0.000893	(0.4794)	(0.4270)
disk	3×3	240	0.000541	(0.7230)	(0.6012)
disk	3×3	480	0.000420	(0.3652)	(0.5441)
ellipse	3×3	30	0.001550		
ellipse	3×3	60	0.001519	(0.0291)	
ellipse	3×3	120	0.001159	(0.3902)	(0.2097)
ellipse	3×3	240	0.000702	(0.7233)	(0.5568)
ellipse	3×3	480	0.000529	(0.4082)	(0.5658)
disk	5×5	30	0.001649		
disk	5×5	60	0.000965	(0.7730)	
disk	5×5	120	0.000909	(0.0862)	(0.4296)
disk	5×5	240	0.000584	(0.6383)	(0.3623)
disk	5×5	480	0.000369	(0.6623)	(0.6503)
ellipse	5×5	30	0.002364		
ellipse	5×5	60	0.001265	(0.9021)	
ellipse	5×5	120	0.001132	(0.1603)	(0.5312)
ellipse	5×5	240	0.000713	(0.6669)	(0.4136)
ellipse	5×5	480	0.000460	(0.6323)	(0.6496)

Table 5: Convergence table. The 3×3 blocks and 5×5 blocks are run with a VOF-ML scheme trained with 5 dense layers, with a large number of epochs (5 000 here) to reach accuracy of the training. For reasons explained in Section 8.3, the finer the mesh, the smaller the relative cost of VOF-ML with respect to the Finite Volume solver. In column 5, the L^1 numerical order of convergence is $o_n = \frac{\log e_{n-1} - \log e_n}{\log 2}$. In column 6, it is $o_n = \frac{\log e_{n-2} - \log e_n}{\log 4}$.

	30	60	120	240	480	order
Upwind	0.6633	0.4895	0.3565	0.2551	0.1818	≈ 0.5
SLIC/Downwind	0.0527	0.0352	0.0195	0.0108	0.0051	≈ 1

Table 6: Reference L^1 norm of the error for the ellipse test case with the Upwind scheme and with SLIC/Downwind scheme. The best error with the SLIC/Downwind scheme on the finest mesh is still much greater than the error of the VOF-ML scheme (in Table 5) on the coarsest mesh.

such that $\max(|u_x|, |u_y|, c)\frac{\Delta t}{\Delta x} \leqslant 1$. For the last test case where the sound speed takes high values, then the Courant number for advection is small. For such simple shock problems, the SLIC/Downwind scheme already provides a satisfactory accuracy. Any dataset made only with arcs of circles and with fine training is sufficient. Here the interfaces are smooth curves. We used 5×5 blocks. These problems also contribute to the validation of the VOF-ML flux for a non-trivial velocity field that contains some degree of vorticity beyond pure translation or pure solid body rotation.

8.2.1. 1D shock tube problem

We consider a symmetric 1D shock tube problem computed with the 2D code. The initial data come from the Sod shock tube problem but with $\gamma_1 \neq \gamma_2$

$$0.3 < x < 0.7 : \quad a_1 = 1,\ a_2 = 0,\ \rho_1 = 1,\ p_1 = 1,\ u_x = 0,\ u_y = 0,\ \gamma_1 = 1.6,$$

$$x < 0.3 \text{ or } 0.7 < x : \quad a_1 = 0,\ a_2 = 1,\ \rho_2 = 0.125,\ p_2 = 0.1,\ u_x = 0,\ u_y = 0,\ \gamma_2 = 1.4.$$

To have good resolution even with a first-order accurate acoustic Lagrangian Riemann solver, we use 1000×5 cells. The Courant number is 0.6. The results calculated with the iso-pressure/iso δQ model are displayed in Figure 12. The steepness of the contact discontinuity is perfectly captured both on the density profile and on the volume fraction profile. In particular the perfect right-left symmetry of the profile is visible on the profile for a_1.

8.2.2. 2D convergent shock problem

We consider a convergent mild shock bi-material test problem where $\gamma_1 = 1.4$, $\gamma_2 = 1.6$. The isobar-isothermal closure model is used with $C_{v1} = C_{v2} = 1$. The initial data are $p_1 = 0.1$, $p_2 = 1$, $\rho_1 = \rho_2 = 1$. The shock is driven by the difference

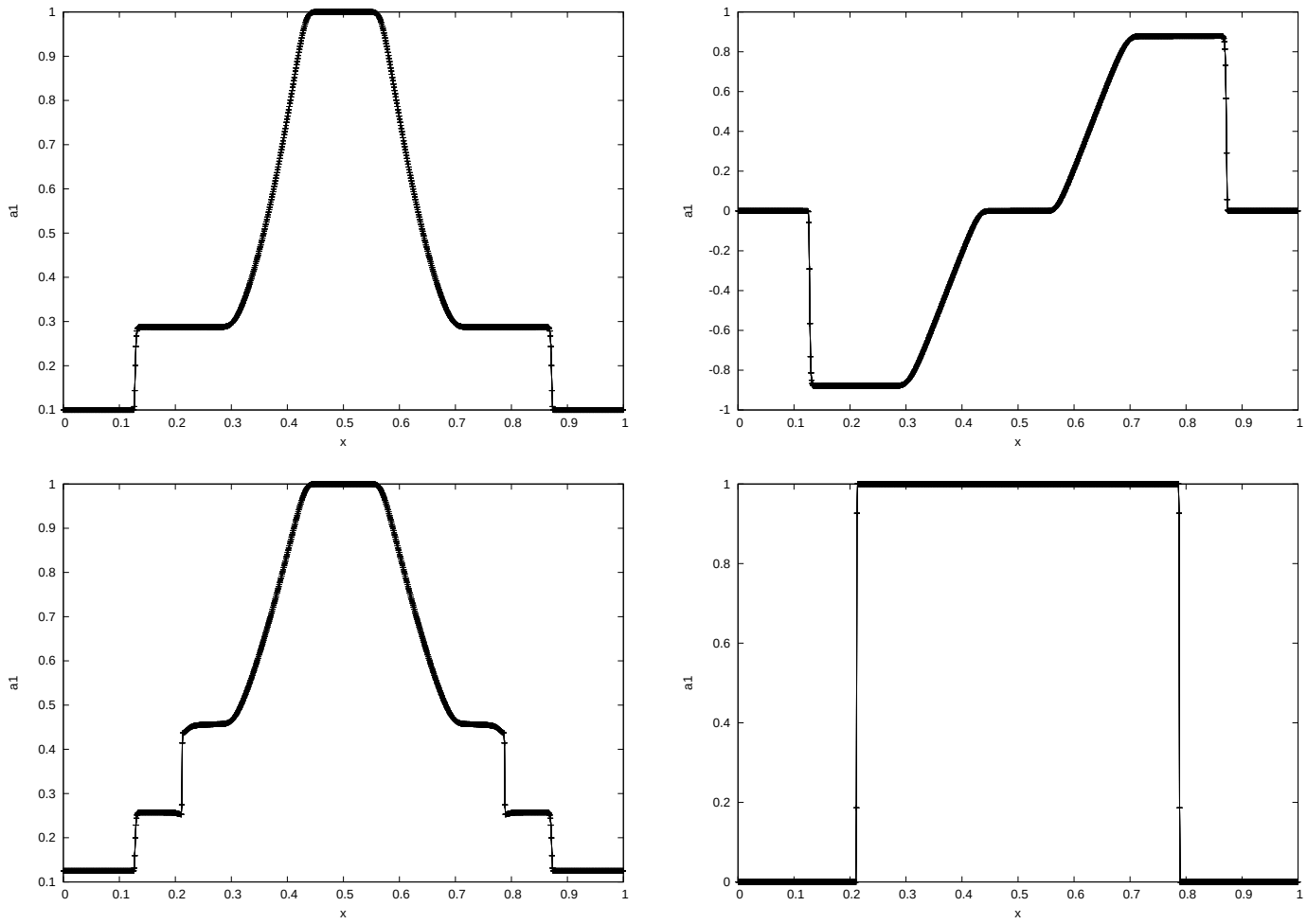

Figure 12: $T = 0.1$. Top left: pressure. Top right velocity. Bottom left: density. Bottom right: volume fraction.

of pressure only. With $\rho_2 > \rho_1$ the shock would be more violent, while with $\rho_2 < \rho_1$ the shock would be weaker. The equality of density also simplifies our implementation of the initial condition. We take wall boundary conditions. The Courant number is 0.4. The interface is initially at $R = 0.3$. The mesh is made of 60×60 cells.

We plot in Figure 13 the volume fraction at time $t = 0$, 0.1, 0.2 and 0.3. We observe a good rendering of the circular nature of the interface which evolves dynamically.

8.2.3. 2D divergent shock problem

We exchange the pressure $p_1 = 1$, $p_2 = 0.1$ with respect to the previous test problem. We plot in Figure 14 the volume fraction at time $t = 0$, 0.1, 0.2 and 0.3. It also shows a good rendering of the steepness of the interface/contact discontinuity.

8.2.4. A Richmyer-Meshkov instability

We finally consider a 2D test problem with the data from [23]. A strong shock hits a one-mode sinusoidal interface which becomes unstable. The domain of calculation is $[0, 3.6] \times [0, 14]$ with an interface $y(x) = 12 + 0.5\cos(2\pi x/3.6)$. The data on both sides of the interface are

$$a_1 = 1,\ a_2 = 0,\ \rho_1 = 2.95,\ p_1 = 50000,\ u_x = 0,\ u_y = 453,\ \gamma_1 = 5/3,$$

$$a_1 = 0,\ a_2 = 1,\ \rho_2 = 1.87,\ p_2 = 50000,\ u_x = 0,\ u_y = 453,\ \gamma_2 = 5/3.$$

The same coefficient $\gamma_1 = \gamma_2 = 5/3$ is used, which allows simple comparisons with purely mono-fluid implementations. The shock is created by a discontinuity at $y = 7$

$$\rho_2^{\text{shock}} = 6.01,\ p_2^{\text{shock}} = 753000,\ u_x^{\text{shock}} = 0,\ u_y^{\text{shock}} = -55.5.$$

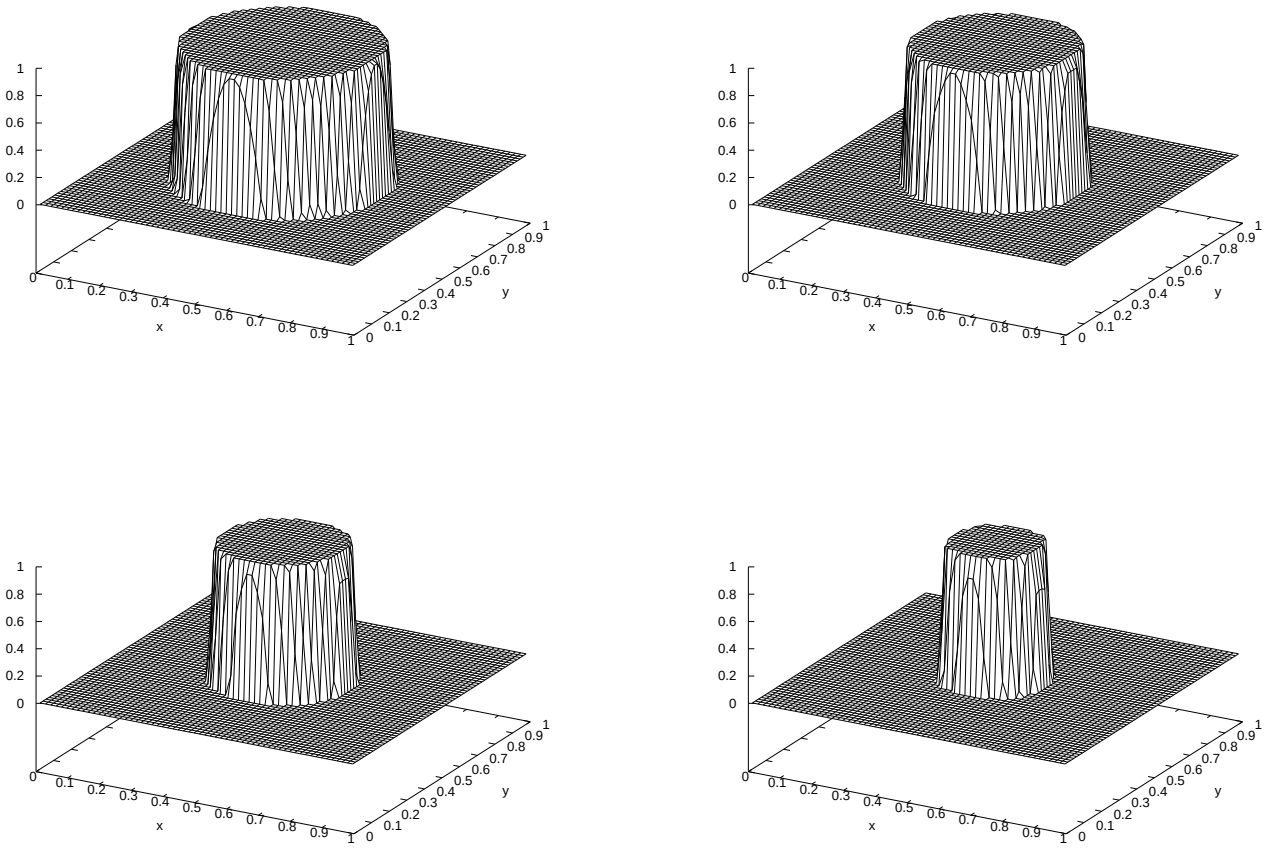

Figure 13: Interface/contact discontinuity in a bi-material convergent test problem. From top left to bottom right, the time is $T = 0$, 0.1, 0.2 and 0.3.

The problem can be run with the iso-pressure/iso-δQ closure model, but it is simpler to run it with the iso-pressure/iso-temperature closure model by taking $C_{v1} = 2$ and $C_{v2} = 3$. With these values, the discontinuity of the density is very well recovered at the interface at the end of the simulation, with the advantage of a simpler and more robust closure model which can be used for comparisons (these important considerations about the choice of a closure model are however not directly related to the core of this work). The physics of this problem is interesting for our algorithms, since the length of the interface increases dynamically during the non linear stage of the instability. The number of square cells is 72×280 to mitigate the low accuracy of the first-order accurate Lagrangian phase.

We represent the volume fraction a_1 at time $t = 0.115$ (CFL=0.5) in Figure 15: on the left part it is with the VOF-ML flux, on the right part it is with the SLIC/Downwind flux. The dynamics is the same even if they are some differences. The structure is narrower with VOF-ML. Actually the quality of results is also very sensitive to the CFL number: if it is too small, extra-diffusion occurs; if it is too large, some oscillations appear. Nevertheless these results shows that the VOF-ML flux has the ability to capture the dynamics of the interface even at the transition between the linear regime and the non linear regime. The symmetry of VOF-ML result is perfectly preserved by our implementation. The number of flagged cells displayed on the bottom-right part of the Figure is clearly related to the length of the interface. Indeed one recognizes the classical strong compression at iteration ≈ 150 followed by a growth of the size of the interface in the linear range. Until the strong compression, the number of flagged cells is approximatively constant as in the right part of Figure 6. Then the number of flagged cells increases linearly, following the physics of the interface. We also plot the velocity in the y direction in the bottom-left part of the figure: since u_y takes different signs, it naturally explains the stretching of the details of the volume fraction.

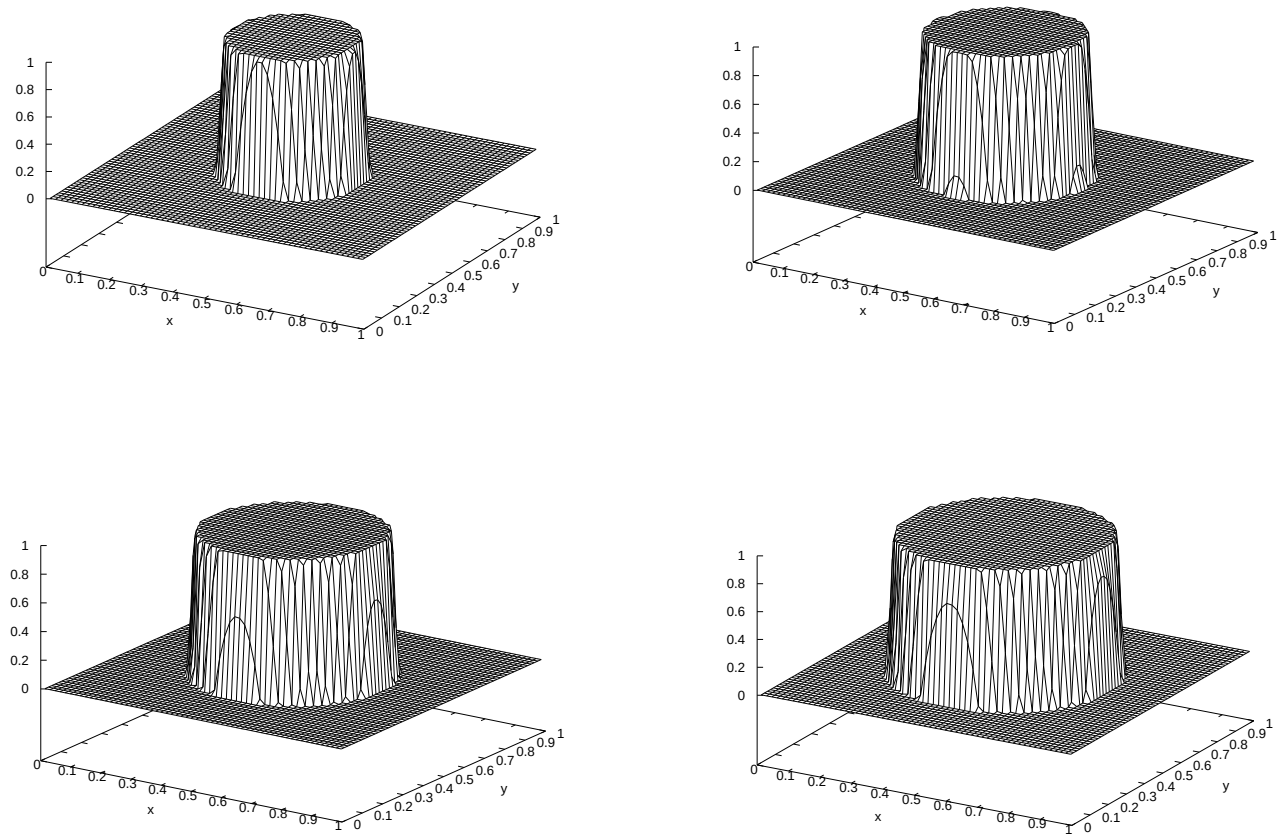

Figure 14: Interface/contact discontinuity in a bi-material divergent test problem. From top left to bottom right, the time is $T = 0$, 0.1, 0.2 and 0.3.

8.3. Computational costs

Finally we give measurements of the CPU costs for the VOF-ML schemes. We remind the reader it is in our tests implemented in C++ using the keras2cpp library. Clearly the CPU cost of one inference (that is one call of the function F) depends on our specific implementation and on the performance of, on the one hand the Euler2D code, and on the other hand the keras2cpp library: the Euler2D code and the keras2cpp library seem to have been implemented with the best practice for such pieces of C++ code.

We begin with a theoretical formula $\text{Time} = \left(N_{\text{cell}} \times \text{T}_{\text{solver}} + N_{\text{flagged}} \times \text{T}_{\text{tensor}}\right) \times N_{\text{cycle}}$. Here T_{solver} is the cost per cell per cycle of Euler2D, T_{tensor} is the average cost per flagged cell per cycle of the VOF-ML schemes (18) when applied to rectangular tensors (16-17), N_{cell} the total number of cells and N_{flagged} the number of flagged cells. The total cost per cell per cycle is

$$\text{T}_{\text{total}} = \text{T}_{\text{solver}} + \sigma \text{T}_{\text{tensor}}, \quad \sigma = \frac{N_{\text{flagged}}}{N_{\text{cell}}} \in [0,1].$$

To determine the values of the various quantities on the right hand side of the equality, we make measurements (in μs per cell per cycle) on a standard Macbook Pro 2.9 GHz Intel Core i7. This is reported in Table 7.

The number of dense layers of the neural network used for measurements is indicated in the second column. For 3×3 blocks, the number of neurons per layer $l \geqslant 2$ is decreasing function of the layer number, equal to $2^{l-2} \times \mathsf{in}$: the number of neurons of the last layer is par. For 5×5 blocks, the number of neurons per layer is the same (that is we use $\mathsf{in} = 10$ and not $\mathsf{in} = 26$), except for the first one for which it is equal to $2^{l-2} \times 26$. The performance of the solver implemented in the Euler2D code is in the third column of the Table: one observes that the cost of the global Finite Volume Lagrange + remap scheme is approximatively 1 μs. It is representative of a reasonably optimized code. The mesh is 60×60 cells and the size of the tensors is $N_{\text{tensor}} = 100$.

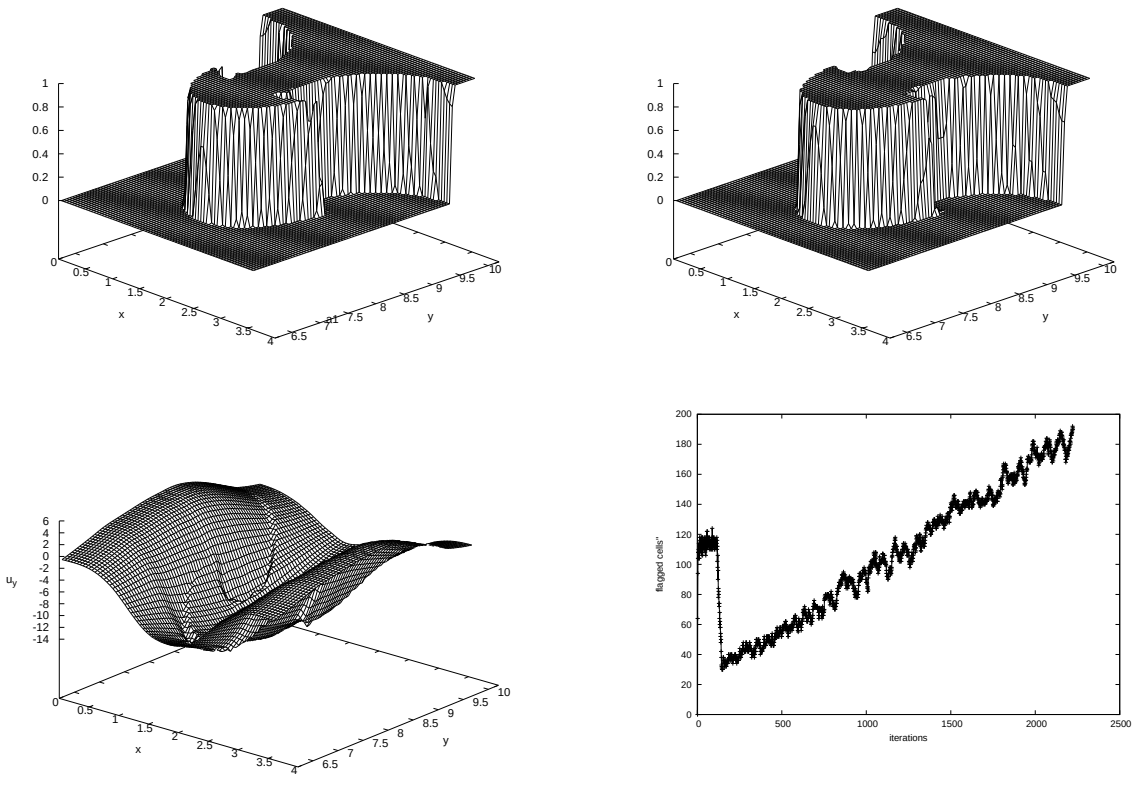

Figure 15: Top: volume fraction a_1 for the Richtmyer-Meshkov test problem in a smaller part of the whole domain $[0, 3.6] \times [6.2, 10.2] \subset [0, 3.6] \times [0, 14]$; top left with the VOF-ML flux; top right with the SLIC/Downwind flux. Bottom left: the velocity field u_y. Bottom right: number of flagged cells versus the number of time steps.

What we observe by comparing the third and fourth columns of Table is that the cost of the VOF-ML scheme implemented with keras2cpp is higher than the cost of the Finite Volume scheme itself. For small test problems such as the ones considered in this work, it is not a real restriction. For more refined grids or for 3D problems, it would become a heavy burden, in particular if one uses a model with higher number of layers.

But fortunately, interface calculations have the benefit of a good scaling. The good control of the number of cells in the neighborhood of the interface, like the ones we show in the numerical results produced in this work, has the effect that the extra cost of the VOF-ML scheme is asymptotically negligible for large number of cells. Indeed in dimension $d = 2$, one has $N_{\text{cell}} = N_{1\text{D}}^2$ and $N_{\text{flagged}} \approx \lambda_{\text{interface}} N_{1\text{D}}$ where $\lambda_{\text{interface}} > 0$ is a constant which depends on the type of interface and on the numerical parameters, but which is independent of $N_{1\text{D}}$. Therefore one can rewrite the theoretical formula as

$$\mathrm{T}_{\text{total}} = \mathrm{T}_{\text{solver}} + \frac{\lambda_{\text{interface}} \times \mathrm{T}_{\text{tensor}}}{N_{1\text{D}}}.$$

Provided $\lambda_{\text{interface}} \times \mathrm{T}_{\text{tensor}}$ is bounded uniformly with respect to $N_{1\text{D}}$, one obtains that

$$\mathrm{T}_{\text{total}} \approx \mathrm{T}_{\text{solver}} \text{ for large } N_{1\text{D}}.$$

It is the result (1) announced in the introduction.

9. Conclusion, perspectives and open problems

In this work we detailed a calculation chain for development of a family of Volume of Fluid-Machine Learning algorithms suitable for bi-material compressible fluid flow calculations on Cartesian grids. We explained a key feature which is the adaptation of the compressible fluid flow solver in order to respect the natural symmetries on a Cartesian grid.

block	layers	T_{solver}	T_{tensor}	rescaled T_{tensor}
3×3	2	0.958	9.78	0.271
3×3	3	0.958	31.9	0.886
3×3	4	0.9807	90.8	2.52
3×3	5	0.977	255	7.08
5×5	2	0.946	15.5	0.432
5×5	3	0.928	44.7	1.24
5×5	4	0.962	113	3.134
5×5	5	1.007	304	8.44

Table 7: Performance measurements in μs. The last column gives the relative cost of the VOF-ML scheme, after multiplication of T_{tensor} by an arbitrary scaling parameter $\sigma = 100/60^2$ which models the relative size of flagged cells near an interface relatively to the total number of cells.

The results show that, using the additivity principle for the construction of the ML dataset, it is possible to incorporate different features of curves (in terms of regularity), and that the final schemes interpolate between the various types of profiles with no noticeable difficulties. A variety of basic test problems common to the compressible community has been computed with satisfactory accuracy. For non smooth interfaces like corners, 5×5 blocks seem to be more accurate than 3×3 blocks, but for smooth interfaces, 5×5 and 3×3 blocks exhibit the same accuracy. To reproduce the results, an accurate model callable with keras2cpp for a 5×5 block trained with a ML dataset with arcs of circles, right and acute angles can be downloaded at [13].

A main issue of the training method is its global accuracy. To compute the numerical values of the volume fractions, one needs a numerical integration procedure adapted to different kind of interfaces (regular, right angle corner, acute angle, non Lipschitz triple point). That is why the Riemann integration with first-order 100×100 uniform sampling is used throughout this work. It yields an accuracy 10^{-2} which is an acceptable scale of error for a compressible calculation: all the rest of the simulation chain follows the same level of accuracy. In principle, it is possible to start with a better accuracy (10^{-3} for example). But it would result in a huge overhead for numerical integration of profiles with only Lipschitz regularity (not to speak of the Trifolium triple point and alike). In this direction, acceleration techniques available for smooth profiles would be a help [2] but with limitations. However it is known that some level of noise is sometimes better than no noise, because it yields robustness. This is why it is not guaranteed that a better global accuracy (10^{-3} for example) would result in any gain for compressible CFD with real data. New developments are necessary to investigate these delicate issues.

The main limitation so far is the overhead of the Volume of Fluid-Machine Learning calculation with respect to the flow solver. However the theoretical scaling shows the overhead is negligible for small mesh size. With dedicated hardware support, it is expected that advanced C++ implementations of Machine Learning functions will reduce this burden. So this limitation is expected to be less stringent in a near future.

The potential of the method for the Volume of Fluid-Machine Learning calculation of interfaces with low regularity seems rich. We have in mind to apply the method to multiphase and multi-material problems (3 phases and more, 2 phases near a wall, 2 materials sliding on a 3rd one, ...) and to tridimensional configurations for which traditional techniques based on regular approximation of interfaces suffer huge limitations.

Among open theoretical problems raised by our numerical experiments, we mention two. A first one is to understand for what reason the dynamical system made of the Volume of Fluid-Machine Learning flux coupled with a more traditional Finite Volume bi-material solver is able to get a good control of the number of flagged cells near the interface. As shown in Section 6.4, this possibility depends crucially of the dataset. Therefore, to get a theoretical answer to this question, it will be necessary to understand the algorithm globally. A second theoretical question is to prove the 1/2 convergence rate for bounded variation data, as it is observed in Section 8.1.4. So far our work is restricted to Cartesian meshes. An open problem of great practical interest is to go beyond such meshes of simple structure. The main difficulty is that adding the local parameters of the mesh in the vector of inputs might result in a very expensive method.

Acknowledgements. The first author thanks Olivier Pironneau for many fruitful discussions during the elaboration stage of this work and acknowledges the support of CEA. The latest version of Euler2D was provided by Frédéric Lagoutière who is kindly thanked.

Appendix A. Interpolation features of ML algorithms

For the completeness of this work, we detail hereafter basic interpolation properties of ML algorithms which justify some of our choices.

Appendix A.1. Interpolation of functions in dimension one

Take a smooth function $f : \mathbb{R} \to \mathbb{R}$ with compact support. Note the Heaviside function $H(x) = 1$ for $x > 0$ and $H(x) = 0$ otherwise. One has the formula $f(x) = \int_{-\infty}^{x} f'(y)dy = \int_{-\infty}^{+\infty} f'(y)H(x-y)dy$. Since $R' = H$ almost everywhere, an integration by part yields $f(x) = \int_{-\infty}^{+\infty} f''(y)R(x-y)dy$. A standard discretization procedure (mesh size $h > 0$) yields the approximation $f_h(x) = \sum_{i\in\mathbb{Z}} f''(ih)R(x-ih)$. This formula shows that linear combinations of offsets of the function R can approximate the function f as accurately as desired. If the function f is not sufficiently smooth (for example if is piece-wise constant), then it is sufficient to regularize it by a factor $\epsilon > 0$ (the function f_ϵ is now smooth with enough derivatives)) to get another approximation

$$f_h^\epsilon(x) = \sum_{i\in\mathbb{Z}} f_\epsilon''(ih)R(x-ih)h. \tag{A.1}$$

This formula can be recovered with one dense hidden layer of neural network, using the rectified linear unit function.

Appendix A.2. Interpolation of functions in higher dimensions

Similar identities hold in any dimension, and it provides a simple approximate version of the Cybenko theorem [8]. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a smooth function with enough derivatives in $L^2(\mathbb{R}^d)$. If it is not the case, it is always possible to regularize it. One has the general formula

$$f(\mathbf{x}) = \int_{\Omega\in S^d}\int_{b\in\mathbb{R}} w(\Omega,b)H(\Omega\cdot\mathbf{x}-b)d\Omega db = \int_{\Omega\in S^d}\int_{b\in\mathbb{R}} \frac{d}{db}w(\Omega,b)R(\Omega\cdot\mathbf{x}-b)d\Omega db$$

where $S^d = \{|\Omega| = 1\} \subset \mathbb{R}^d$ and $w : S^d\times\mathbb{R} \to \mathbb{R}$ is a function to identify. To construct w, differentiate $f(\mathbf{x}) = \int_{\Omega\in S^d}\int_{b<\Omega\cdot\mathbf{x}} w(\Omega,b)d\Omega db$. It yields $\nabla f(\mathbf{x}) = -\int_{\Omega\in S^d} w(\Omega,\Omega\cdot\mathbf{x})\Omega d\Omega$. Take w as the half Fourier transform of another function z, that is $w(\Omega,a) = \int_{\mu>0} e^{i\mu a}z(\Omega,\mu)d\mu$. One gets

$$\nabla f(\mathbf{x}) = -\int_{\Omega\in S^d}\int_{\mu>0} e^{i\mu\Omega\cdot\mathbf{x}}z(\Omega,\mu)\Omega d\mu d\Omega.$$

Make the change of variable $\mathbf{k} = \Omega\mu$ that is $\mu = |\mathbf{k}|$ and $\Omega = \frac{\mathbf{k}}{|\mathbf{k}|}$. The equivalence of the measures writes $d\mathbf{k} = \mu^{d-1}d\mu d\Omega$. One gets

$$\nabla f(\mathbf{x}) = \int_{\mathbf{k}\in\mathbb{R}^d}\left(-\frac{z(\Omega,\mu)}{\mu^d}\right)\mathbf{k}e^{i\mathbf{k}\cdot\mathbf{x}}d\mathbf{k}.$$

By identification with the formula $\nabla f(\mathbf{x}) = \frac{i}{2\pi}\int_{\mathbf{k}\in\mathbb{R}^d}\widehat{f}(\mathbf{k})\mathbf{k}e^{i\mathbf{k}\cdot\mathbf{x}}d\mathbf{k}$, it shows that $z(\Omega,\mu) = -\frac{i}{2\pi}\mu^d\widehat{f}(\mu\Omega)$. With this formula, one can get w, and finally represent f.

Appendix A.3. Recursive construction of "hat" elements

The previous formulas did not make use of the recursive features. Now we state a known result [9] which shows that recursivity helps to reconstruct basic finite elements shape functions. In dimension one, the Finite Element hat function φ_j is

$$\varphi_j(x) = \begin{cases} 0 & \text{for } |x-jh| \geqslant h, \\ \frac{jh-x}{h} & \text{for } (j-1)h \leqslant x \leqslant jh, \\ \frac{x-jh}{h} & \text{for } jh \leqslant x \leqslant (j+1)h. \end{cases}$$

One has $\varphi_j(x) = R\left(\frac{1}{h}R(x-(j-1)h) - \frac{2}{h}R(jh-x)\right)$. So one can also approximate a given function with the well known Finite Element interpolation formula

$$f_{hh}(x) = \sum_{j\in\mathbb{Z}} f(jh)R\left(\frac{1}{h}R(x-(j-1)h) - \frac{2}{h}R(jh-x)\right)$$

which shows the interest of having a recursive ML construction. Generalization of this principle in higher dimensions is left to the reader.

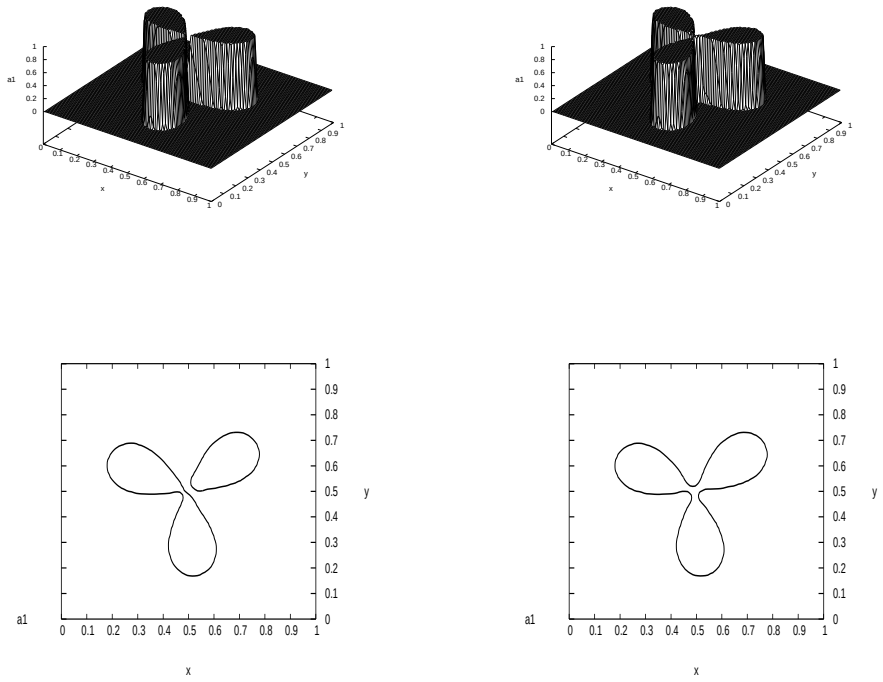

Figure B.16: Value of volume fraction a_1 at time $t = 1$ after one round turn. Left, without a Trifolium dataset. Right, with a Trifolium dataset. The isoline 0.5 is plotted in the bottom row. All results are good, the difference being the connectivity of the triple point.

Appendix B. The Trifolium pure advection test problem

This test problem illustrates the ability of VOF-ML schemes to treat some non Lipschitz profiles which are difficult to conceptualize with standard methods. We consider the pure advection $\mathbf{u} = (1,1)$ of an interface defined in polar coordinates as $r^3 = \frac{1}{27}\cos 3\theta$. It is a modification with more pronounced lobes of the Trifolium rosace $r = \cos 3\theta$. The Trifolium has a triple point where the interface is non Lipshitz (even if it is piecewise Lipshitz). It also has a symmetry with angle $\frac{2\pi}{3}$ which is not tackled by the natural symmetries of a Cartesian grid exposed in Section 6.1.

The results computed with a ML dataset made only of arcs of circle are displayed on the left of Figure B.16, calculated on a mesh with 100×100 cells. They show that the Trifolium is already accurately and robustly recovered. However the structure of the triple point is lost, in particular its connectivity because two lobes stay connected and the third one is disconnected. The $\frac{2\pi}{3}$ perfect symmetry near the triple point is not respected. So one can think of a new somewhat artificial ML dataset with a specific treatment of the triple point of the Trifolium. Following the additivity principle, a new ML dataset is created with Trifolium profiles

$$0 \leqslant \epsilon r^3 \leqslant \cos(3\theta)$$

where ϵ is a scaling parameter which is drawn randomly ($0 \leqslant \epsilon \leqslant 1/20$ in our tests). In the right part of Figure B.16, we represent the new results. We observe that the connectivity of the 3 lobes and the $\frac{2\pi}{3}$ symmetry of the Trifolium are now preserved with the VOF-ML scheme constructed from the full dataset. The Courant number is 0.1.

Appendix C. A test problem with vorticity field beyond pure solid body rotation

The design principle of the VOF-ML flux is based on the incorporation of local pure translations/advections in datasets. However it must be noticed that the method has been validated for test problems which display velocity fields which are quite different from global pure translation. Indeed the dynamically created velocity vector field for the compressible Euler equations is convergent in Section 8.2.2, divergent in Section 8.2.3 and more complex with a vortex (which can be seen as a double shear

in Figure 15) in Section 8.2.4. Inspired by many works in the literature, see [1] and references therein, we add one more test problem with a vorticity field beyond pure solid body rotation and with simple analytical solution.

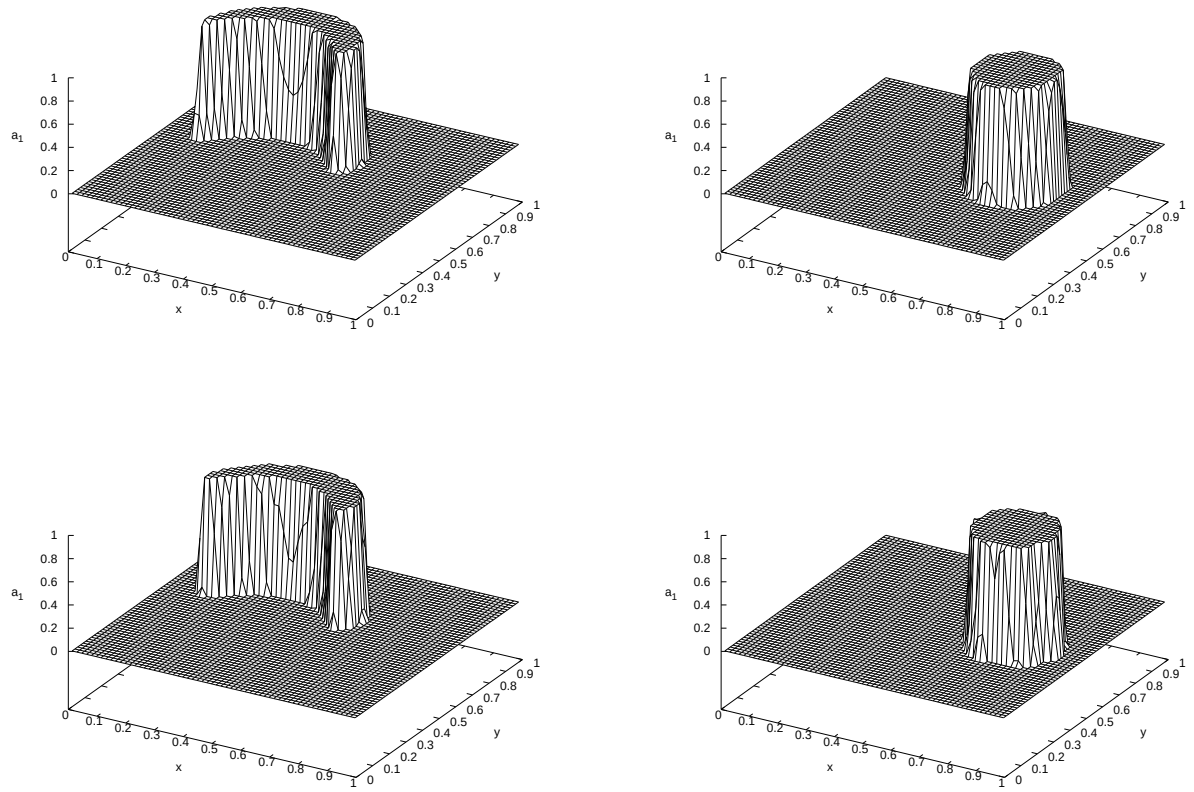

Figure C.17: The volume fraction at mid-time $\frac{1}{2}t_{\text{end}}$ and final time t_{end} (where the initial profile is reconstructed). The velocity field is $\mathbf{u} = \pm 8r\pi(-y,x)$. Top row with the VOF-ML flux. Bottom row with the SLIC/Downwind flux.

The domain is $[0,1]\times[0,1]$. We note $r = \sqrt{|x-0.75|^2+|y-0.5|^2}$. The initial data is $a_1 = 1$ in the disk $\{r < 0.15\}$ and $a_1 = 0$ outside the disk. The free-divergence velocity field $\mathbf{u} = 8\pi\sqrt{x^2+y^2}(-y,x)$ is constant in time for $0 < t < \frac{1}{2}t_{\text{end}}$, and is reversed for $\frac{1}{2}t_{\text{end}} < t < t_{\text{end}}$. Due to the term r, the velocity field can be seen as the composition of a pure body rotation with an additional distortion which has a "slicing effect" on the solutions. The reversal of the velocity field has the effect that the exact solution at $t = t_{\text{end}}$ is equal to the initial one. We take $t_{\text{end}} = 0.6$ and $CFL = 0.2$. The blocks are 5×5 cells. The VOF-ML flux is the one that was carefully trained for the convergence tests of Section 8.1.4. We remind the reader that the code is run with the splitting method $(x,y,x,y,\dots)$ which a priori is not the best one for rotation dominated problems. We first take a 60×60 cells discretization.

The results are displayed in Figure C.17. One observes that the SLIC/Downwind scheme performs already quite well, however the VOF-ML flux provides smoother results, it is visible on the isolines. The L^1 norm of the error at final time is smaller with the VOF-ML flux than it is with the SLIC/Downwind flux, that is $\text{err}_{L^1}^{\text{VOF-ML}} = 0.025 < \text{err}_{L^1}^{\text{SLIC/D.}} = 0.055$. It is similar in quadratic norm, that is $\text{err}_{L^2}^{\text{VOF-ML}} = 0.070 < \text{err}_{L^1}^{\text{SLIC/D.}} = 0.140$. The enhanced accuracy of the VOF-ML flux is a confirmation of the previous results.

Taking a more stringent velocity field $\mathbf{u} = 8\pi(x^2+y^2)(-y,x)$ and a coarser mesh made of 40×40 square cells results in a more severe test problem. It also illustrates the natural limits of VOF methods. The results in Figure C.18 show that the VOF-ML flux and the SLIC/Downwind flux have difficulties to capture the tail of the filament at mid-time $t = \frac{1}{2}t_{\text{end}}$, even if the VOF-ML flux is perhaps better at keeping the connection of the plateau. At final time $t = t_{\text{end}}$, the results are of bad quality with the SLIC/Downwind flux, because the connectivity is lost. On the other hand the VOF-ML flux shows some ability to recover the connectivity of the correct profile, even if a corner is visible: our interpretation is that the intermediate profiles are smoother,

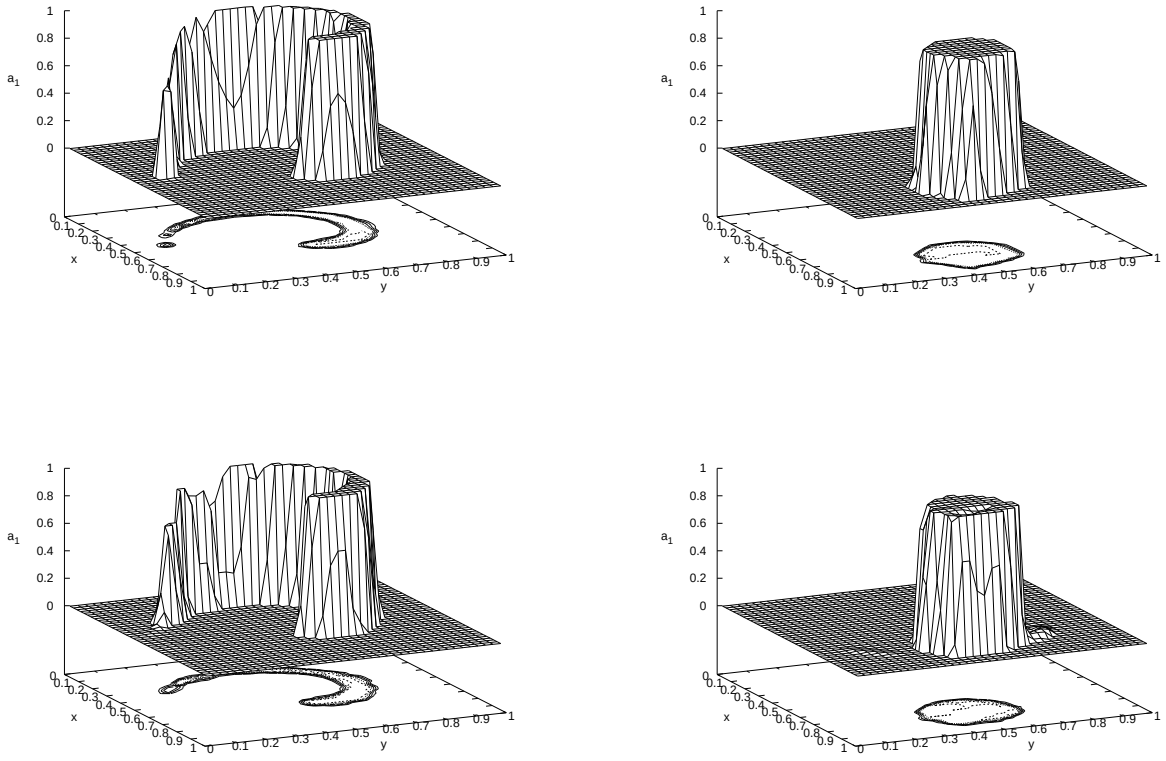

Figure C.18: The volume fraction at mid-time $\frac{1}{2}t_{\text{end}}$ and final time t_{end} (where the initial profile is reconstructed). The velocity field is $\mathbf{u} = \pm 8r^2\pi(-y, x)$. Top row with the VOF-ML flux. Bottom row with the SLIC/Downwind flux.

which is ultimately what is important for the quality at final time. With filament profiles in the dataset, it is possible that the results would be better (this is a completely open question).

References

[1] H.T. Ahn and M. Shashkov, Multi-material interface reconstruction on generalized polyhedral meshes, J. Comput. Phys. 226, no. 2, 2096–2132, 2007.

[2] W. Aniszewski, T. Arrufat, M. Crialesi-Esposito, S. Dabiri, D. Fuster, Y. Ling, J. Lu, L. Malan, S. Pal, R. Scardovelli, G. Tryggvason, P. Yecko and S. Zaleski, PArallel, Robust, Interface Simulator (PARIS), `https://hal.sorbonne-universite.fr/hal-02112617/document`, Hal preprint server, 2019.

[3] S. Basting, A. Quaini, S. Canic and R. Glowinski, Extended ALE Method for fluid-structure interaction problems with large structural displacements, Journal of Computational Physics, Volume 331, 312-336, 2017.

[4] D.J. Benson, Computational methods in Lagrangian and Eulerian hydrocodes, CMAME, 99, 235-394, 1992.

[5] L. Boilevin-Kayl, M. A. Fernandez, J.-F. Gerbeau, A loosely coupled scheme for fictitious domain approximations of fluid-structure interaction problems with immersed thin-walled structures, SIAM Journal on Scientific Computing, Society for Industrial and Applied Mathematics, 41 (2), 351-374, 2019.

[6] F. Chollet, Deep learning with Python, Manning Publications, 2018.

[7] B. Cockburn, P.-A. Gremaud and J. X. Yang, A priori error estimates for numerical methods for scalar conservation laws. III. Multidimensional flux-splitting monotone schemes on non-Cartesian grids. SIAM J. Numer. Anal. 35, no. 5, 1775–1803, 1998.

[8] G. Cybenko, Approximation by Superpositions of a Sigmoidal Function, Math. Control Signals Systems, 2:303-31, 1989.

[9] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, Nonlinear Approximation and (Deep) ReLU Networks, arxiv 1905.02199v1.

[10] R.B. DeBar, Fundamentals of the KRAKEN code, California Univ., Lawrence Livermore Lab., UCID-17366, 1974.

[11] F. Delarue and F. Lagoutière, Probabilistic analysis of the upwind scheme for transport equations. Arch. Ration. Mech. Anal. 199, no. 1, 229–268, 2011.

[12] B. Després, An explicit a priori estimate for a finite volume approximation of linear advection on non-Cartesian grids, SIAM J. Numer. Anal. 42, no. 2, 484–504, 2004.

[13] B. Després web page, LJLL/Sorbonne university, `https://www.ljll.math.upmc.fr/despres/BD_fichiers/VOF.model`, 2019.

[14] B. Després, Numerical methods for Eulerian and Lagrangian conservation laws, Frontiers in Mathematics. Birkhauser/Springer, Cham, 2017.

[15] D. Dobias, Frugally-deep github page, `https://github.com/Dobiasd/frugally-deep`, MIT licence.

[16] D. Fuster and S. Popinet, An all-Mach method for the simulation of bubble dynamics problems in the presence of surface tension, Journal of Computational Physics, 374, 752-768, 2018.

[17] F. Gibou, D. Hydec and R Fedkiw, Sharp interface approaches and deep learning techniques for multiphase flows, Journal of Computational Physics, 380, 442-463, 2019.

[18] E. Godlevski and P.A. Raviart, Numerical Approximation of Hyperbolic Systems of Conservation Laws, Springer, 1996.

[19] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, `http://www.deeplearningbook.org`, 2016.

[20] Keras2cpp github page, `https://github.com/gosha20777/keras2cpp`, MIT licence.

[21] Kerasify github page, `https://github.com/moof2k/kerasify`, MIT licence.

[22] D. P. Kingma and J. Lei Ba, ADAM: a method for stochastic optimization, conference paper, ICLR conference, 2015.

[23] F. Lagoutière and B. Després, Numerical resolution of a two-component compressible fluid model with interfaces: Progress in Computational Fluid Dynamics, 7(6), 295–310, 2007.

[24] R. J. Leveque, Finite Volume Methods for Hyperbolic Problems, Cambridge University Press, 2004.

[25] B. Merlet, L^∞- and L^2-error estimates for a finite volume approximation of linear advection, SIAM J. Numer. Anal. 46, no. 1, 124–150, 2007/08.

[26] W. F. Noh and P. Woodward, SLIC (Simple Line Interface Calculation), Lecture Notes in Physics, vol. 59, 330-340, 1976.

[27] J. E. Pilliod Jr. and E. G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces, Journal of Computational Physics 199, 465-502, 2004.

[28] Y. Qi, J. Lu, R. Scardovelli, S. Zaleski and G. Tryggvason, Computing curvature for volume of fluid methods using machine learning, Journal of Computational Physics 377, 155-161, 2019.

[29] M. Raissi and G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics, 357, Pages 125-141, 2018.

[30] D. Ray and J. S. Hesthaven, An artificial neural network as a troubled-cell indicator, Journal of Computational Physics, 367, 166-191, 2018.

[31] W.J. Rider and D.B. Kothe, Reconstructing volume tracking, Journal of Computational Physics, 141, 112-152, 1998.

[32] R. Scardovelli and S. Zaleski, Direct numerical simulation of free-surface and interfacial flow, Annu. Rev. Fluid Mech., 31:567?603, 1999.

[33] Tensorflow, `https://www.tensorflow.org`.

[34] Q. Wang, J. S. Hesthaven and D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, Journal of Computational Physics, 384, 289-307, 2019.

[35] R. Ward, X. Wu and L. Boutou, AdaGrad stepsizes: sharp convergence over nonconvex landscapes, from any initialization, arxiv 1806.01811.

[36] D.L. Youngs, Time-dependent multi-material flow with large fluid distortion, Numerical methods for fluid dynamics, Morton and Baines Editors, 273-285, 1982.

[37] S. Zaleski, private communication, 2019.