

CERF : Computation, ERosion and Flow V1.0

GOLAY Frédéric

Table of contents

Introduction	3
License	4
1 Modeling	5
1.1 The finite-volume method	5
1.1.1 Principle of the finite volume method	5
1.1.2 Time discretization	7
1.1.3 Spatial discretization	8
1.2 Saint-Venant model	13
1.2.1 Governing equations	13
1.2.2 Handling bathymetry	14
1.2.3 Refinement criterion	15
1.2.4 Handling friction	15
1.2.5 Riemann solver for the Saint-Venant model	16
1.3 Two-fluid flow model	22
1.3.1 Governing equations	22
1.3.2 Refinement criterion	24
1.3.3 Free-surface sharpening	25
1.3.4 The “ <i>rigid fluid</i> ” concept	25
1.3.5 Riemann solver for the two-fluid model	26
2 User Guide	33
2.1 Software installation	33
2.1.1 Prerequisites	33
2.1.2 Installation	33
2.1.3 Running a computation	35
2.2 The configuration file *.inp	37
2.2.1 General overview	37
2.2.2 Defining the physical parameters: PHYS	37
2.2.3 Defining the master mesh: MAME	39
2.2.4 Defining the initial conditions: INIT	40
2.2.5 Defining bathymetry and friction: BATH	41
2.2.6 Defining the boundary conditions: COND	41
2.2.7 Defining the mesh: MESH	43

2.2.8	Defining “obstacles”: OBST	44
2.2.9	Defining the numerical parameters: NUME	46
2.3	The CERF source files	47
2.3.1	Structure and modules	47
2.3.2	Dependency graph of the main program	48
2.3.3	Dependency graph of the pre-processing program	48
2.3.4	Dependency graph of the mesh-modification program	48
3	Documented examples	52
3.1	Examples with the Saint-Venant model	52
3.1.1	Riemann problem	52
3.1.2	Hydraulic jump	60
3.1.3	Friction, McDonald test case	73
3.2	Examples with the Eulerian two-fluid model	78
3.2.1	Riemann problem	78
3.2.2	Dam break	85
3.2.3	A cylinder falling onto a body of water	90
	Références	101

Introduction

This document is a user guide for **CERF**. **CERF** is a finite-volume computational code dedicated to the simulation of free-surface flows. **CERF** is a research code developed at the [Institut de Mathématiques de Toulon](#) under the supervision of [Frédéric Golay](#). **CERF** solves the Saint-Venant equations and the isothermal two-fluid Euler equations. These hyperbolic systems of equations allow the use of an explicit and parallel time integration scheme. The data structure of **CERF** is based on a BB-AMR (Block Based Adaptive Mesh Refinement) approach, which makes it possible to handle meshes that adapt in both time and space. **CERF** is a parallel computational code that uses the MPI library for communication between processes. **CERF** is written in Fortran 90. In its V1.0 version, the **CERF** mesh consists exclusively of hexahedra.

This document is organized as follows. In the [first chapter](#), we present the physical models used in **CERF** to simulate free-surface flows. Erosion models are currently under development. We also describe the numerical schemes used. The [second chapter](#) is devoted to the installation and use of **CERF**, as well as the description of its input and output files. We explain how to configure a simulation and how to visualize the results. Finally, the [third chapter](#) consists of a set of documented examples.

! Warning

My command of the English language is, shall we say... rough. At best, it's understandable "Globish." As a result, this documentation has been automatically translated. I apologize in advance if any inaccuracies—which I may not be able to detect—appear from time to time.

This document was written with [Quarto](#).

This software is governed by the [CeCILL-B](#) license, subject to French law and complying with the principles of free software distribution.

License

Copyright [Institut de Mathématiques de Toulon](#), (2025).

frederic.golay@univ-tln.fr

This software is a finite-volume computational code dedicated to the simulation of free-surface flows.

This software is governed by the [CeCILL-B](#) license, subject to French law and complying with the principles of free software distribution. You may use, modify and/or redistribute this program under the terms of the [CeCILL-B](#) license as distributed by CEA, CNRS and INRIA at the following URL “https://cecill.info/licences/Licence_CeCILL-B_V1-en.html”.

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty. For the same reason, only limited liability is incurred by the software’s author, the holder of the economic rights, and the successive licensors.

In this respect, the user’s attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user, given its free-software nature, which may make it complex to manipulate, and which therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software’s suitability for their needs under conditions that ensure the security of their systems and/or data, and, more generally, to use and operate it under the same conditions of security.

The fact that you are presently reading this means that you have had knowledge of the [CeCILL-B] license and that you accept its terms.

1 Modeling

This chapter describes the physical models used in **CERF** to simulate flows (without erosion in the version V1.0), as well as how these models are implemented in the code. It is divided into three sections. The first section presents the finite-volume method in the general case. The second section describes the Saint-Venant shallow-water flow model. The third section presents the Eulerian two-fluid flow model used in **CERF**.

1.1 The finite-volume method

The finite volume method is a numerical method used to solve partial differential equations. This method is used in **CERF** to solve the Saint-Venant equations and the two-fluid flow equations.

1.1.1 Principle of the finite volume method

In this section, we briefly recall the finite volume approximation, in dimension n_{dim} , of a system of conservation equations of the form:

$$\frac{\partial \vec{w}}{\partial t} + \nabla \cdot \vec{F}(\vec{w}) = \vec{S}(\vec{w}) \quad (1.1)$$

where $\vec{w}(\vec{x}, t)$ is the vector of conservative variables of size n_{dof} depending on space and time, \vec{F} the flux matrix of size $n_{dof} \times n_{dim}$, \vec{S} the vector of source terms of size n_{dof} , and $\nabla \cdot$ the *divergence* operator.

The system is solved in $\mathbb{R}^{n_{dim}}$. The computational domain $\Omega \subset \mathbb{R}^{n_{dim}}$ is divided into a set of control volumes, also called cells, $\Omega = \cup_k C_k$. We begin by integrating Equation 1.1 over a cell C_k using Green's formula:

$$\begin{aligned} \int_{C_k} \frac{\partial \vec{w}}{\partial t} d\Omega + \int_{C_k} \nabla \cdot \vec{F} d\Omega &= \int_{C_k} \vec{S}(\vec{w}) d\Omega \\ \int_{C_k} \frac{\partial \vec{w}}{\partial t} d\Omega + \sum_a \int_{\partial C_k/a} \vec{F} \cdot \vec{n}_{k/a} d\partial\Omega &= \int_{C_k} \vec{S}(\vec{w}) d\Omega \end{aligned} \quad (1.2)$$

where $\vec{n}_{k/a}$ denotes the unit normal vector on the boundary $\partial C_{k/a}$ between cell C_k and the neighboring cell C_a . The summation is carried out over all the neighboring cells of C_k .

The finite volume method then consists in approximating the solution by piecewise constant functions per cell, such that by setting

$$\vec{w}_k(t) \approx \frac{1}{|C_k|} \int_{C_k} \vec{w}(\vec{x}, t) d\Omega$$

where $|C_k|$ represents the volume of cell C_k , we approximate Equation 1.2 by

$$|C_k| \frac{\partial \vec{w}_k(t)}{\partial t} + \sum_a |\partial C_{k/a}| \hat{\vec{F}}(\vec{w}_k(t), \vec{w}_a(t), \vec{n}_{k/a}) = |C_k| S_k(t) \quad (1.3)$$

where $S_k(t) \approx \frac{1}{|C_k|} \int_{C_k} S(\vec{x}, t) d\Omega$ represents the average value of the source term vector over the cell, $|\partial C_{k/a}|$ the surface area of the face between cells C_k and C_a , and $\hat{\vec{F}}$ an approximation of the flux.

Since the unknowns \vec{w} are not defined at the interface between cells, the flux cannot be computed directly. It is therefore approximated by a numerical flux depending on the interface and on the values of the variables in the adjacent cells. We denote:

$$\vec{F}_{k/a}(t) = \hat{\vec{F}}(\vec{w}_k(t), \vec{w}_a(t), \vec{n}_{k/a}) \approx \frac{1}{|\partial C_{k/a}|} \int_{\partial C_{k/a}} \vec{\bar{F}} \cdot \vec{n}_{k/a} ds$$

There are many possible approaches: Rusanov, HLLC, etc. In **CERF**, the numerical flux is computed using Godunov's method, based on the exact solution of the Riemann problem at the interface k/a (for more details, see Section 1.2.5 and Section 1.3.5).

Second-order accuracy in space can be achieved using the MUSCL (Monotone Upstream Scheme for Conservation Laws) method to reconstruct the primitive variables at the interfaces. From the values of the variables on the neighboring cells, the gradient of the primitive variables is estimated. Let w_{p_i} be the i -th component of the primitive variables and $w_{p_i}^k$ its value at the center of the cell; the gradient over cell C_k can be estimated as:

$$|C_k| \widetilde{\nabla} w_{p_i} = \int_{C_k} \nabla w_{p_i} d\Omega = \sum_a \int_{\partial C_{k/a}} w_{p_i} \vec{n}_{k/a} d\Omega = \sum_a |\partial C_{k/a}| \frac{1}{2} (w_{p_i}^k + w_{p_i}^a) \vec{n}_{k/a}$$

and the primitive variables of cell C_k with center K can then be reconstructed at the centers F of the cell faces such that:

$$w_{p_i}^F = w_{p_i}^k + \widetilde{\nabla} w_{p_i} \cdot \vec{KF}$$

To achieve second-order accuracy, $\hat{\vec{F}}(\vec{w}_k(t), \vec{w}_a(t), \vec{n}_{k/a})$ is then replaced by $\hat{\vec{F}}(\vec{w}_k^F(t), \vec{w}_a^F(t), \vec{n}_{k/a})$

i Note

In general, a limiter must be combined with the MUSCL method. In **CERF**, the Barth-Jespersen limiter is used: $Min(w_{p_F}, w_{p_F}) \leq w_{p_i}^F \leq Max(w_{p_F}, w_{p_F})$.

1.1.2 Time discretization

To facilitate the parallelization of the code, **CERF** uses only explicit schemes: the first- or second-order Runge-Kutta scheme. The time interval $[0, T]$ is discretized such that $t_{n+1} = t_n + \delta t_n$, where δt_n represents the time step at time t_n . The first-order Runge-Kutta method, also known as the first-order explicit Euler method, consists in rewriting Equation 1.3 at time t_n by approximating the time derivative as a finite difference, such that:

$$|C_k| \frac{\vec{w}_k(t_{n+1}) - \vec{w}_k(t_n)}{\delta t_n} + \sum_a |\partial C_{k/a}| \vec{F}_{k/a}(t_n) = |C_k| \vec{S}_k(t_n)$$

By denoting $\vec{w}_k(t_n) = \vec{w}_k^n$, the new value of the conservative variables at time t_{n+1} is thus computed explicitly:

$$\vec{w}_k^{n+1} = \vec{w}_k^n + \frac{\delta t_n}{|C_k|} \sum_a |\partial C_{k/a}| \vec{F}_{k/a}^n - \delta t_n \vec{S}_k^n$$

The second-order Runge-Kutta method consists in computing the solution at time t_{n+1} in two steps. The first step consists in computing the solution at time $t_{n+\frac{1}{2}}$ using the first-order explicit Euler scheme. The second step consists in computing the solution at time t_{n+1} using the first-order explicit Euler scheme together with the solution obtained at time $t_{n+\frac{1}{2}}$.

$$\begin{aligned} \vec{w}_k^{n+\frac{1}{2}} &= \vec{w}_k^n + \frac{\delta t_n}{2|C_k|} \sum_a |\partial C_{k/a}| \vec{F}_{k/a}^n - \frac{\delta t_n}{2} \vec{S}_k^n \\ \vec{w}_k^{n+1} &= \vec{w}_k^n + \frac{\delta t_n}{|C_k|} \sum_a |\partial C_{k/a}| \vec{F}_{k/a}^{n+\frac{1}{2}} - \delta t_n \vec{S}_k^{n+\frac{1}{2}} \end{aligned}$$

! Important 1: Courant-Friedrichs-Lewy stability condition

Explicit methods are simple to implement, but their stability is constrained by the Courant-Friedrichs-Lewy (CFL) condition. The CFL condition is a necessary condition to guarantee the stability of the numerical scheme. This condition depends on the wave propagation speed in the medium and on the size of the cells. In practice, it is determined for each hyperbolic model considered. In **CERF**, the cell size d_k is determined by the

ratio between the volume of the cell and the surface area of its boundary:

$$d_k = \frac{|C_k|}{\sum_a |\partial C_{k/a}|}$$

In the 1D or 2D case, symmetry faces are not taken into account when computing the boundary surface area. The CFL condition is then given by:

$$\delta t_n \leq \alpha_{CFL} \frac{\min_k d_k}{\max_k \lambda_k}$$

where α_{CFL} is a coefficient to be defined by the user and λ_k is the wave propagation speed in cell C_k , depending on the model used (Note 1 for Saint-Venant and Note 2 for two-fluid flow).

1.1.3 Spatial discretization

! Important

In its V1.0 version, **CERF** uses **exclusively** hexahedral cells and therefore quadrangular faces! Generalization is currently under development.

i Note

CERF allows the use of dynamic meshes, meaning that the mesh can be regularly refined or coarsened (**AMR**: Adaptive Mesh Refinement).

CERF is a natively three-dimensional code. In its initial version, **CERF** uses only hexahedral cells and therefore quadrangular faces. Cells and faces are not necessarily regular. Nevertheless, as illustrated in Figure 1.1, two-dimensional simulations can be performed by creating a mesh with a **single** cell in the z direction and imposing “**mirror**”-type conditions on the faces in the x, y plane. Similarly, one-dimensional simulations can be performed by creating a mesh with a **single** cell in the y and z directions and imposing “*mirror*”-type conditions on the faces in the x, z and x, y planes.

1.1.3.1 Meshing strategy

CERF provides mesh refinement and coarsening tools (AMR: Adaptive Mesh Refinement). To avoid penalizing computation time, the mesh is not modified at every time step, but at instants determined by the user. This approach makes it possible to maintain good mesh quality while

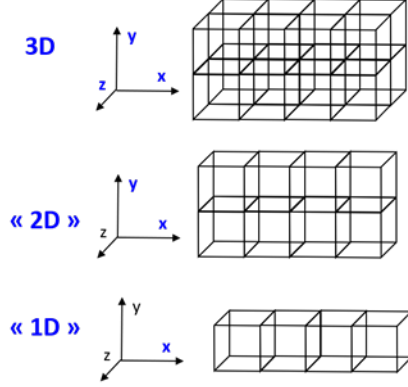


Figure 1.1: Three-, two-, and one-dimensional meshes with hexahedral cells

limiting computational cost. This method is referred to as BB-AMR (Block Based Adaptive Mesh Refinement). The various steps of the meshing strategy used in **CERF** are described below.

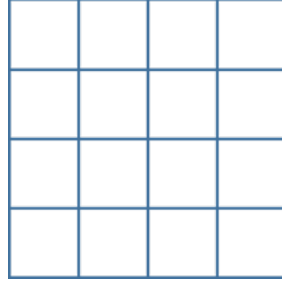


Figure 1.2: Initial conforming mesh: the “*Master Mesh*”.

Step 1: For the sake of simplicity, the mesh shown in Figure 1.2 consists of regular quadrangles, which can represent a hexahedral mesh viewed along z with a single cell in the z direction. As illustrated in Figure 1.2, we start from an initial **conforming** mesh made up solely of hexahedra of arbitrary shape. In a simple case, the mesh can be created by **CERF**; otherwise it can be imported in GMSH V2.2 format. This mesh is referred to as the **master mesh**. It is essential that the mesh be conforming in order to easily define the faces connecting the cells. The cells of this initial mesh are referred to as **blocks**.

Step 2: The blocks of the initial mesh are ordered by numbering them according to the *Z-order* or *Morton code* (Figure 1.3). With this approach, known as a *Space Filling Curve* (Section 1.1.3.2), the numbering makes it possible to traverse the blocks, even in three dimensions, following a numbering such that successive blocks in the numbering are, most of the time, neighbors in space.

Step 3: The refinement level of the blocks is then defined, as illustrated in Figure 1.4. Initially,

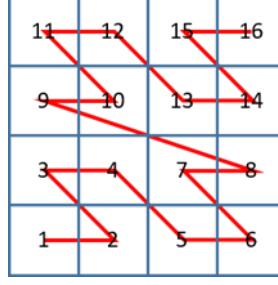


Figure 1.3: Numbering of the initial mesh according to *Morton codes*.

0	0	0	0
0	1	1	0
1	2	2	1
0	1	1	0

Figure 1.4: Definition of the block refinement level.

the refinement level is defined by the user. During the computation, the refinement level is defined by a criterion Section 1.1.3.3. As verified in Ersoy, Golay, and Yushchenko (2013) and Thomas Altazin and Yushchenko (2016), the difference in refinement level between two blocks must not exceed 1. During refinement, the block is divided by 2 in each direction, for each level required. Thus, denoting by Niv the required refinement level (where $Niv = 0$ represents a block that is not refined) and $dim = 1, 2, 3$ the dimension of the problem, the number of cells in a block will be: $2^{dim \times Niv}$. In the example shown in Figure 1.4, block 1 is not refined, block 2 is refined by 1 level, and block 4 by 2 levels. This leads, as illustrated in Figure 1.5 for the case $n_{dim} = 2$, to 1 cell in block 1, 4 cells in block 2, and 16 cells in block 4.

1	1	1	1
1	4	4	1
4	16	16	4
1	4	4	1

Figure 1.5: Decomposition of the mesh into domains.

Step 4: Since the time integration schemes in **CERF** are explicit, it is straightforward to

parallelize the solution algorithm. To do this, the mesh must be *split* into several domains, ideally with a balanced number of cells and minimal interfaces, in order to minimize data exchange between domains. Many algorithms exist, some of them quite complex, that perform this operation efficiently. In **CERF**, we chose a compromise between simplicity and efficiency. We opted for a decomposition driven by Morton codes. Knowing the number of cells per block, the blocks are traversed following the Morton numbering and assigned to a domain. The number of cells per domain is kept balanced. As illustrated in Figure 1.5, the initial mesh is split into 3 domains made up of 25, 21, and 18 cells. Each computational domain is then assigned to a computing process.

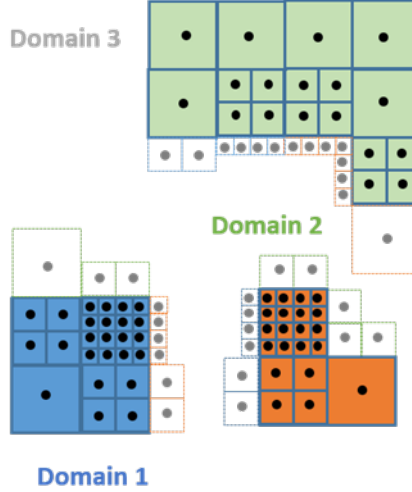


Figure 1.6: Mesh generation.

Step 5: The final step consists in generating the mesh. To exchange information between domains, **CERF** creates ghost cells. These cells are copies of the neighboring cells of the domain. They are used to compute the numerical fluxes at the interfaces between domains. As illustrated in Figure 1.6, ghost cells are shown in gray. Note that the mesh generated in this way is **not conforming**! For example, the first cell at the bottom left of domain 1 in Figure 1.6 is surrounded by 6 faces (in the x, y plane). The refinement of the blocks defining the mesh naturally depends on the dimension of the problem being treated (Figure 1.7). Finally, if the mesh is modified during the computation, it is necessary to project the conservative variables onto the new mesh. In a remeshing operation, the level can only increase or decrease by at most 1, so that the projection step is relatively simple.

1.1.3.2 Morton code

Space-filling curves are curves that make it possible to traverse a higher-dimensional space. First introduced by [Peano](#), then by [Hilbert](#) or [Lebesgue](#), they are used for example to index

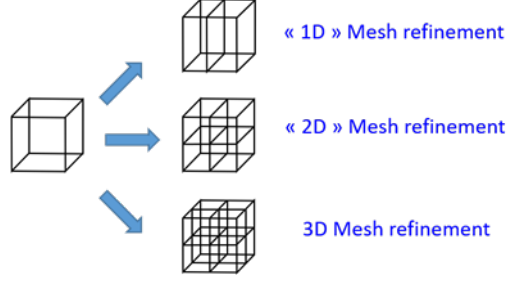


Figure 1.7: Refinement depending on the dimension of the problem.

data, and in particular in the construction of *quadtrees* or *octrees*. In **CERF**, numbering by *Morton code*, also known as the *Z-order curve*, is used. This numbering makes it possible to index a set of data (points, cells, faces, etc.) such that data that are neighbors in space are also neighbors in the indexing. This property is used in **CERF** to split the mesh into computational domains by indexing the blocks of the master mesh. Consider a point with coordinates (x, y, z) in a three-dimensional space. To index this point, the coordinates are first converted into positive integers. To do this, the smallest coordinates of the study domain $(x_{min}, y_{min}, z_{min})$ are defined, along with a distance d_{morton} defining the grid of the domain, for example one tenth of the size of the smallest cell in the domain. The integer coordinates of the point (ix, iy, iz) are then defined by:

$$ix = \frac{x - x_{min}}{d_{morton}}, iy = \frac{y - y_{min}}{d_{morton}}, iz = \frac{z - z_{min}}{d_{morton}}$$

Morton numbering then consists in converting these integer coordinates into base 2 (ix_2, iy_2, iz_2) , then interleaving the bits of each coordinate to form a single base-2 integer M_2 . Finally, converting back to base 10, the Morton index M of the point (x, y, z) is obtained.

Example 1.1. $ix = 2, iy = 5, iz = 0$.

We then have $ix_2 = 010$, $iy_2 = 101$, and $iz_2 = 000$.

Interleaving the bits gives $M_2 = 010100010$, and in base 10, $M = 162$.

i Note

All of **CERF**'s indexing tools can be found in the module `~/sources/UTI/mod_zorder.f90`.

1.1.3.3 Refinement criterion

In general, a refinement criterion is defined for each physical model considered. This criterion is often derived from physical considerations based, for example, on the gradient of one of the primitive variables, or, more relevantly and more elaborately, on an *a posteriori* error estimate

or the construction of an error indicator. In the case of hyperbolic problems, there exists a more general criterion derived from the work of Croisille (1990). This criterion is based on the numerical production of entropy. It was used by Frédéric Golay (2009), Ersoy, Golay, and Yushchenko (2013), and studied at length by Puppo (2004). The entropy inequality of Lax (1971) allows us to state that for any hyperbolic system of conservation laws of the form Equation 1.1, there exists a mathematical entropy $s(\vec{w})$ and an entropy flux $\vec{\psi}_s(\vec{w})$ such that:

$$\frac{\partial s(\vec{w})}{\partial t} + \nabla \cdot \vec{\psi}_s(\vec{w}) \leq 0 \quad (1.4)$$

where the entropy flux $\vec{\psi}_s(\vec{w})$ must satisfy a compatibility equation $\nabla_w \psi_s = \nabla_w \bar{F} \cdot \nabla_w s$. For smooth solutions, Equation 1.4 is an equality. In the case of a shock, it is a strict inequality. However, in numerical simulations, the numerical production of entropy is observed not to be zero! It turns out that the numerical production of entropy is a very good error indicator, except at shocks, where it tends to infinity. We therefore define p_k^s , the numerical entropy production over cell C_k , as:

$$p_k^s = \frac{1}{|C_k|} \int_{C_k} \left(\frac{\partial s(\vec{w})}{\partial t} + \nabla \cdot \vec{\psi}_s(\vec{w}) \right) d\Omega$$

The refinement criterion C_b is defined per block as the ratio of the numerical entropy production in the block to the total numerical entropy production over the domain.

$$C_b = \frac{\int_{Bloc} \left(\frac{\partial s(\vec{w})}{\partial t} + \nabla \cdot \vec{\psi}_s(\vec{w}) \right) d\Omega}{\int_{\Omega} \left(\frac{\partial s(\vec{w})}{\partial t} + \nabla \cdot \vec{\psi}_s(\vec{w}) \right) d\Omega}$$

In **CERF**, the refinement criterion is defined per block. If the criterion exceeds a threshold value α_{max} , the block is refined. If the criterion is below a threshold value α_{min} , the block is coarsened. The threshold values are defined by the user.

1.2 Saint-Venant model

1.2.1 Governing equations

The shallow-water flow model, or Saint-Venant model, is a simplified flow model. It is derived from the Navier-Stokes equations by neglecting the viscosity terms, assuming a hydrostatic pressure distribution, and assuming that the water depth is much smaller than the characteristic length of the flow. The original Barré de Saint-Venant model is a 1D model Saint-Venant (1871); the derivation of the model is explained, for example, by Marche (2007) or Poussel (2024). Development of the model in **CERF** was initiated by Pons (2018). In a frame of reference (O, x, y, z) , we consider a shallow-water flow over a domain where the bathymetry

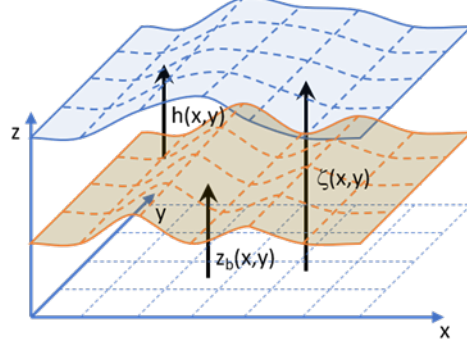


Figure 1.8: Notation of the shallow-water flow model

is defined by $z_b(x, y)$, with a water depth $h(x, y)$, so that the free-surface level is defined by $\zeta = h + z_b$, as illustrated in Figure 1.8.

The Saint-Venant equations are as follows:

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\vec{u}) = 0 \quad (1.5)$$

$$\frac{\partial h\vec{u}}{\partial t} + \nabla \cdot (h\vec{u} \otimes \vec{u} + g\frac{h^2}{2}\bar{I}) = -gh\nabla z_b - gh\vec{S}_f \quad (1.6)$$

where $\vec{u} = \langle u, v \rangle^T$ represents the flow velocity, g the acceleration due to gravity, S_f the friction term, and where \bar{I} denotes the identity matrix, ∇ the gradient operator, $\nabla \cdot$ the divergence operator, and \otimes the tensor product. We define the conservative variables $\vec{w} = \langle h, hu, hv \rangle^T$ and the primitive variables $\vec{w}_p = \langle h, u, v \rangle^T$. This corresponds to the general form Equation 1.1 with:

$$\bar{F}(\vec{w}) = \begin{bmatrix} hu & hv \\ hu^2 + g\frac{h^2}{2} & huv \\ huv & hv^2 + g\frac{h^2}{2} \end{bmatrix}$$

and

$$\vec{S} = -gh\nabla z_b - gh\vec{S}_f$$

1.2.2 Handling bathymetry

Handling bathymetry in the finite-volume resolution scheme requires particular care. Indeed, the bathymetry jump inherent to the fact that bathymetry is considered constant per cell can generate spurious waves. The reconstruction method proposed by Audusse et al. (2004) is therefore implemented in **CERF**. Let $\Delta z_{k/a}$ be the *bathymetry jump* at the interface between

cells. At the interface, we then distinguish the flux \vec{F}_k contributing to cell C_k and the flux \vec{F}_a contributing to cell C_a , such that:

$$\begin{aligned}\vec{F}_k(\vec{w}_k(t), \vec{w}_a(t), \vec{n}_{k/a}, \Delta z_{k/a}) &= \vec{F}(\vec{w}_k^*(t), \vec{w}_a^*(t), \vec{n}_{k/a}) + \left\{ \begin{array}{c} 0 \\ \frac{g}{2}(h_k^2 - h_k^{*2}) \vec{n}_{k/a} \end{array} \right\} \\ \vec{F}_a(\vec{w}_k(t), \vec{w}_a(t), \vec{n}_{k/a}, \Delta z_{k/a}) &= \vec{F}(\vec{w}_k^*(t), \vec{w}_a^*(t), \vec{n}_{k/a}) + \left\{ \begin{array}{c} 0 \\ \frac{g}{2}(h_a^2 - h_a^{*2}) \vec{n}_{k/a} \end{array} \right\}\end{aligned}$$

where $\vec{w}_k^* = \langle h_k^*, \vec{u}_k \rangle^T$ and $\vec{w}_a^* = \langle h_a^*, \vec{u}_a \rangle^T$, with:

$$\begin{aligned}h_k^* &= \max(0, h_k - \max(0, \Delta z_{k/a})) \\ h_a^* &= \max(0, h_a - \max(0, \Delta z_{k/a}))\end{aligned}$$

1.2.3 Refinement criterion

For the Saint-Venant model, to define the numerical entropy production Equation 1.4, we use the entropy defined by:

$$s(\vec{w}) = \frac{1}{2} (h \|\vec{u}\|^2 + gh^2 + 2ghz_b)$$

and the entropy flux

$$\vec{\psi}_s(\vec{w}) = \left(s + \frac{gh^2}{2} \right) \vec{u}.$$

1.2.4 Handling friction

In the momentum conservation equation Equation 1.6, the friction term is taken into account. In **CERF**, the friction term is modeled using either the Manning-Strickler term or the Darcy-Weisbach term. The Manning-Strickler term is defined by:

$$\vec{S}_f = C_f \frac{\|\vec{u}\| \vec{u}}{h^{4/3}}$$

where $C_f = n^2$ is the Manning-Strickler friction coefficient.

The Darcy-Weisbach term is defined by:

$$\vec{S}_f = C_f \frac{\|\vec{u}\| \vec{u}}{h}$$

The friction term is handled using a “*splitting*” method. We first solve problem Equation 1.1 without the friction term,

$$\frac{\partial \vec{w}}{\partial t} + \nabla \cdot \vec{F}(\vec{w}) = -gh \nabla z_b \quad (1.7)$$

then we solve

$$\frac{\partial \vec{w}}{\partial t} = -gh\vec{S}_f \quad (1.8)$$

We then assume that friction modifies neither the water depth nor the direction of the flow, but only the magnitude of the flow velocity. The friction problem for the velocity can then be solved using a first-order implicit scheme over a time step δt_n and the velocity “*penalized*” as in Varra et al. (2024).

Let \vec{u}^* be the solution of problem Equation 1.7; problem Equation 1.8 can then be solved and the velocity penalized by:

$$\vec{u} = \frac{2\vec{u}^*}{1 + \sqrt{1 + 4\beta}}$$

where

$$\beta = \delta t_n g C_f h^{-q}$$

with $q = 4/3$ for the Manning-Strickler model and $q = 1$ for the Darcy-Weisbach model.

1.2.5 Riemann solver for the Saint-Venant model

1.2.5.1 Overview

Numerical fluxes in **CERF** are computed using a Riemann solver. There are many references concerning this solver, such as Toro (1997). This section describes the Riemann solver used for the Saint-Venant model, based on the [course](#) by [Gloria Faccanoni](#).

We therefore seek to solve the Saint-Venant problem in one dimension. We consider a shallow-water flow with depth $h(x, t)$ and horizontal velocity $u(x, t)$ over a flat bottom, described by the following problem:

$$\begin{cases} \frac{\partial h}{\partial t} + \frac{\partial(hu)}{\partial x} = 0 \\ \frac{\partial(hu)}{\partial t} + \frac{\partial(hu^2 + \frac{1}{2}gh^2)}{\partial x} = 0 \end{cases} \quad (1.9)$$

With initial conditions such that:

$$\vec{w}_p(x, 0) = \begin{cases} \vec{w}_{p_L} & \text{if } x < 0 \\ \vec{w}_{p_R} & \text{if } x > 0 \end{cases}$$

where $\vec{w}_{p_L} = \langle h_L, u_L \rangle^T$ and $\vec{w}_{p_R} = \langle h_R, u_R \rangle^T$ are two constant states.

By expanding Equation 1.9, the system can be rewritten in the following form:

$$\frac{\partial}{\partial t} \begin{Bmatrix} h \\ u \end{Bmatrix} + \begin{bmatrix} u & h \\ g & u \end{bmatrix} \frac{\partial}{\partial t} \begin{Bmatrix} h \\ u \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

We then seek the eigenvalues λ_1 and λ_2 as well as the eigenvectors \vec{r}_1 and \vec{r}_2 of the matrix thus obtained. A quick calculation gives:

$$\lambda_1 = u - \sqrt{gh}, \lambda_2 = u + \sqrt{gh}$$

with $\lambda_1 \leq \lambda_2$ and

$$\vec{r}_1 = \begin{Bmatrix} -\sqrt{h} \\ \sqrt{g} \end{Bmatrix}, \vec{r}_2 = \begin{Bmatrix} \sqrt{h} \\ \sqrt{g} \end{Bmatrix}$$

i Note 1

The eigenvalues are used to determine the wave propagation speed within the cell Important 1 as: $\lambda_k = \text{Max}(|u - \sqrt{gh}|, |u + \sqrt{gh}|)$

To determine the nature of the two characteristic fields, we compute $\nabla_{w_p} \lambda_1 \cdot \vec{r}_1 = 3/2 \neq 0$ and $\nabla_{w_p} \lambda_2 \cdot \vec{r}_2 = 3/2 \neq 0$. The characteristic fields are therefore genuinely nonlinear. There will therefore be no contact discontinuity, and the two waves are either shocks or rarefaction waves.

Since there are 2 waves, 1 Riemann invariant $I_{1;1}, I_{2;1}$ can be defined for each wave, such that $\nabla_{w_p} I_{1;1} \cdot \vec{r}_1 = 0$ and $\nabla_{w_p} I_{2;1} \cdot \vec{r}_2 = 0$. It is easy to verify that $I_{1;1} = u + 2\sqrt{gh}$ and $I_{2;1} = u - 2\sqrt{gh}$ are Riemann invariants.

The entropy weak solution of the Riemann problem consists of at most three constant states $\vec{w}_{p_L}, \vec{w}_{p_*}, \vec{w}_{p_R}$, separated by two discontinuities. These discontinuities are either shocks or rarefaction waves. In what follows, we determine the state \vec{w}_{p_*} .

1.2.5.2 Study of the rarefaction waves

Consider a state \vec{w}_{p_g} and a state \vec{w}_{p_d} , connected by a k-rarefaction wave. The following conditions must then be satisfied:

- $\lambda_k(\vec{w}_{p_g}) < \lambda_k(\vec{w}_{p_d})$
- The Riemann invariant is conserved along the rarefaction wave $I_{k;1}(\vec{w}_{p_g}) = I_{k;1}(\vec{w}_{p_d})$

Let us apply these conditions to the Saint-Venant problem.

Consider first a 1-rarefaction connecting a state \overrightarrow{w}_{p_L} to a state \overrightarrow{w}_{p_*} , separated by a rarefaction wave. The Riemann invariants give

$$I_{k;1}(\overrightarrow{w}_{p_L}) = I_{k;1}(\overrightarrow{w}_{p_*}) \rightarrow u_L + 2\sqrt{g h_L} = u_* + 2\sqrt{g h_*}$$

that is

$$u_* = u_L + 2\sqrt{g}(\sqrt{h_L} - \sqrt{h_*}) \quad (1.10)$$

Furthermore,

$$\lambda_k(\overrightarrow{w}_{p_L}) < \lambda_k(\overrightarrow{w}_{p_*}) \rightarrow u_L - \sqrt{g h_L} < u_* - \sqrt{g h_*} \rightarrow h_* < h_L$$

and

$$u_* > u_L$$

Consider finally a 2-rarefaction connecting a state \overrightarrow{w}_{p_*} to a state \overrightarrow{w}_{p_R} , separated by a rarefaction wave. The Riemann invariants give

$$I_{k;1}(\overrightarrow{w}_{p_*}) = I_{k;1}(\overrightarrow{w}_{p_R}) \rightarrow u_* - 2\sqrt{g h_*} = u_R - 2\sqrt{g h_R}$$

that is

$$u_* = u_R + 2\sqrt{g}(\sqrt{h_*} - \sqrt{h_R}) \quad (1.11)$$

Furthermore,

$$\lambda_k(\overrightarrow{w}_{p_*}) < \lambda_k(\overrightarrow{w}_{p_R}) \rightarrow u_* + \sqrt{g h_*} < u_R + \sqrt{g h_R} \rightarrow h_* < h_R$$

and

$$u_* < u_R$$

1.2.5.3 Study of the shocks

Consider a state \overrightarrow{w}_{p_g} and a state \overrightarrow{w}_{p_d} , connected by a k shock wave. The following conditions must then be satisfied:

- The Rankine-Hugoniot conditions, where $\dot{\sigma}_k$ denotes the shock speed: $\dot{\sigma}_k = \frac{[\vec{F}(\overrightarrow{w})]}{[\overrightarrow{w}]}$
- The Lax entropy conditions for a genuinely nonlinear field k :

$$\begin{cases} \lambda_{k-1}(\overrightarrow{w}_{p_g}) < \dot{\sigma}_k \\ \lambda_k(\overrightarrow{w}_{p_d}) < \dot{\sigma}_k < \lambda_k(\overrightarrow{w}_{p_g}) \\ \dot{\sigma}_k < \lambda_{k+1}(\overrightarrow{w}_{p_d}) \end{cases}$$

Applied to the Saint-Venant problem, the Rankine-Hugoniot conditions give:

$$\dot{\sigma}_k = \frac{h_d u_d - h_g u_g}{h_d - h_g} = \frac{h_d u_d^2 + \frac{g}{2} h_d^2 - h_g u_g^2 + \frac{g}{2} h_g^2}{h_d u_d - h_g u_g}$$

in particular, we can define j_k such that

$$j_k = h_g u_g - \dot{\sigma}_k h_g = h_d u_d - \dot{\sigma}_k h_d.$$

By eliminating the shock speed, we obtain:

$$(u_g - u_d)^2 = \frac{g}{2 h_d h_g} (h_g - h_d)^2 (h_g + h_d)$$

and therefore

$$j_k = \frac{g}{2} \frac{h_g - h_d}{u_d - u_g}$$

Consider first a 1-shock connecting a state \overrightarrow{w}_{p_L} to a state \overrightarrow{w}_{p_*} , separated by a shock wave. The Lax conditions give:

$$\begin{cases} \lambda_1(\overrightarrow{w}_{p_*}) < \dot{\sigma}_1 < \lambda_1(\overrightarrow{w}_{p_L}) \\ \dot{\sigma}_1 < \lambda_2(\overrightarrow{w}_{p_*}) \end{cases} \rightarrow \begin{cases} u_* - \sqrt{g h_*} < \dot{\sigma}_1 < u_L - \sqrt{g h_L} \\ \dot{\sigma}_1 < u_* + \sqrt{g h_*} \end{cases}$$

The Rankine-Hugoniot conditions give:

$$\begin{cases} j_1 = h_* (u_* - \dot{\sigma}_1) = h_L (u_L - \dot{\sigma}_1) \\ \dot{\sigma}_1 = \frac{h_* u_*^2 + \frac{g}{2} h_*^2 - h_L u_L^2 - \frac{g}{2} h_L^2}{h_* u_* - h_L u_L} \end{cases}$$

So, from the Lax conditions and using the first Rankine-Hugoniot condition, we obtain:

$$\begin{cases} u_* - \sqrt{g h_*} < \dot{\sigma}_1 \\ u_L - \sqrt{g h_L} > \dot{\sigma}_1 \\ u_* + \sqrt{g h_*} > \dot{\sigma}_1 \end{cases} \rightarrow \begin{cases} h_* \sqrt{g h_*} > j_1 \\ h_L \sqrt{g h_L} < j_1 \\ -h_* \sqrt{g h_*} < j_1 \end{cases}$$

That is

$$h_* > h_L$$

and since $j_1 = \frac{g}{2} \frac{h_L - h_*}{u_* - u_L} > 0$, we have

$$u_* < u_L$$

and using the expression for $\dot{\sigma}_1$, we find the velocity

$$u_* = u_L + (h_L - h_*) \sqrt{\frac{g}{2} \frac{h_L + h_*}{h_L h_*}} \quad (1.12)$$

Consider finally a 2-shock connecting a state \vec{w}_{p_*} to a state \vec{w}_{p_R} , separated by a shock wave. The Lax conditions give:

$$\begin{cases} \lambda_1(\vec{w}_{p_*}) < \dot{\sigma}_2 \\ \lambda_2(\vec{w}_{p_R}) < \dot{\sigma}_2 < \lambda_2(\vec{w}_{p_*}) \end{cases} \rightarrow \begin{cases} u_* - \sqrt{gh_*} < \dot{\sigma}_2 \\ < u_R + \sqrt{gh_R} < \dot{\sigma}_2 < u_* + \sqrt{gh_*} \end{cases}$$

The Rankine-Hugoniot conditions give:

$$j_2 = h_* (u_* - \dot{\sigma}_2) = h_R (u_R - \dot{\sigma}_2) = \frac{g}{2} \frac{h_* - h_R}{u_R - u_*}$$

So, from the Lax conditions and using the first Rankine-Hugoniot condition, we obtain:

$$\begin{cases} u_* - \sqrt{gh_*} < \dot{\sigma}_2 \\ u_R + \sqrt{gh_R} < \dot{\sigma}_2 \\ u_* + \sqrt{gh_*} > \dot{\sigma}_2 \end{cases} \rightarrow \begin{cases} h_* \sqrt{gh_*} > j_2 \\ -h_R \sqrt{gh_R} > j_2 \\ -h_* \sqrt{gh_*} > j_2 \end{cases}$$

That is

$$h_* > h_R$$

and since $j_2 < 0$, we have

$$u_* > u_L$$

and using the expression for $\dot{\sigma}_2$, we find the velocity

$$u_* = u_R + (h_* - h_R) \sqrt{\frac{g}{2} \frac{h_R + h_*}{h_R h_*}} \quad (1.13)$$

1.2.5.4 Constructing the solution

To determine the solution of the Riemann problem, the state \vec{w}_{p_*} must be determined as a function of the states \vec{w}_{p_L} and \vec{w}_{p_R} . The intermediate state is connected to the left state by a 1-wave, using Equation 1.12 and Equation 1.10:

$$u_* = v^L(h_*) = \begin{cases} u_L - (\sqrt{h_*} - \sqrt{h_L}) \frac{2\sqrt{g}}{\sqrt{h_L} + \sqrt{h_*}} & \text{if } h_* \leq h_L \text{ (1-rarefaction)} \\ u_L - (\sqrt{h_*} - \sqrt{h_L}) \sqrt{\frac{g}{2} \frac{h_* + h_L}{h_* h_L}} & \text{if } h_* > h_L \text{ (1-shock)} \end{cases}$$

and the intermediate state is connected to the right state by a 2-wave, using Equation 1.11 and Equation 1.13:

$$u_* = v^R(h_*) = \begin{cases} u_R + (\sqrt{h_*} - \sqrt{h_R}) \frac{2\sqrt{g}}{\sqrt{h_R} + \sqrt{h_*}} & \text{if } h_* \leq h_R \text{ (2-rarefaction)} \\ u_R + (\sqrt{h_*} - \sqrt{h_R}) \sqrt{\frac{g}{2} \frac{h_* + h_R}{h_* h_R}} & \text{if } h_* > h_R \text{ (2-shock)} \end{cases}$$

So to find h_* , the equation $v^L(h_*) - v^R(h_*) = 0$ must be solved using a Newton-Raphson scheme.

By introducing the function $\zeta(h, \psi) = \begin{cases} \frac{2\sqrt{g}}{\sqrt{h} + \sqrt{\psi}} & \text{if } h \leq \psi \\ \sqrt{\frac{g}{2}} \frac{h + \psi}{h\psi} & \text{if } h > \psi \end{cases}$, we will simply write

$$u_* = u_R + (\sqrt{h_*} - \sqrt{h_R}) \zeta(h_*, h_R)$$

The previous calculations allow us to define the speeds of the shock waves $\dot{\sigma}_1 = u_L - h_* \sqrt{\frac{g}{2} \frac{h_L + h_*}{h_L h_*}}$ and $\dot{\sigma}_2 = u_R + h_* \sqrt{\frac{g}{2} \frac{h_R + h_*}{h_R h_*}}$.

and the speeds of the rarefaction waves $\lambda_1 = u_L - \sqrt{gh_L}$, $\lambda_1^* = u_* - \sqrt{gh_*}$, $\lambda_2^* = u_* + \sqrt{gh_*}$ and $\lambda_2 = u_R + \sqrt{gh_R}$.

The solution in a rarefaction zone is given by:

$$\vec{w}_{p_{1det}}(x, t) = \begin{cases} \frac{1}{9g} \left(u_L + 2\sqrt{gh_L} - \frac{x}{t} \right)^2 \\ \frac{1}{3} \left(u_L + 2\sqrt{gh_L} + 2\frac{x}{t} \right) \end{cases}$$

and

$$\vec{w}_{p_{2det}}(x, t) = \begin{cases} \frac{1}{9g} \left(-u_R + 2\sqrt{gh_R} + \frac{x}{t} \right)^2 \\ \frac{1}{3} \left(u_R - 2\sqrt{gh_R} + 2\frac{x}{t} \right) \end{cases}.$$

We can now write the complete solution of the Riemann problem.

Case 1: $h_* > h_L$ and $h_* > h_R$, the solution consists of a 1-shock and a 2-shock:

$$\vec{w}_p(x, t) = \begin{cases} \vec{w}_{p_L} & \text{if } x < \dot{\sigma}_1 t \\ \vec{w}_{p_*} & \text{if } \dot{\sigma}_1 t < x < \dot{\sigma}_2 t \\ \vec{w}_{p_R} & \text{if } x > \dot{\sigma}_2 t \end{cases}$$

Case 2: $h_L < h_* < h_R$, the solution consists of a 1-shock and a 2-rarefaction:

$$\vec{w}_p(x, t) = \begin{cases} \vec{w}_{p_L} & \text{if } x < \dot{\sigma}_1 t \\ \vec{w}_{p_*} & \text{if } \dot{\sigma}_1 t < x < \lambda_2^* t \\ \vec{w}_{p_{2det}}(x, t) & \text{if } \lambda_2^* t < x < \lambda_2^R t \\ \vec{w}_{p_R} & \text{if } x > \lambda_2^R t \end{cases}$$

Case 3: $h_R < h_* < h_L$, the solution consists of a 1-rarefaction and a 2-shock:

$$\vec{w}_p(x, t) = \begin{cases} \vec{w}_{p_L} & \text{if } x < \lambda_1^L t \\ \vec{w}_{p_{1det}}(x, t) & \text{if } \lambda_1^L t < x < \lambda_1^* t \\ \vec{w}_{p_*} & \text{if } \lambda_1^* t < x < \dot{\sigma}_2 t \\ \vec{w}_{p_R} & \text{if } x > \dot{\sigma}_2 t \end{cases}$$

Case 4: $h_* < h_L$ and $h_* < h_R$, the solution consists of a 1-rarefaction and a 2-rarefaction:

$$\vec{w}_p(x, t) = \begin{cases} \vec{w}_{p_L} & \text{if } x < \lambda_1^L t \\ \vec{w}_{p_{1det}}(x, t) & \text{if } \lambda_1^L t < x < \lambda_1^* t \\ \vec{w}_{p_*} & \text{if } \lambda_1^* t < x < \lambda_2^* t \\ \vec{w}_{p_{2det}}(x, t) & \text{if } \lambda_2^* t < x < \lambda_2^R t \\ \vec{w}_{p_R} & \text{if } x > \lambda_2^R t \end{cases}$$

i Note

In the case where dry zones appear, the algorithm is adapted as explained, for example, in this [course](#) by [Gloria Faccanoni](#) or in the thesis of Pons (2018).

i Note

This Riemann solver is used in **CERF**. It is implemented in Fortran90 in `~/sources/PHY/svriemann.F90`.

! Important 2

In practice, it is necessary to define a threshold value defining the “dry” zone. In **CERF**, we therefore define h_{crit} such that if $h < h_{crit}$, the zone is considered dry and $h = 0$ and $\vec{u} = \vec{0}$ are imposed. This value can be modified by the user.

1.3 Two-fluid flow model

1.3.1 Governing equations

This model was originally developed to simulate flows in marine hydrodynamics, wave propagation and wave breaking. For breaking, the most physically relevant approach is to consider the two fluids, air and water, with behavior governed by an incompressible Navier-Stokes model, with or without turbulence. Handling the interface is, moreover, a topic in its own

right! Whatever discretization approach is used, these simulations are very costly in terms of computation time. Moreover, due to numerical dissipation, it is difficult to simulate flows at very large scales. This is why, to simulate wave propagation phenomena, models of the Saint-Venant, Serre-Green-Naghdi, or inviscid irrotational flow type are often preferred. These models, however, model breaking phenomena poorly, or not at all. The literature on these topics is very extensive, and we do not claim to provide a state of the art here. We will simply offer a partial comparative study in Helluy, Philippe et al. (2005).

The two-fluid model presented here stems from the desire to find a compromise between the relevance of Navier-Stokes-type models and the simplicity of Saint-Venant-type models. We therefore consider a **mixture** of two fluids, so that no interface “tracking” is required and no surface tension can be taken into account. We further assume, which holds true in most cases studied in hydrodynamics, that viscosity can be neglected, which, from a numerical standpoint, advantageously frees us from modeling the Laplacian operator. Finally, as in Chorin (1967), we consider that the two fluids are very weakly compressible. This frees us from the need to impose incompressibility, which is costly in computation time; however, it becomes necessary to add an equation of state for the mixture. After investigation, according to Frédéric Golay and Helluy (2007), it appears that an isothermal equation of state is sufficient for most of the cases studied, so that we can also dispense with solving the energy equation. We thus obtain a Eulerian model of the form:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \quad (1.14)$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \otimes \vec{u} + p \bar{\bar{I}}) = \rho \vec{g} \quad (1.15)$$

where ρ is the density of the mixture, $\vec{u} = \langle u, v, w \rangle^T$ is the velocity of the mixture, \vec{g} is the gravity vector, and where $\bar{\bar{I}}$ denotes the identity matrix, $\nabla \cdot$ the divergence operator, and \otimes the tensor product. Equation 1.14 represents conservation of mass and Equation 1.15 represents conservation of momentum. p is the pressure of the mixture, determined by the equation of state:

$$p = p_0 + c_0^2 (\rho - (\phi \rho_a + (1 - \phi) \rho_w)) \quad (1.16)$$

where ρ_a represents the density of air, ρ_w the density of water, p_0 the reference pressure, and c_0 the speed of sound in the mixture. ϕ represents the volume fraction of air in the mixture. ϕ satisfies a transport equation, which we take in non-conservative form (Frédéric Golay and Helluy (2007)), as follows:

$$\frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi = 0 \quad (1.17)$$

Typically, we take $p_0 = 10^5 Pa$, $\rho_a = 1.kg/m^3$, $\rho_w = 1000.kg/m^3$ and $c_0 = 20.m/s$. In hydrodynamics, flows are on the order of a few m/s , so that a speed of sound on the order

of 20 m/s leads to a Mach number $M = \frac{\|\vec{u}\|}{c_0} < 0.3$. The mixture can then be considered nearly incompressible. This speed of sound is entirely artificial and does not correspond to any physical reality. It is chosen for numerical reasons. Indeed, the choice of c_0 influences the stability of the explicit numerical scheme used to solve the equations, through the CFL condition (Important 1).

However, as it happens, according to Wood's equation (Figure 1.9), while the speed of sound in water is on the order of 1481 m/s and in air on the order of 343 m/s , the speed of sound in an air-water mixture is on the order of 23 m/s . This is in no way a justification for the model used here, but it is interesting to note that the speed of sound in an air-water mixture is of the same order as the artificial speed of sound used in the two-fluid model.

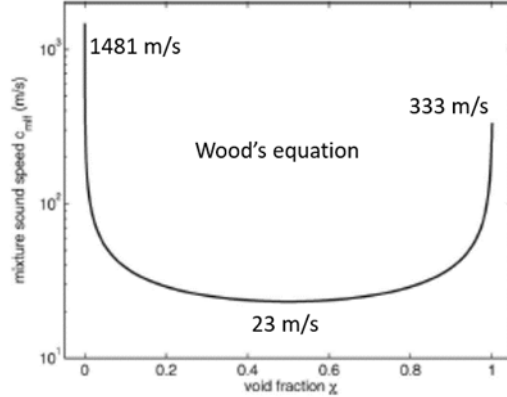


Figure 1.9: Speed of sound in an air-water mixture according to Wood's equation

Equations Equation 1.14, Equation 1.15 and Equation 1.17, together with Equation 1.16, constitute the hyperbolic problem solved by **CERF** to model two-fluid flows. We define the conservative variables $\vec{w} = \langle \rho, \rho u, \rho v, \rho w, \phi \rangle^T$ and the primitive variables $\vec{w}_p = \langle \rho, u, v, w, p \rangle^T$. This corresponds to the general form Equation 1.1.

1.3.2 Refinement criterion

For the two-fluid model, to define the numerical entropy production Equation 1.4, we use the entropy defined by:

$$s(\vec{w}) = \frac{1}{2}\rho\|\vec{u}\|^2 + c_0^2\rho\ln(\rho) - c_0^2(\rho_w - \rho_a)\phi$$

and the entropy flux

$$\vec{\psi}_s(\vec{w}) = (s + c_0^2\rho + c_0^2(\rho_w - \rho_a))\vec{u} = \left(\frac{1}{2}\rho\|\vec{u}\|^2 + c_0^2\rho(\ln(\rho) + 1)\right)\vec{u}.$$

1.3.3 Free-surface sharpening

The finite-volume resolution scheme for the hyperbolic system Equation 1.14, Equation 1.15 and Equation 1.17, even at second order, is known to be dissipative. As a result, the air-water interface diffuses and widens. To counteract this phenomenon, an interface sharpening model can be used. The idea is to add a source term to the transport equation for ϕ Equation 1.17, which penalizes mixing zones using a splitting method. Over a fictitious time step, we therefore compute:

$$\frac{\partial \phi}{\partial \tau} = \phi^2(1 - \phi)^2(\phi - c)$$

where c is determined so as to conserve the volume fraction, i.e.:

$$c = \frac{\int_{\Omega} \phi^3(1 - \phi)^2 d\Omega}{\int_{\Omega} \phi^2(1 - \phi)^2 d\Omega}$$

Note that for $\phi = 0$ or $\phi = 1$, this source term has no effect. Finally, since ϕ is modified, for the sake of conservation we also modify the density and momentum of the mixture:

$$\begin{aligned} \frac{\partial \rho}{\partial \tau} &= \phi^2(1 - \phi)^2(\phi - c)(\rho_a - \rho_w) \\ \frac{\partial(\rho \vec{u})}{\partial \tau} &= \phi^2(1 - \phi)^2(\phi - c)\vec{u}(\rho_a - \rho_w) \end{aligned}$$

The fictitious time step is calibrated to represent a fraction of the flux. This fraction is determined by a sharpening coefficient α_{sharp} . In practice, this coefficient is on the order of $10^{-3} - 10^{-2}$.

1.3.4 The “rigid fluid” concept

Following the fictitious domain principle, part of the cells can be penalized to simulate an obstacle by imposing a zero velocity. Following the work of Coquerelle and Cottet (2008), part of the cells can also be penalized to obtain “rigid solid”-type behavior. The extension of the work of Coquerelle and Cottet (2008) to a two-fluid model was carried out by Altazin (2017). Consider a “solid” occupying the domain Ω_s , of uniform density ρ_s , center of gravity G , mass M_s , and inertia matrix $\overline{\overline{J}}_s$. The velocity of any point P of the solid is given by:

$$\vec{v}_s = \vec{v}_G + \vec{\omega}_G \wedge \overrightarrow{GP}$$

where \vec{v}_G is the velocity of the center of gravity, $\vec{\omega}_G$ is the angular velocity of the solid, and \overrightarrow{GP} is

the vector connecting the center of gravity to the point P . By minimizing $\int_{\Omega_s} \|\rho \vec{u} - \rho_s \vec{v}_s\|^2 d\Omega_s$, we obtain, for a uniform solid density:

$$\vec{v}_G = \frac{\int_{\Omega_s} \rho \vec{u} d\Omega_s}{M_s}$$

and

$$\vec{\omega}_G = \overline{\overline{J}}_s^{-1} \int_{\Omega_s} \overline{GP} \wedge \vec{u}.$$

At the end of each time step, the position and velocity of the solid's center of gravity and the solid's rotational velocity are computed. ρ_s and \vec{v}_s are then assigned to all the cells of the “solid”. In **CERF**, the solid is determined by its envelope. This envelope is defined by a triangular mesh. The cells inside the envelope are determined using a “ray casting” technique. In **CERF**, a solid can be considered “fixed”, “with imposed velocity”, or “free”.

1.3.5 Riemann solver for the two-fluid model

1.3.5.1 Overview

Numerical fluxes in **CERF** are computed using a Riemann solver. There are many references concerning this solver, such as Toro (1997). This section describes the Riemann solver used for the two-fluid flow model, based on Frédéric Golay and Helluy (2007).

We therefore seek to solve the isothermal two-fluid problem in one dimension. We consider a flow with density $\rho(x, t)$, horizontal velocity $u(x, t)$, pressure $p(x, t)$, and volume fraction $\phi(x, t)$, described by the following problem:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0 \\ \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2 + p)}{\partial x} = 0 \\ \frac{\partial \rho \phi}{\partial t} + \frac{\partial \rho \phi u}{\partial x} = 0 \end{cases} \quad (1.18)$$

With an equation of state of the form:

$$p = p_0 + c_0^2 (\rho - (\phi \rho_a + (1 - \phi) \rho_w))$$

and initial conditions such that:

$$\overline{w}_p(x, 0) = \begin{cases} \overline{w}_{pL} & \text{if } x < 0 \\ \overline{w}_{pR} & \text{if } x > 0 \end{cases}$$

where $\vec{w}_{p_L} = \langle \rho_L, u_L, \phi_L \rangle^T$ and $\vec{w}_{p_R} = \langle \rho_R, u_R, \phi_R \rangle^T$ are two constant states.

By expanding Equation 1.18, the system can be rewritten in the following form:

$$\frac{\partial}{\partial t} \begin{Bmatrix} \rho \\ u \\ \phi \end{Bmatrix} + \begin{bmatrix} u & \rho & 0 \\ c_0^2 & u & c_0^2(\rho_w - \rho_a) \\ \rho & 0 & u \end{bmatrix} \frac{\partial}{\partial x} \begin{Bmatrix} \rho \\ u \\ \phi \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}.$$

We then seek the eigenvalues λ_1 , λ_2 , and λ_3 as well as the eigenvectors \vec{r}_1 , \vec{r}_2 , and \vec{r}_3 of the matrix thus obtained. A quick calculation gives:

$$\lambda_1 = u - c_0, \lambda_2 = u, \lambda_3 = u + c_0$$

with $\lambda_1 \leq \lambda_2 \leq \lambda_3$ and

$$\vec{r}_1 = \begin{Bmatrix} -\rho/c_0 \\ 1 \\ 0 \end{Bmatrix}, \vec{r}_2 = \begin{Bmatrix} (\rho_w - \rho_a) \\ 0 \\ -1 \end{Bmatrix}, \vec{r}_3 = \begin{Bmatrix} \rho/c_0 \\ 1 \\ 0 \end{Bmatrix}.$$

i Note 2

The eigenvalues are used to determine the wave propagation speed within the cell Important 1 as: $\lambda_k = \text{Max}(|u - c_0|, |u|, |u + c_0|)$

To determine the nature of the three characteristic fields, we compute $\nabla_{w_p} \lambda_1 \cdot \vec{r}_1 = 1 \neq 0$, $\nabla_{w_p} \lambda_2 \cdot \vec{r}_2 = 0$ and $\nabla_{w_p} \lambda_3 \cdot \vec{r}_3 = 1 \neq 0$. Characteristic fields 1 and 3 are therefore **genuinely nonlinear**, while field 2 is **linearly degenerate**. There will therefore be one contact discontinuity and two waves, which are either shocks or rarefaction waves.

Since there are 3 waves, 2 Riemann invariants can be defined for each wave. The Riemann invariants of wave k are such that $\nabla_{w_p} I_{k;1} \cdot \vec{r}_k = 0$ and $\nabla_{w_p} I_{k;2} \cdot \vec{r}_k = 0$. It is easy to verify that:

$$\begin{aligned} I_{1;1} &= \phi, I_{1;2} = u + c_0 \ln(\rho), \\ I_{2;1} &= u, I_{2;2} = p, \\ I_{3;1} &= \phi, I_{3;2} = u - c_0 \ln(\rho). \end{aligned}$$

The entropy weak solution of the Riemann problem consists of at most four constant states \vec{w}_{p_L} , \vec{w}_{p_I} , $\vec{w}_{p_{II}}$, \vec{w}_{p_R} , separated by a shock or a rarefaction wave, a contact discontinuity, then a shock or a rarefaction wave (Figure 1.10). In what follows, we determine the states \vec{w}_{p_I} and $\vec{w}_{p_{II}}$.

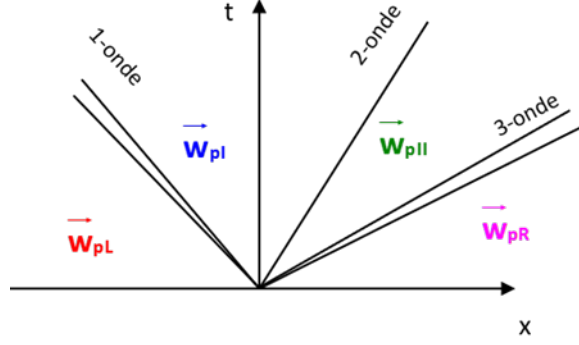


Figure 1.10: Diagram of the solutions to the Riemann problem with the isothermal two-fluid model

1.3.5.2 Study of the rarefaction waves

Consider a state $\overrightarrow{w_{p_g}}$ and a state $\overrightarrow{w_{p_d}}$, connected by a k-rarefaction wave. The following conditions must then be satisfied:

- $\lambda_k(\overrightarrow{w_{p_g}}) < \lambda_k(\overrightarrow{w_{p_d}})$
- The Riemann invariants are conserved along the rarefaction wave $I_{k;i}(\overrightarrow{w_{p_g}}) = I_{k;i}(\overrightarrow{w_{p_d}})$ for $i = 1, 2$

Let us apply these conditions to the isothermal two-fluid problem.

Consider first a 1-rarefaction connecting a state $\overrightarrow{w_{p_L}}$ to a state $\overrightarrow{w_{p_I}}$, separated by a rarefaction wave. The Riemann invariants give:

$$\begin{cases} I_{1;1}(\overrightarrow{w_{p_L}}) = I_{1;1}(\overrightarrow{w_{p_I}}) \\ I_{1;2}(\overrightarrow{w_{p_L}}) = I_{1;2}(\overrightarrow{w_{p_I}}) \end{cases} \rightarrow \begin{cases} \phi_L = \phi_I \\ u_L + c_0 \ln(\rho_L) = u_I + c_0 \ln(\rho_I) \end{cases}$$

that is

$$u_I = u_L + c_0 \ln\left(\frac{\rho_L}{\rho_I}\right) \quad (1.19)$$

Furthermore,

$$\lambda_1(\overrightarrow{w_{p_L}}) < \lambda_1(\overrightarrow{w_{p_I}}) \rightarrow u_L < u_I$$

and consequently, from Equation 1.19, $\rho_L > \rho_I$.

Consider finally a 3-rarefaction connecting a state $\overrightarrow{w_{p_{II}}}$ to a state $\overrightarrow{w_{p_R}}$, separated by a rarefaction wave. The condition on the velocities is written:

$$\lambda_3(\overrightarrow{w_{p_{II}}}) < \lambda_3(\overrightarrow{w_{p_R}}) \rightarrow u_{II} < u_R.$$

The Riemann invariants give

$$\begin{cases} I_{3;1}(\vec{w}_{p_{II}}) = I_{3;1}(\vec{w}_{p_R}) \\ I_{3;2}(\vec{w}_{p_{II}}) = I_{3;2}(\vec{w}_{p_R}) \end{cases} \rightarrow \begin{cases} \phi_{II} = \phi_R \\ u_{II} - c_0 \ln(\rho_{II}) = u_R - c_0 \ln(\rho_R) \end{cases}$$

that is

$$u_{II} = u_R - c_0 \ln\left(\frac{\rho_R}{\rho_{II}}\right) \quad (1.20)$$

so since $u_{II} < u_R$, we have $\rho_{II} < \rho_R$.

1.3.5.3 Study of the shocks

Consider a state \vec{w}_{p_g} and a state \vec{w}_{p_d} , connected by a k shock wave. The following conditions must then be satisfied:

- The Rankine-Hugoniot conditions, where $\dot{\sigma}_k$ denotes the shock speed: $\dot{\sigma}_k = \frac{[\vec{F}(\vec{w})]}{[\vec{w}]}$
- The Lax entropy conditions for a genuinely nonlinear field k :

$$\begin{cases} \lambda_{k-1}(\vec{w}_{p_g}) < \dot{\sigma}_k < \lambda_k(\vec{w}_{p_g}) \\ \lambda_k(\vec{w}_{p_d}) < \dot{\sigma}_k < \lambda_{k+1}(\vec{w}_{p_d}) \end{cases}$$

Applied to the two-fluid problem, the Rankine-Hugoniot conditions give:

$$\begin{aligned} \dot{\sigma}_k &= \frac{\rho_d u_d - \rho_g u_g}{\rho_d - \rho_g} = \frac{\rho_d u_d^2 + p_d - \rho_g u_g^2 - p_g}{\rho_d u_d - \rho_g u_g} = \frac{\rho_d u_d \phi_d - \rho_g u_g \phi_g}{\rho_d \phi_d - \rho_g \phi_g} \\ &\begin{cases} \rho_d \dot{\sigma}_k - \rho_g \dot{\sigma}_k = \rho_d u_d - \rho_g u_g \\ \rho_d u_d \dot{\sigma}_k - \rho_g u_g \dot{\sigma}_k = \rho_d u_d^2 + p_d - \rho_g u_g^2 - p_g \\ \rho_d \phi_d \dot{\sigma}_k - \rho_g \phi_g \dot{\sigma}_k = \rho_d u_d \phi_d - \rho_g u_g \phi_g \end{cases} \\ &\begin{cases} \rho_g(u_g - \dot{\sigma}_k) = \rho_d(u_d - \dot{\sigma}_k) \\ u_d \rho_d(u_d - \dot{\sigma}_k) - u_g \rho_g(u_g - \dot{\sigma}_k) + p_d - p_g = 0 \\ \phi_g \rho_g(u_g - \dot{\sigma}_k) = \phi_d \rho_d(u_d - \dot{\sigma}_k) \end{cases} \end{aligned}$$

So, denoting $j_k = \rho_g(u_g - \dot{\sigma}_k) = \rho_d(u_d - \dot{\sigma}_k)$, we obtain:

$$\phi_g = \phi_d$$

and

$$j_k = \frac{p_d - p_g}{u_g - u_d}$$

By eliminating the shock speed, we also have:

$$(\rho_d u_d - \rho_g u_g)^2 = (\rho_d - \rho_g) (\rho_d u_d^2 - \rho_g u_g^2 + p_d - p_g)$$

or equivalently

$$\rho_d \rho_g (u_d - u_g)^2 = (\rho_d - \rho_g) (p_d - p_g)$$

Consider first a 1-shock connecting a state \vec{w}_{p_L} to a state \vec{w}_{p_I} , separated by a shock wave. The Lax admissibility conditions are written:

$$\left\{ \begin{array}{l} \dot{\sigma}_1 < \lambda_1(\vec{w}_{p_L}) \\ \lambda_1(\vec{w}_{p_I}) < \dot{\sigma}_1 < \lambda_2(\vec{w}_{p_I}) \end{array} \right\} \rightarrow \left\{ \begin{array}{l} \dot{\sigma}_1 < u_L - c_0 \\ u_I - c_0 < \dot{\sigma}_1 \\ \dot{\sigma}_1 < u_I \end{array} \right\} \rightarrow \left\{ \begin{array}{l} \rho_L c_0 < j_1 \\ j_1 < \rho_I c_0 \\ \dot{\sigma}_1 < u_I \end{array} \right\}$$

So in particular $u_I < u_L$ and $\rho_L < \rho_I$. The Rankine-Hugoniot conditions give:

$$\left\{ \begin{array}{l} \phi_L = \phi_I \\ (u_I - u_L)^2 = \frac{\rho_I - \rho_L}{\rho_I \rho_L} (p_I - p_L) \end{array} \right.$$

from which, since $u_I < u_L$ and $\rho_L < \rho_I$:

$$u_I = u_L - \sqrt{\frac{\rho_I - \rho_L}{\rho_I \rho_L} (p_I - p_L)} = u_L + c_0 \frac{\rho_L - \rho_I}{\sqrt{\rho_I \rho_L}} \quad (1.21)$$

Consider finally a 3-shock connecting a state $\vec{w}_{p_{II}}$ to a state \vec{w}_{p_R} , separated by a shock wave. The Lax conditions give:

$$\left\{ \begin{array}{l} \lambda_2(\vec{w}_{p_{II}}) < \dot{\sigma}_3 < \lambda_3(\vec{w}_{p_{II}}) \\ \lambda_3(\vec{w}_{p_R}) < \dot{\sigma}_3 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} u_{II} < \dot{\sigma}_3 \\ \dot{\sigma}_3 < u_{II} + c_0 \\ u_R + c_0 < \dot{\sigma}_3 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} u_{II} < \dot{\sigma}_3 \\ -c_0 \rho_{II} < j_3 \\ j_3 < -c_0 \rho_R \end{array} \right.$$

So in particular $u_R < u_{II}$ and $\rho_R < \rho_{II}$. The Rankine-Hugoniot conditions are written:

$$\left\{ \begin{array}{l} \phi_{II} = \phi_R \\ (u_R - u_{II})^2 = \frac{\rho_R - \rho_{II}}{\rho_R \rho_{II}} (p_R - p_{II}) \end{array} \right.$$

from which, since $u_R < u_{II}$ and $\rho_R < \rho_{II}$:

$$u_{II} = u_R + \sqrt{\frac{\rho_R - \rho_{II}}{\rho_R \rho_{II}} (p_R - p_{II})} = u_R - c_0 \frac{\rho_R - \rho_{II}}{\sqrt{\rho_R \rho_{II}}} \quad (1.22)$$

1.3.5.4 Study of the contact discontinuity

Consider a state \vec{w}_{p_g} and a state \vec{w}_{p_d} , connected by a contact discontinuity. Across a k-contact discontinuity, entropy and the Riemann invariants are conserved. Since the discontinuity moves at speed $\dot{\sigma}_k$, we have:

$$\dot{\sigma}_k = \lambda_k(\vec{w}_{p_g}) = \lambda_k(\vec{w}_{p_d}) = \frac{[\vec{F}(\vec{w})]}{[\vec{w}]}$$

Consider then a 2-contact discontinuity connecting the state \vec{w}_{p_I} to the state $\vec{w}_{p_{II}}$. We then have

$$\dot{\sigma}_2 = u_I = u_{II} = \frac{\rho_{II}u_{II} - \rho_I u_I}{\rho_{II} - \rho_I} = \frac{\rho_{II}u_{II}^2 + p_{II} - \rho_I u_I^2 - p_I}{\rho_{II}u_{II} - \rho_I u_I} = \frac{\rho_{II}u_{II}\phi_{II} - \rho_I u_I \phi_I}{\rho_{II}\phi_{II} - \rho_I \phi_I}$$

From which we deduce:

$$u_I = u_{II} \text{ and } p_I = p_{II} \quad (1.23)$$

1.3.5.5 Constructing the solution

To simplify notation, let us define a function $H(a, b)$ such that $H(a, b) = \begin{cases} c_0 \frac{\rho_a - \rho_b}{\sqrt{\rho_a \rho_b}} & \text{if } \rho_a < \rho_b \\ c_0 \ln\left(\frac{\rho_a}{\rho_b}\right) & \text{if } \rho_a > \rho_b \end{cases}$.

For the first wave, using Equation 1.19, Equation 1.21, we can then write $u_I = u_L + H(\rho_L, \rho_I)$, and for the third wave, using Equation 1.20, Equation 1.22, $u_{II} = u_R - H(\rho_R, \rho_{II})$.

We know from the study of the contact discontinuity that:

$$u_I - u_{II} = 0$$

so

$$u_L + H(\rho_L, \rho_I) - u_R + H(\rho_R, \rho_{II}) = 0$$

Now, $\rho_I = \rho_I(p_I, \phi_I)$ and $\rho_{II} = \rho_{II}(p_{II}, \phi_{II})$. We also know that $p_I = p_{II}$; we will therefore denote the intermediate pressure p^* such that $p^* = p_I = p_{II}$. Moreover, we know $\phi_I = \phi_L$ and $\phi_{II} = \phi_R$. We can then write:

$$u_L - u_R + H(\rho_L, \rho(p^*, \phi_L)) + H(\rho_R, \rho(p^*, \phi_R)) = 0$$

This is therefore a nonlinear equation in p^* that can be solved using a Newton-Raphson method. Once p^* is known, the states \vec{w}_{p_I} and $\vec{w}_{p_{II}}$ can be determined.

i Note

This Riemann solver is used in **CERF**. It is implemented in Fortran90 in *~/sources/PHY/riemisot.F90*.

2 User Guide

This chapter describes the installation of the **CERF** software, as well as its use.

2.1 Software installation

2.1.1 Prerequisites

CERF is software written in **Fortran 90** using the **MPI** library for communication between the different processes.

! Important

It is therefore necessary to have:

- a **Linux** operating system or emulator (Ubuntu, CentOS, WSL on Windows 10 or 11, Mac OS, ...)
- a **Fortran 90** compiler (gfortran, ifort, ...)
- the **MPI** library (OpenMPI)

The output files of **CERF** are in **Tecplot** format (**VTK** to come). It is therefore recommended to have visualization software such as **ParaView** or **Visit**.

2.1.2 Installation

To install **CERF**, after downloading the archive ‘cerf_v1.0.tar’, simply unpack it in a directory of your choice.

```
tar xvf cerf_v1.0.tar .
```

The unpacked archive will then present a directory tree as illustrated in Figure 2.1.

The main **CERF** directory contains 4 subdirectories: *doc*, *sources*, *lib* and *exec*.

- the *doc* directory contains the software documentation

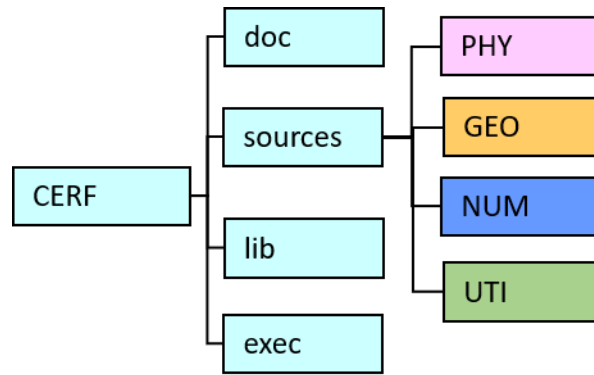


Figure 2.1: **CERF** directory tree

- the *sources* directory contains the software sources
- the *lib* directory contains the libraries created during compilation, required to build the executables
- the *exec* directory contains all the software’s executables. This is the working directory

To install **CERF**, simply go to the *exec* directory and run the *Makefile*.

```
cd CERF/exec  
make
```

i Note

By default, the *Makefile* uses the **gfortran** compiler and the **OpenMPI** library. However, these settings can be changed in the *Makefile*.

- Parameter **FC** = “gfortran” or “ifort” or for a sequential computation
- Parameter **FC** = “mpif90” or for a parallel computation
- Parameter **MPI** = “_MPI” or “NONE_MPI” depending on whether the **MPI** library is installed or not
- Parameter **FFLAGS** to redefine the compilation settings

i Note

The *Makefile* is configured to recompile only newly created or modified source files. In the event of a full recompilation, for example following a change of machine, it is advisable to clean the object files and executables before rerunning the *Makefile*, using the command:

```
make clean
```

Once the *Makefile* has finished running, the **CERF** executables are created in the *exec* directory: *cerf*, *cerf_amr*, *cerf_input* and *cerf2tec*.

i Note

Depending on the operating system used, the executables may or may not have the “.exe” extension.

2.1.3 Running a computation

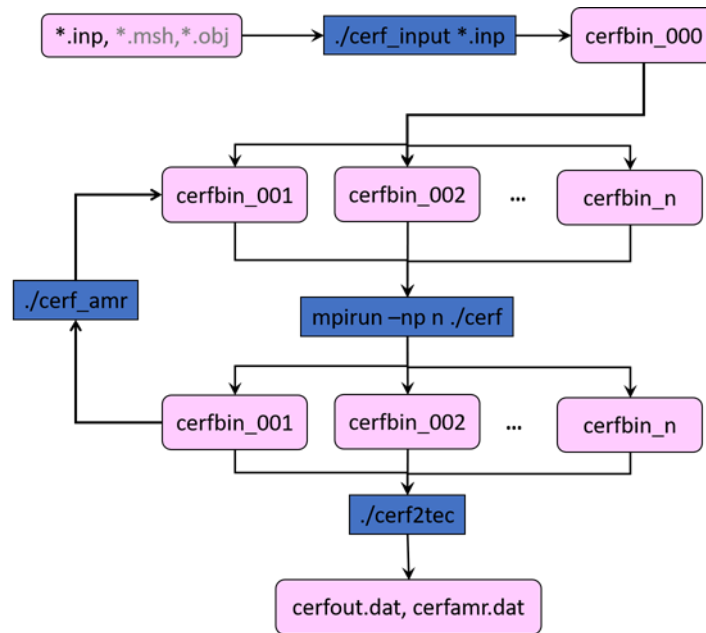


Figure 2.2: Steps of a **CERF** computation

As illustrated in Figure 2.2, running a **CERF** computation involves several steps. A *pre-processing* phase is used to generate the mesh, the initial conditions of the computation, and the computation parameters. This phase is carried out by the executable *cerf_input*. Once the mesh has been generated, the computation is performed by the executable *cerf*, and the

mesh can be modified by the executable **cerf_amr**. Finally, the executable **cerf2tec** converts the output files to **Tecplot** format for a *post-processing* phase.

- **cerf_input**

As input, **cerf_input** reads a configuration file **.inp** in a format specific to **CERF**, in which the physical and numerical parameters of the model to be processed, the mesh, the initial conditions, the boundary conditions, etc., are defined. If the mesh is defined by a file in **GMSH V2.2** format, it will also read the **.msh** file. If *rigid fluid* penalizations are defined, it will also read the *objects* in the **.obj** files.

As output, **cerf_input** generates a binary file **cerfbin_000** containing the definition of the *master mesh*, and as many binary files **cerfbin_*** as required *domains*, containing all the domain information: parameters, mesh, degrees of freedom, boundary conditions, etc.

- **cerf**

As input, **cerf** reads the binary files **cerfbin_*** generated by **cerf_input** or modified by **cerf_amr**. It then performs a computation in parallel (via **MPI** processes) on the defined domains over the required time period. During this step, the mesh is fixed. As output, once the computation is complete, **cerf** updates the binary files **cerfbin_*** with the new values of the computed fields.

- **cerf_amr**

As input, **cerf_amr** reads the binary files **cerfbin_*** generated by **cerf**. It then refines the mesh according to criteria defined in the configuration file **.inp**. As output, **cerf_amr** updates the binary files **cerfbin_*** with the new mesh. In addition, **cerf_amr** generates a file **cerfamr.dat** containing information related to the *master mesh*: refinement criterion, refinement level, etc. If *rigid fluid* penalizations are defined, **cerf_amr** also generates **solide_*** files containing the new position of the *objects*.

- **cerf2tec**

As input, **cerf2tec** reads the binary files **cerfbin_***. It then converts the binary files into a single ASCII file, named **cerfout.dat***, in **Tecplot** format for visualizing the results.

Running a computation using only command lines can become tedious. This is why scripts are often used. Below is an example of a script that lets you run a complete **CERF** computation with a single command. Simply modify it according to your needs.

💡 Example shell script

Below is an example shell script for running a computation from the file **dam.inp**. This script runs the computation on 4 processors, then modifies the mesh 20 times. The output files are renamed according to the iteration so that the progress of the computation can be visualized.

```
#!/bin/bash
./cerf_input dam.inp
./cerf2tec
mv cerfout.dat d_0.dat
mv cerfamr.dat d_mame_0.dat
for i in $(seq 1 20 )
do
mpirun -np 4 ./cerf
./cerf2tec
mv cerfout.dat d_${i}.dat
./cerf_amr
mv cerfamr.dat d_mame_${i}.dat
done
```

2.2 The configuration file *.inp

2.2.1 General overview

The configuration file is a text file used to define all the parameters and entities required for the simulation. It is divided into several sections, each beginning with a **4-character header**. The headers, to be **provided in this order**, are as follows: **PHYS**, **MAME**, **BATH**, **INIT**, **COND**, **MESH**, **OBST**, **NUME**. Each header is followed on the same line by 1 or 2 integers defining the **code** and, if applicable, the **argument**. All lines starting with “!” are comments and are ignored by the program. Comments may be placed between sections, but not within a section. Each section is then completed with lines of real or integer data, or keywords, that define the simulation parameters.

 Example line with header, code, and argument

```
! here is a line with header, code, and argument
MAME      1      1
```

The first line is a comment line. The second line is a line with a header. It starts with the header **MAME**, followed by 2 integers, **1** and **1** defining the **code** and the **argument**.

2.2.2 Defining the physical parameters: PHYS

The **PHYS** header is used to define the physical parameters of the model. The model is applied to the entire computational domain. The available models are the Saint-Venant model

Section 1.2 and the two-fluid model Section 1.3.

The model used is selected by setting the **keyword MODE**, which is mandatory and takes the value 2 for the two-fluid model and 1 for the Saint-Venant model.

For the isothermal two-fluid model, the following real-valued parameters can be defined as needed:

- **GPES**: Orientation of the gravity field along 1/ x , 2/ y , 3/ z , or none 0 (default value 2.)
- **PREF**: Reference pressure p_0 of the equation of state Equation 1.16 (default value 10^5)
- **CSSM**: Artificial speed of sound c_0 of the equation of state Equation 1.16 (default value 20.)
- **REFA**: Air density ρ_a of the equation of state Equation 1.16 (default value 1.)
- **REFW**: Water density ρ_w of the equation of state Equation 1.16 (default value 1000.)
- **SHAR**: Interface sharpening coefficient Section 1.3.3 (default value 0.)

For the Saint-Venant model, the following parameters can be defined as needed:

- **WLTR**: Threshold value defining the **dry zone** (default value 10^{-4})
- **FRMO**: Inclusion of **friction**. 0 no friction, 1 Manning-Strickler, 2/Darcy-Weisbach (default value 0). The value of the friction coefficient, which may vary spatially, is defined later in the **BATH** section.
- **BAGR**: The bathymetry gradient is either given (by a user-defined function or by bilinear interpolation) **BAGR**=0, or computed using the Barth method **BAGR**=1 (default value 0).

💡 Example parameter definition for the two-fluid model

```
PHYS 0
MODE 2.
SHAR 0.01
```

This defines a two-fluid model for an air-water flow with gravity along y and an interface sharpening coefficient of 0.01.

💡 Example parameter definition for the Saint-Venant model

```
PHYS 0
MODE 1.
FRMO 1.
```

This defines a shallow-water flow model with Manning-Strickler-type friction.

2.2.3 Defining the master mesh: MAME

The **MAME** header is used to define the *master mesh*. As a reminder, this mesh must be conformal! **CERF** has a rudimentary mesh generator (**code 1**) and reads ASCII mesh files in GMSH V2.2 format (**code 2**).

The mesh generated by **CERF** is a Cartesian mesh; the domain boundaries must be defined on one line, and the discretization in each direction on another line.

- For a “1D” mesh: **argument 0**
 - 1 line consisting of 2 reals: x_{min}, x_{max}
 - 1 line consisting of 1 integer: n_x
- For a “2D” mesh: **argument 1**
 - 1 line consisting of 4 reals: $x_{min}, x_{max}, y_{min}, y_{max}$
 - 1 line consisting of 2 integers: n_x, n_y
- For a “3D” mesh: **argument 2**
 - 1 line consisting of 6 reals: $x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$
 - 1 line consisting of 3 integers: n_x, n_y, n_z

In 1D, the dimensions along y and z are automatically determined by **CERF**, such that $y_{min} = z_{min} = -\frac{x_{max}-x_{min}}{2n_x}$ and $y_{max} = z_{max} = \frac{x_{max}-x_{min}}{2n_x}$.

In 2D, the dimension along z is automatically determined by **CERF**, such that $z_{min} = -Min\left(\frac{x_{max}-x_{min}}{2n_x}, \frac{y_{max}-y_{min}}{2n_y}\right)$ and $z_{max} = Min\left(\frac{x_{max}-x_{min}}{2n_x}, \frac{y_{max}-y_{min}}{2n_y}\right)$.

💡 Example of a master mesh definition via **CERF**

```
MAME      1      1
0. 1.    0.  1.
2 2
```

This defines a “2D” mesh consisting of 4 hexahedral-shaped blocks, 2 along x , 2 along y , and 1 along z . The blocks have size 0.5×0.5 along x and y . The dimension along z is automatically determined by **CERF**.

For more complex meshes, a mesh generated by an external mesh generator, such as **GMSH**, can be used. In this case, the mesh must be defined in a file in **GMSH V2.2** format. The **MAME** header must then be given **code 2**, and the next line contains the name of the file, which must be located in the *exec* directory.

💡 Example of a master mesh definition via **GMSH**

```
MAME      2      0
toto.msh
```

This defines a “2D” mesh from the file *toto.msh* in **GMSH** V2.2 format.

2.2.4 Defining the initial conditions: **INIT**

The **INIT** header is used to define the initial conditions of the problem to be solved. The initial conditions can be defined for the *shock tube* case (**code 0**), via a user-defined function (**code 1**), or by zones (**code 2**).

As a reminder, the primitive variables of the Saint-Venant model number $n_{dof} = 3$: h, u, v .
As a reminder, the primitive variables of the two-fluid model number $n_{dof} = 5$: ρ, u, v, w, p .

- For a “shock tube” initialization: **code 0**
The **argument** is 0. Then:
 - 1 line consisting of 2 reals for the Saint-Venant model (h, u) or 3 reals for the two-fluid model (ρ, u, p) at the initial time on the “left” side ($x < 0$)
 - 1 line consisting of 2 reals for the Saint-Venant model (h, u) or 3 reals for the two-fluid model (ρ, u, p) at the initial time on the “right” side ($x > 0$)
- For an initialization via a user-defined function: **code 1**
The **argument** defines the number of the user-defined function, written in the routine `~/CERF/sources/PHY/USER/fct_iniw.f90`.
- For an initialization by zones: **code 2**
The **argument** defines the number of zones to create. The order in which zones are defined is important. It allows zones to be “nested”. It is therefore recommended to define the first zone so that it broadly covers the study domain. Then, for each zone, 2 lines are used:
 - 1 line consisting of 6 reals defining the zone: $x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$
 - 1 line consisting of n_{dof} reals defining the primitive variables at the initial time

💡 Tip 2: Example of initialization by zones

```
INIT      2      2
-1. 4.1 -1. 3.1 -1. 1.
1. 0. 0. 0. 1.e5
-1. 1. -1. 2. -1. 1.
1000. 0. 0. 0. 1.e5
```

In this example, we aim to initialize a “dam-break” problem. In a “2D” domain of 4×3 filled with air, a column of water of height 2 and width 1 is initialized on the left. We therefore define 2 zones. The first zone broadly covers the entire domain, and the second covers the domain occupied by the water. Throughout the domain, we initialize $\rho = 1$, $\vec{u} = \vec{0}$, and $p_0 = 10^{.5}$. With the second zone, in the part occupied by the water, we initialize $\rho = 1000$, $\vec{u} = \vec{0}$, and $p_0 = 10^{.5}$.

! Important

If a user-defined function is used for initialization, it is necessary to rerun compilation and linking using the *make* command!

2.2.5 Defining bathymetry and friction: BATH

The **BATH** header is used to define the bathymetry and friction parameters, naturally in the case where the Saint-Venant model is selected.

- In the case of a flat bottom with no friction: **code 0**
- In the case of bathymetry and friction defined by a user-defined function: **code 1**
The **argument** defines the number of the user-defined function, written in the routine `~/CERF/sources/PHY/USER/fct_bathy.f90`.
- In the case of bathymetry and friction defined bilinearly per block: **code 2**
Argument 0 reads the file, and argument `-1` creates it. The next line specifies the name of the ASCII file containing the bathymetry and friction information. This file must be located in the *exec* directory. When creating the file (argument `-1`), it consists of 2 columns containing the x and y coordinates of the four vertices of each block of the master mesh. The user must then complete the file by adding two columns of reals giving, for each coordinate, the bathymetry and friction values.

2.2.6 Defining the boundary conditions: COND

The **COND** header is used to define the boundary conditions of the problem. Boundary conditions are defined by a number (written on one line) and, if applicable, real values (written on another line). The available boundary conditions are:

- **0**: free outflow condition (copied through)
- **1**: mirror condition
- **2**: Dirichlet condition on the primitive variables. The values of the n_{dof} primitive variables must then be defined on the next line. If the value 10^{20} is used, the variable is considered free.

- **3**: user-defined condition. The user must then define the boundary condition function in the routine `~/CERF/sources/PHY/USER/fct_cond.f90`. The next line defines the number of the user-defined function.
- **4**: Dirichlet condition on the conservative variables. The values of the n_{dof} conservative variables must then be defined on the next line. If the value 10^{20} is used, the variable is considered free.
- **999**: No boundary condition. Fluxes are not computed.

The definition of the boundary conditions at the domain edges then depends on the type of master mesh used:

- In the case of a “shock tube”: **code 0**
This is the case where the **MAME** section has **code 1**, **argument 0**, and the **INIT** section has **code 0**. In this case, no definition is necessary.
- In the case of a “2D” mesh: **code 1**
This is the case where the **MAME** section has **code 1**, **argument 1**. The boundary conditions are then defined on the domain edges in the order x_{min} , x_{max} , y_{min} , y_{max} . For each edge, 1 or 2 lines are used to define the boundary condition.
- In the case of a “3D” mesh: **code 2**
This is the case where the **MAME** section has **code 1**, **argument 2**. The boundary conditions are then defined on the domain edges in the order x_{min} , x_{max} , y_{min} , y_{max} , z_{min} , z_{max} . For each edge, 1 or 2 lines are used to define the boundary condition.
- In the case of a “**GMSH**” mesh: **code 3**
This is the case where the **MAME** section has **code 2**. The “*zone*” numbers defined by **GMSH** are then used. The **argument** defines the number of *zones* to process. These zones correspond to the domain edges. For each zone, 1 or 2 lines are used to define the boundary condition.

💡 Example boundary conditions

```
COND      1      0
1
1
1
1
```


In this example, “*mirror*” conditions are applied on the 4 edges of a rectangular “2D” domain.

2.2.7 Defining the mesh: MESH

The **MESH** header is used to define the domain mesh, i.e. the discretization of the blocks defined in the **MAME** section. In addition, the mesh (de)refinement parameters are defined here. The default **code** is 0.

The following **keywords** can then be defined/modified:

- **NBDS**: Number of domains to create, i.e. number of **MPI** processes (default value 1.)
- **MARL**: Maximum refinement level (“MAximal Refinement Level”) (default value 0.)
- **COPA**: Threshold value of the coarsening criterion (“mesh COarsening PArameter”) (default value 0.002)
- **REPA**: Threshold value of the refinement criterion (“mesh REfinement PArameter”) (default value 0.02)
- **FDRA**: Number of the user-defined function specifying the refinement to apply. This function is defined in the routine `~/CERF/sources/UTI/USER/fct_mesh.f90`
- **NZRA**: Number of zones defining the refinement to apply (default 1.). Followed, for each zone, by 1 integer and 6 reals defining the refinement level to apply to each block in the zone and the zone boundaries x_{min} , x_{max} , y_{min} , y_{max} , z_{min} , z_{max} .

 Tip 2: Example of refinement by zones

```
MESH 0
NBDS 4
MARL 3
COPA 0.8
REPA 1.
NZRA 3
0 -1. 4.1 -1. 3.1 -1. 1.
3 -1. 1.25 -1. 2.25 -1. 1.
0 -1. .75 -1. 1.75 -1. 1.
```

We return to the “dam-break” example Tip 2. In a “2D” domain of 4×3 filled with air, we consider a column of water on the left of height 2 and width 1. The objective is to refine the mesh around the air-water interface. We therefore define 3 zones. The first zone broadly covers the entire domain and is assigned refinement level 0. The second covers the domain occupied by the water with a margin of 0.25, to which the maximum refinement level 3 is assigned. And finally, the third covers the domain occupied by the water with a margin of -0.25 , to which the minimum refinement level 0 is assigned, in order to minimize the number of cells. **CERF** will automatically (i.e. without user intervention) adjust the mesh level between levels 0 and 3 in order to satisfy the criteria imposed by the mesh generator. In addition, the domain is decomposed into 4 domains, and the mesh refinement criterion management parameters are modified Section 1.1.3.3.

2.2.8 Defining “obstacles”: OBST

The **OBST** header is used to define a set of cells to which a “*rigid fluid*” penalization is applied, naturally in the case where the two-fluid model is selected. The default **code** is 0, and the **argument** defines the number of obstacles to process.

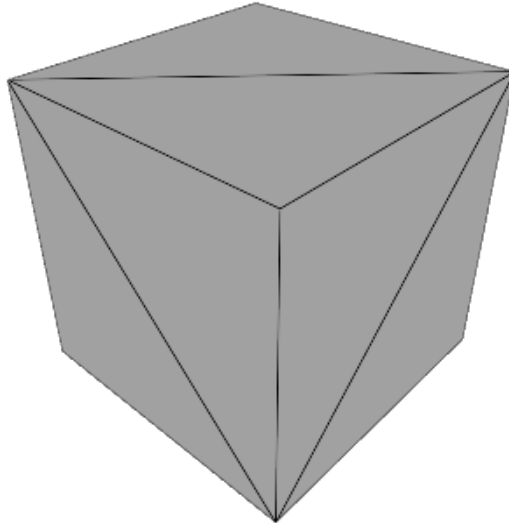
For each obstacle, the following is written on one line: a *file name*, the *obstacle density*, and the *penalization type*.

- *file name*: this is a character string (max 20 characters) defining the name of the ASCII file containing the mesh of the obstacle envelope, with triangular elements in **.OBJ** format. This file must be located in the *exec* directory.
- *obstacle density*: this is a real number defining the density of the obstacle. This density must satisfy $\rho_a \leq \rho_{obs} \leq \rho_w$.
- *penalization type*: this is an integer defining the type of penalization. The available penalization types are:
 - **-1**: “fixed” type penalization. A zero velocity is imposed.
 - **0**: “free” type penalization. The obstacle behaves as a free rigid solid.
 - **>0**: “imposed” type penalization. The obstacle behaves as a rigid solid for which the velocity of the center of gravity and the rotational velocity are imposed. The integer value defines the number of the user-defined function describing the motion in the routine `~/CERF/sources/PHY/USER/fct_mvsl.f90`.

💡 Example obstacle file in .obj format

```
v 0. 0. 0.  
v 1. 0. 0.  
v 1. 1. 0.  
v 0. 1. 0.  
v 0. 0. 1.  
v 1. 0. 1.  
v 1. 1. 1.  
v 0. 1. 1.  
f 1 2 3  
f 1 3 4  
f 2 6 7  
f 2 7 3  
f 6 5 7  
f 5 8 7  
f 5 1 4  
f 5 4 8  
f 3 4 7  
f 4 7 8  
f 2 5 6  
f 2 1 5
```

In this example, we define a cube-shaped obstacle with side length 1, centered at $(0.5, 0.5, 0.5)$. The vertices of the obstacle are defined, followed by its faces. Vertices are defined by the letter **v** followed by the vertex coordinates. Faces are defined by the letter **f** followed by the indices of the face's vertices. The figure below illustrates the shape of the obstacle thus defined.



2.2.9 Defining the numerical parameters: NUME

The **NUME** header is used to define the numerical parameters of the problem to be solved. The default **code** is 0.

The following **keywords** can then be defined/modified:

- **TINT**: Computation time interval (Time INTerval) (default value 0.)
- **CCFL**: CFL coefficient Important [1](#) (default value 0.9)
- **TDOR**: Time discretization order (Time Discretization ORder) 1 or 2 (default value 1.)
- **SDOR**: Space discretization order (Space Discretization ORder) 1 or 2 (default value 1.)
- **LIMI**: Limiter type 0/Barth, 1/Cartesian (default value 0.)
- **NPRO**: Number of probes < 10 (default value 0.). Then, for each probe, one line containing 3 reals (coordinates x , y , z of the probe)
- **SAVE**: Save type 0/No, 1/Yes. The units digit refers to saving in *cerfbin-xxx* files. The tens digit refers to saving in 1D gnuplot format. (default value 01.)

2.3 The CERF source files

2.3.1 Structure and modules

As illustrated in Figure 2.1, the **CERF** source code is organized into several directories:

- **./sources/PHY** contains all the lowest-level routines for handling physical variables and entities. The global physical variables and the data *types* (in the Fortran 90 sense), such as the degrees of freedom, their gradients, the fluxes, or the physical characteristics, are defined in the module **phy_typ** (*./sources/PHY/mod_phy_typ.f90*). The module **phy** (*./sources/PHY/mod_phy.f90*), in turn, contains the module **phy_typ** as well as all the interfaces of the routines contained in the **PHY** directory. During compilation, the compiled result of the routines in this directory is archived in the library *./lib/lib_PHY.a*.
- **./sources/GEO** contains all the routines for handling geometric variables and entities: computing interface fluxes, computing source terms, etc. The global geometric variables and the data *types* (in the Fortran 90 sense), such as the list of faces, the list of cells, the mesh, etc., are defined in the module **geo_typ** (*./sources/GEO/mod_geo_typ.f90*). The module **geo_typ** contains the module **phy**. They are nested. The module **geo** (*./sources/GEO/mod_geo.f90*), in turn, contains the module **geo_typ** as well as all the interfaces of the routines contained in the **GEO** directory. During compilation, the compiled result of the routines in this directory is archived in the library *./lib/lib_GEO.a*.
- **./sources/NUM** contains all the higher-level routines for handling numerical variables and entities: the rk2 algorithm, saving, MPI communication, etc. The global numerical variables and the data *types* (in the Fortran 90 sense), such as the “computation”, are defined in the module **num_typ** (*./sources/NUM/mod_num_typ.f90*). The module **num_typ** contains the module **geo**. They are nested. The module **num** (*./sources/NUM/mod_num.f90*), in turn, contains the module **num_typ** as well as all the interfaces of the routines contained in the **NUM** directory. During compilation, the compiled result of the routines in this directory is archived in the library *./lib/lib_NUM.a*.
- **./sources/UTI** contains the programs and all the routines required for pre- and post-processing, and therefore in particular for managing the AMR mesh. The module **uti** (*./sources/UTI/mod_uti.f90*) contains the module **num** as well as all the interfaces of the routines contained in the **UTI** directory. During compilation, the compiled result of the routines in this directory is archived in the library *./lib/lib_UTI.a*.

The modules **phy**, **geo**, **num**, and **uti** are nested and interdependent. The data types defined in them are the basic tools of the finite-volume method. As part of managing **AMR** meshes, it was necessary to develop new tools, which have been grouped into two modules.

- **zorder** (*./sources/UTI/mod_zorder.f90*): this module contains the tools for managing the “z-order” data structure used to manage the AMR mesh. It contains all the routines required for handling Morton codes.

- **msh** (*./sources/UTI/mod_msh.f90*): this module contains the tools for managing the “mesh” data structure used to manage the AMR mesh. It contains all the routines required for managing the AMR mesh. The data structures of the finite-volume code are stored in “fixed” arrays for performance reasons. Managing mesh graphs, on the other hand, is more complex and requires dynamic data structures with linked lists. The **msh** module contains all the tools required for managing these data structures.

2.3.2 Dependency graph of the main program

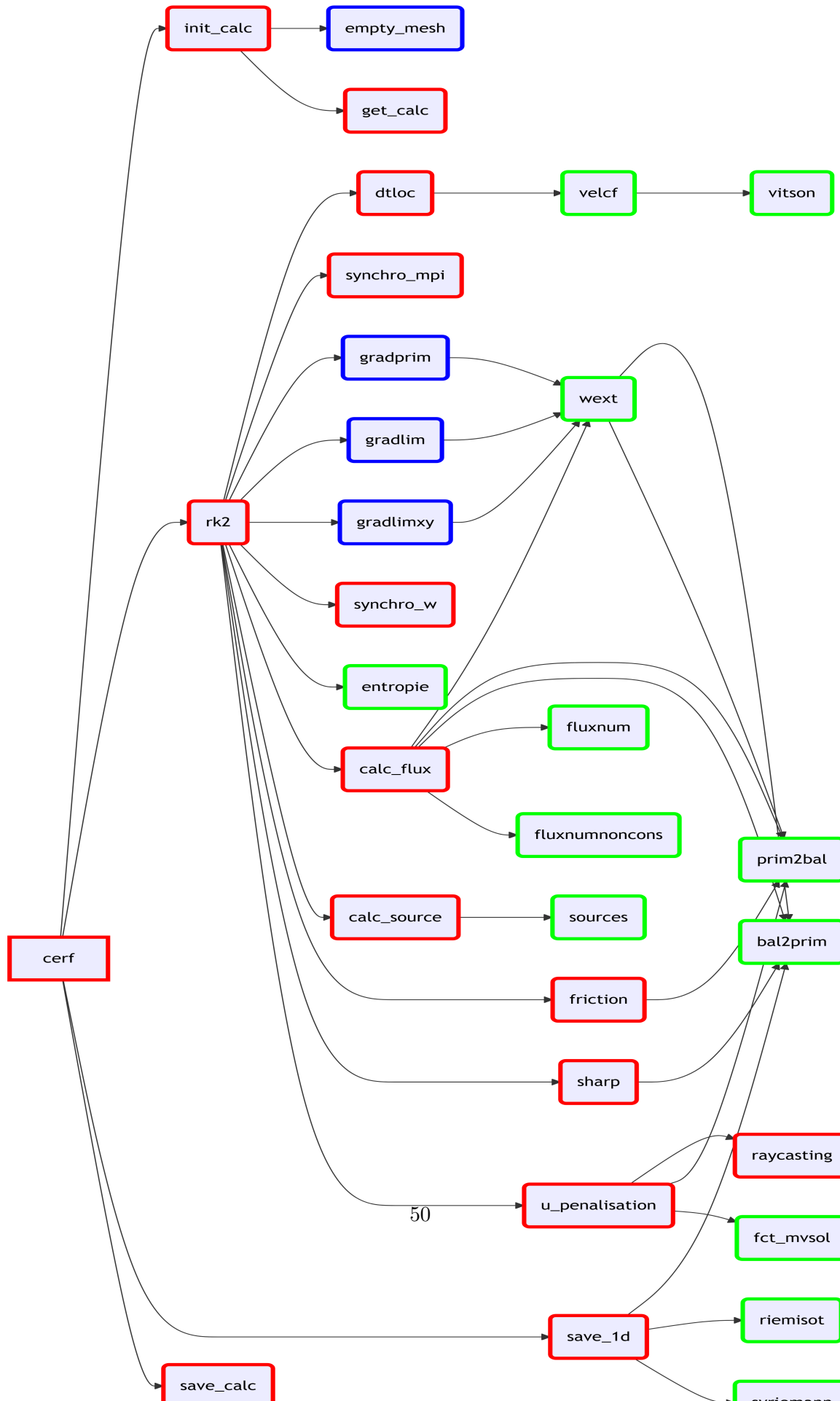
Below is the dependency graph of the main program *cerf* (excluding MPI routines and display routines). Routines depending on the **phy** module are outlined in **green**, routines depending on the **geo** module are outlined in **blue**, and routines depending on the **num** module are outlined in **red**. The HTML documentation provides direct access to a routine’s source code by clicking on its name in the graph Figure 2.3.

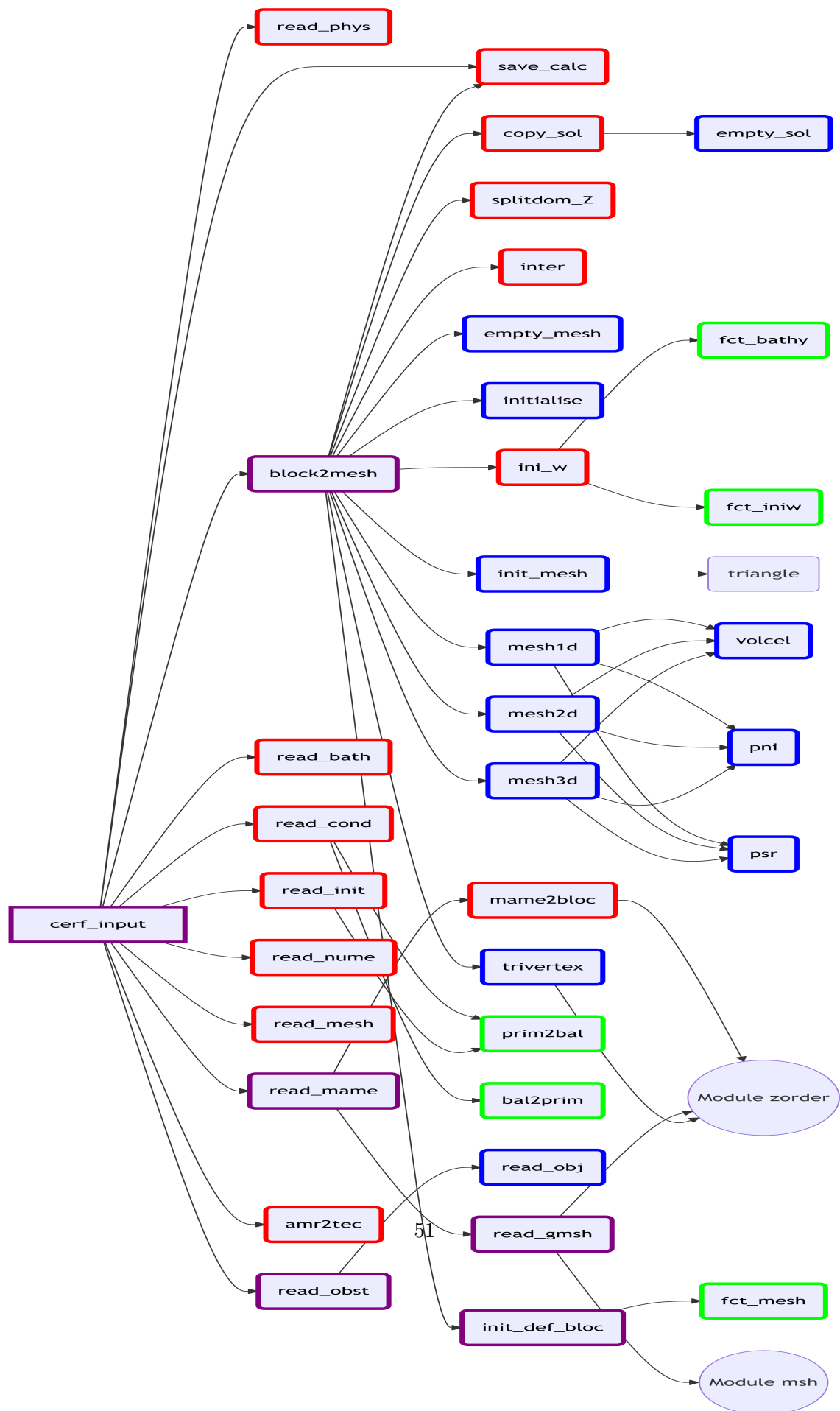
2.3.3 Dependency graph of the pre-processing program

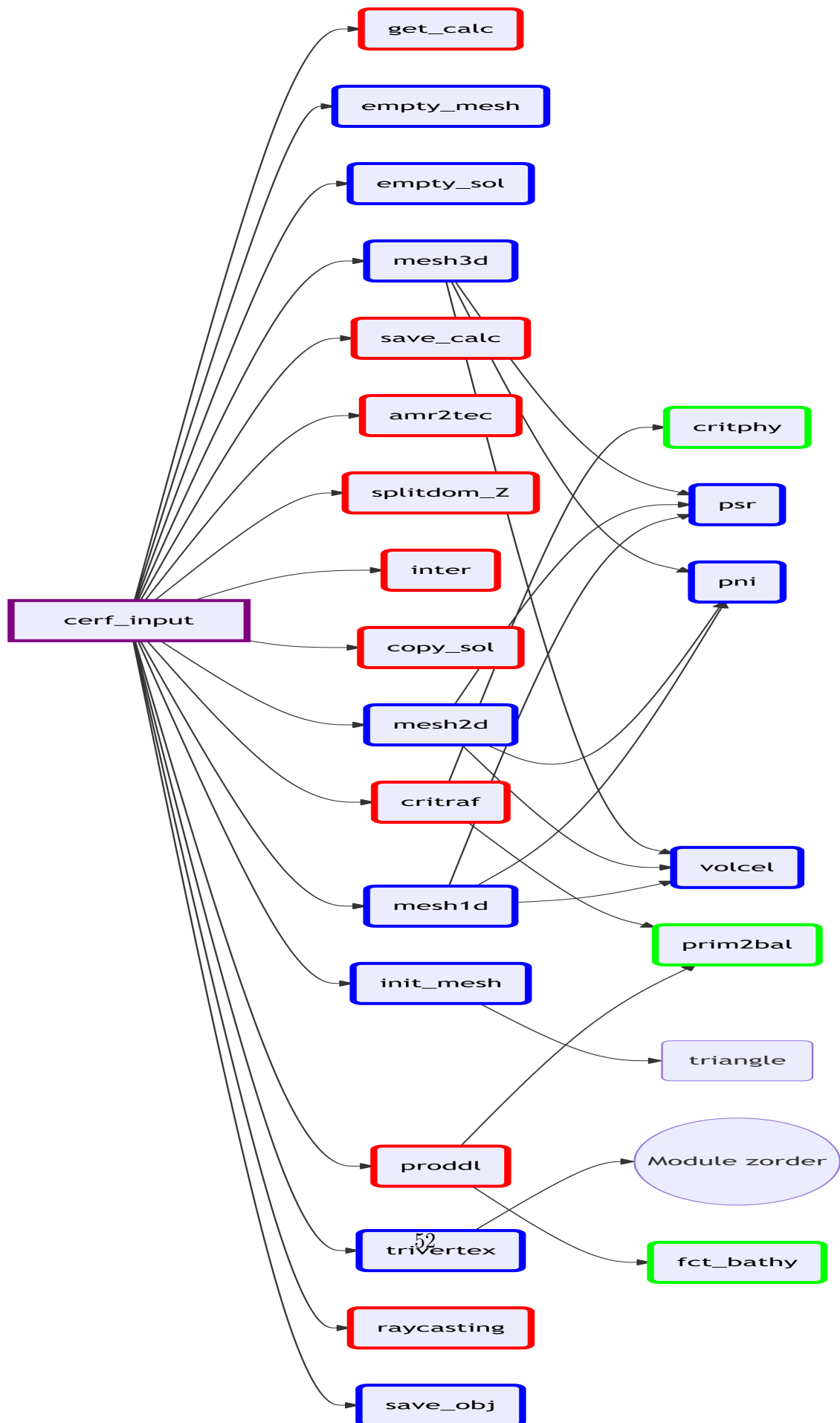
Below is the dependency graph of the pre-processing program *cerf_input* (excluding MPI routines and display routines). Routines depending on the **phy** module are outlined in **green**, routines depending on the **geo** module are outlined in **blue**, routines depending on the **num** module are outlined in **red**, and routines depending on the **uti** module are outlined in **purple**. The HTML documentation provides direct access to a routine’s source code by clicking on its name in the graph Figure 2.4.

2.3.4 Dependency graph of the mesh-modification program

Below is the dependency graph of the mesh-modification program *cerf_amr* (excluding MPI routines and display routines). Routines depending on the **phy** module are outlined in **green**, routines depending on the **geo** module are outlined in **blue**, routines depending on the **num** module are outlined in **red**, and routines depending on the **uti** module are outlined in **purple**. The HTML documentation provides direct access to a routine’s source code by clicking on its name in the graph Figure 2.5.







3 Documented examples

3.1 Examples with the Saint-Venant model

In this chapter, we present the implementation of a few classic flow test cases for the Saint-Venant model. The cases covered are as follows:

- Solving a Riemann problem
- Hydraulic jump
- Friction, McDonald test case
- ...


3.1.1 Riemann problem

We consider the classic 1D dam-break test case for the Saint-Venant model on a flat bottom. We use the test case from Faccanoni and Mangeney (2013) (test case 5). The domain is defined by $x \in [-2., 2.]$ and $t \in [0, 0.5]$. The initial conditions are given by:

$$\overrightarrow{w}_p(x, 0) = \begin{cases} \overrightarrow{w}_{p_L} = \langle h_L, u_L \rangle^T = \langle 0.0046, 0. \rangle^T & \text{if } x < 0 \\ \overrightarrow{w}_{p_R} = \langle h_R, u_R \rangle^T = \langle 0.1446, 0. \rangle^T & \text{if } x > 0 \end{cases}$$

This is therefore a dam-break case without a dry zone, with a shock and a rarefaction wave, for which the exact solution can be computed using the Riemann solver.

First, we consider a uniform, fixed 1D mesh of 400 cells over a single domain. The data file corresponding to this test case is provided below:

 Tip 3: Data file *ex1_sv_Riemann.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
```

```

!-----
!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----
! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases          (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference        (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)     (default 20.)
! REFA : Isothermal reference density for Air    (default 1.)
! REFW : Isothermal reference density for Water  (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                  (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level        (default 0.0001)
! FRMO : Friction Model
! 0/none, 1/Manning-Strickler, 2/Darcy-Weisbach  (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----
PHYS 0
MODE 1.

!-----
! Reading the Master MESH
! Header MAME
!-----
! code 1 -> creation of a mesh
!
!       argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!       argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!

```

```

!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME 1 0
-2. 2.
400

!-----
! Reading Initial conditions
! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT 0 0
0.0046 0. 0.
0.1446 0. 0.

!-----
! Reading bathymetry and friction
! Header BATH
!-----
! code 0 -> Default : flat bottom and no friction

! code 1 -> bathymetry and friction defined by user function (defined in sources/PHY/fct_bath)
!           argument: Function's Number
! code 2 -> bathymetry and friction defined linearly per block

```

```

!          argument: 0   reading the file
!          argument:-1  creation of a file containing the coordinates of the vertices of the
! next line: file name
!-----
BATH      0      0

!-----
! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case  "shock tube" : Nothing
!
! code 1 -> Description in the case  "2d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case  "3d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for  GMSH mesh
!           argument: number of zones defining a boundary condition
!           number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables  (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      0      0

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value

```



```

!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS (default 001.)
! MARL : MAlximal Refinement Level (default 000.)
! COPA : Mesh COarsening PArameter 0<.. $<1$  (default 0.002)
! REPA : Mesh REfinement PArameter 0<.. $<1$  (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!       then for each zone
!       nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 1
MARL 0
NZRA 1
0 -2. 2. -100. 100. -100. 100.

!-----
! Reading numerical parameter
! Header NUME
!-----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute (default 0.)
! CCFL : cfl condition (default 0.9)
! TDOR : Time discretization order 1 ou 2 (default 1.)
! SDOR : Space discretization order 1 ou 2 (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian (default 0.)
! SAVE : kind of backup, three-digit number (default 001.)
!       decade : shock tube backup 0/no 1/yes
!       unit : binary backup 0/no 1/yes
! NPRO : number of probe < 10 (default 000.)
!       then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT .5
CCFL .9
SDOR 2.
TDOR 2.
SAVE 11.

```

To run the computation, since there is only one domain to process over a single time interval:

```
./cerf_input ex1_sv_Riemann.inp
./cerf
```

Since the data file *ex1_sv_Riemann.inp* requests output in 1D format (parameter **SAVE**=11, section **NUME**), 1D output files have been generated. Files for the numerical solution in water depth and velocity '*n_h_001*' and '*n_u_001*', and files for the exact solution in water depth and velocity: '*e_h_001*' and '*e_u_001*'. These files can be visualized, for example, with gnuplot:

```
gnuplot
gnuplot> set term x11
gnuplot> plot 'e_h_001' w l, 'n_h_001' w l
gnuplot> plot 'e_u_001' w l, 'n_u_001' w l
gnuplot> quit
```

The numerical solution can thus be compared to the exact solution.

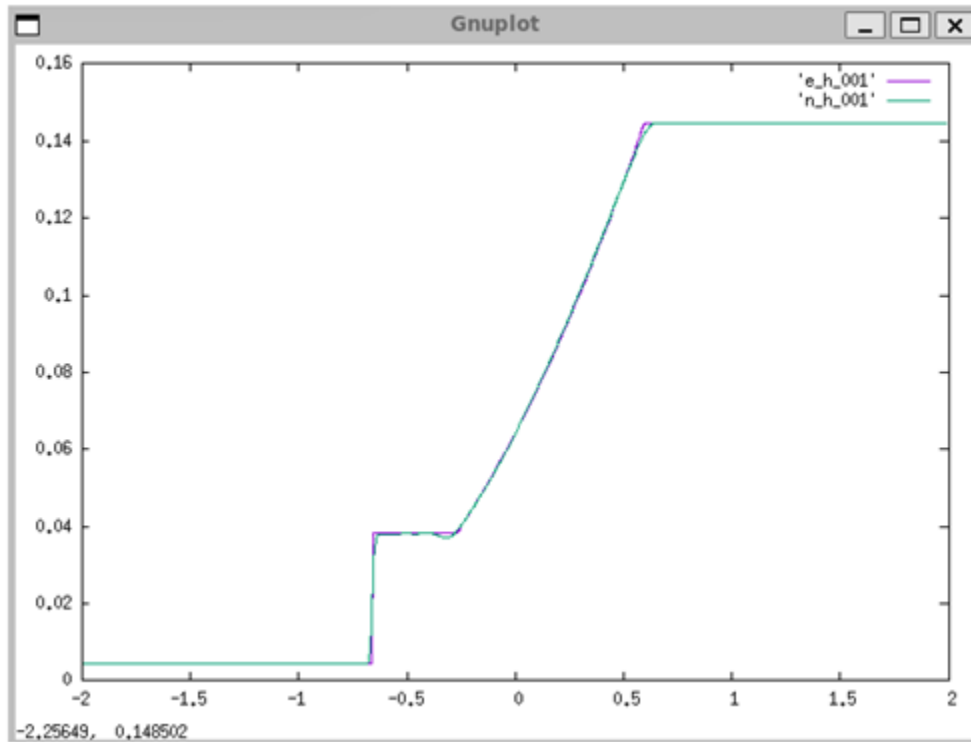


Figure 3.1: Numerical/exact comparison of water depth at $t = 0.5s$, uniform mesh of 400 cells

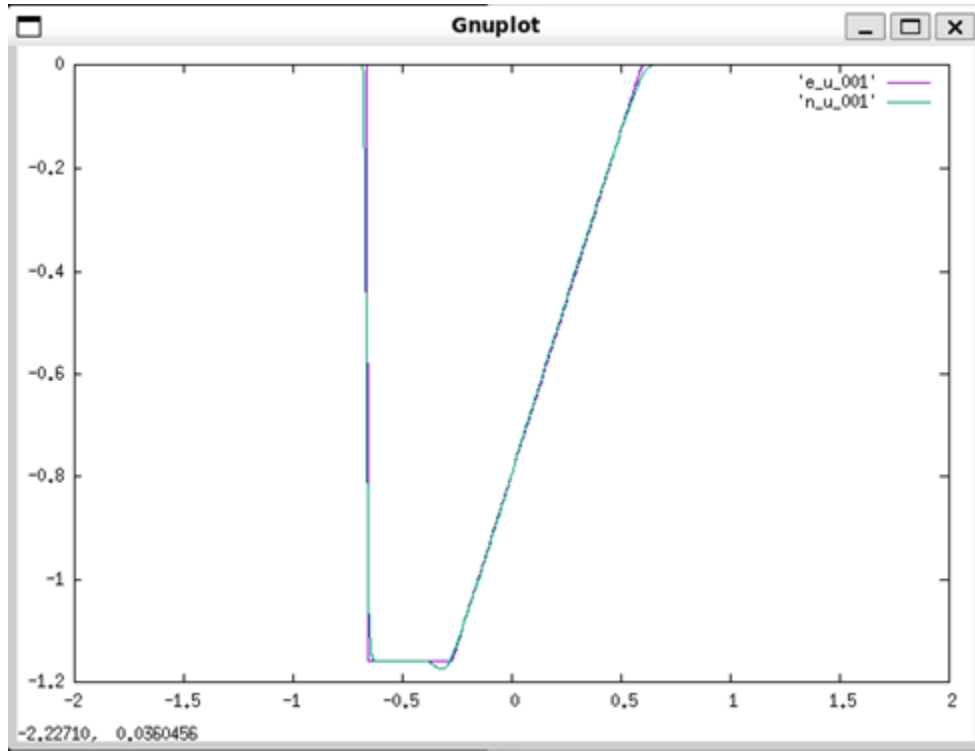


Figure 3.2: Numerical/exact comparison of velocity at $t = 0.5s$, uniform mesh of 400 cells

To test the code with a “dry” zone (Faccanoni and Mangeney (2013), test case 1), the same problem can for example be revisited with a different initial condition by modifying the **INIT** section of the data file *ex1_sv_Riemann.inp*:

```
INIT    0    0
0. 0. 0.
0.1446 0. 0.
```

This then gives:

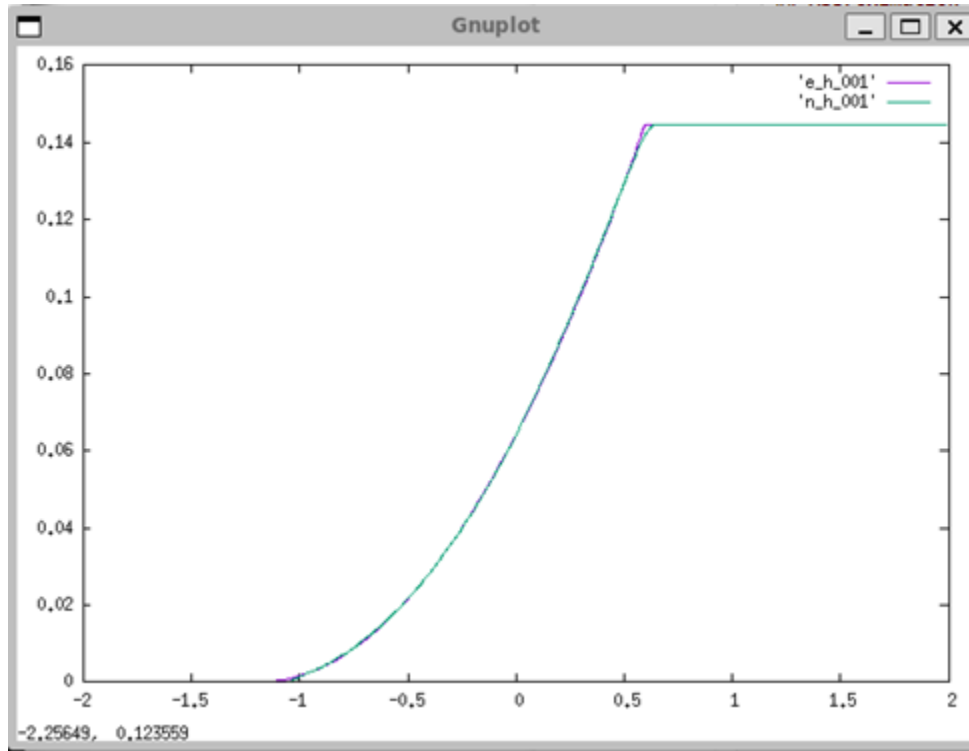


Figure 3.3: Numerical/exact comparison of water depth at $t = 0.5s$, uniform mesh of 400 cells with a dry zone

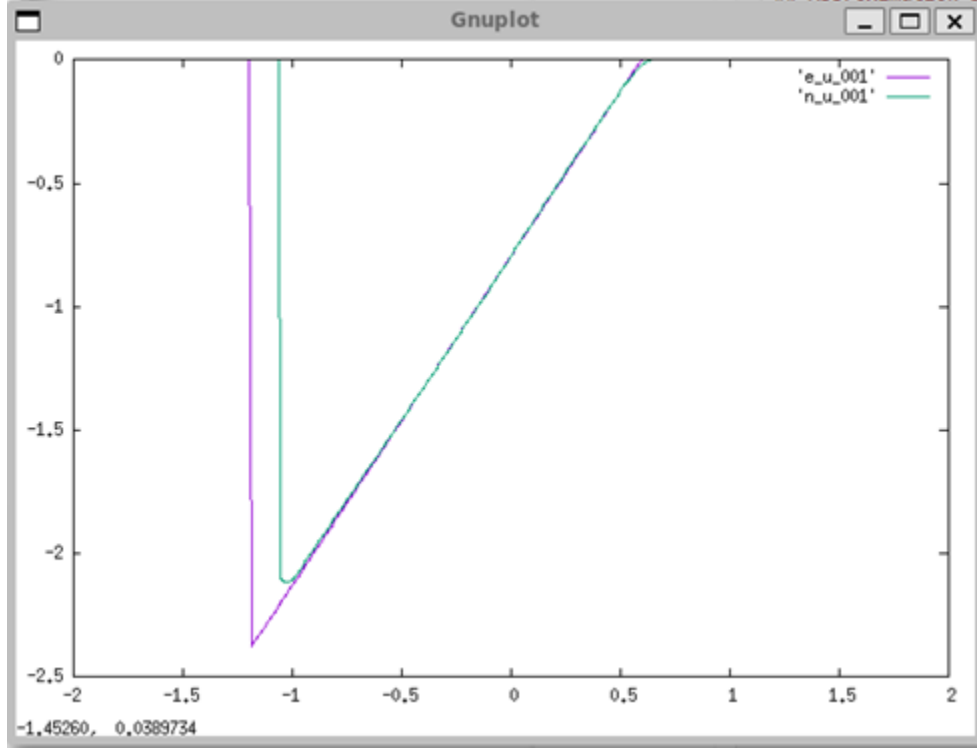


Figure 3.4: Numerical/exact comparison of velocity at $t = 0.5s$, uniform mesh of 400 cells with a dry zone

3.1.2 Hydraulic jump

We consider here a steady-state test case, without friction, with a non-flat bottom. The analytical solution is taken from Delestre et al. (2013). On a domain $x \in [0., 25.]$, we consider a topography defined by:

$$z_b(x) = \begin{cases} 0.2 - 0.05(x - 10)^2 & \text{if } 8 < x < 12 \\ 0. & \text{otherwise} \end{cases}$$

We consider the case of transcritical flow with a shock, characterized by the following boundary conditions:

- On the left, $h = 0.33m$ is imposed and the discharge is left free.
- On the right, $hu = 0.18m^2/s$ is imposed and the water depth is left free.

As recommended by Delestre et al. (2013), the initial conditions are given by: $h + z_b = 0$ and $u = 0$.

The computation is carried out on a fixed, regular mesh of 2500 cells distributed over 8 domains. In order to reach a steady state, the computation is run up to $T = 100s$. The data file corresponding to this test case is provided below Tip 4:

💡 Tip 4: Data file *ex2_sv_Bump.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----

! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference         (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)      (default 20.)
! REFA : Isothermal reference density for Air     (default 1.)
! REFW : Isothermal reference density for Water   (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                    (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level         (default 0.0001)
! FRMO : Friction Model
!       0/none, 1/Manning-Strickler, 2/Darcy-Weisbach (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----

PHYS 0
MODE 1.
FRMO 0.
BAGR 0.
!-----
```

```

! Reading the Master MESH
! Header  MAME
!-----
! code 1 -> creation of a mesh
!
!         argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!         argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!
!         argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME      1      0
0. 25.
2500
!-----
! Reading Initial conditions
! Header  INIT
!-----
! code 0 -> Definition for a shock tube
!         primitive variables on left
!         primitive variables on right
!         argument: 0
! code 1 -> user function
!         argument: Function's Number
! code 2 -> Definition using areas
!         argument: number of areas
!         and for each area 2 lignes
!         - xmin,xmax,ymin,ymax,zmin,zmax
!         - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      1      1

```

```

!-----
! Reading bathymetry and friction
! Header BATH
!-----
! code 0 -> Default : flat bottom and no friction

! code 1 -> bathymetry and friction defined by user function (defined in sources/PHY/fct_bath)
!           argument: Function's Number
! code 2 -> bathymetry and friction defined linearly per block
!           argument: 0   reading the file
!           argument:-1  creation of a file containing the coordinates of the vertices of the
! next line: file name
!-----
BATH      1      1

!-----
! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case "shock tube" : Nothing
!
! code 1 -> Description in the case "2d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case "3d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for GMSH mesh
!           argument: number of zones defining a boundary condition
!           number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----

```



```

COND      2      0
4
1.e20 0.18 0.
2
0.33 1.e20 0.
999
999
999
999

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS                      (default 001.)
! MARL : MArximal Refinement Level                (default 000.)
! COPA : Mesh COarsening PArameter                0<.. $<1$  (default 0.002)
! REPA : Mesh REfinement PArameter                0<.. $<1$  (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!         then for each zone
!         nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 8
NZRA 1
0 0. 25. -100. 100. -100. 100.

!-----
! Reading numerical parameter
! Header NUME
!-----
!
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute                  (default 0.)
! CCFL : cfl condition                             (default 0.9)

```

```

! TDOR : Time discretization order 1 ou 2      (default 1.)
! SDOR : Space discretization order 1 ou 2     (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian  (default 0.)
! SAVE : kind of backup, three-digit number    (default 001.)
!       decade : shock tube backup 0/no 1/yes
!       unit    : binary backup      0/no 1/yes
! NPRO : number of probe < 10                  (default 000.)
!       then coordinates x,y,z of the probe (1 probe per line)
! -----
NUME 0
TINT 100.
CCFL .9
SDOR 2.
TDOR 2.
SAVE 01.

```

To run the computation, since there is only one domain to process over a single time interval, but distributed over 8 domains, the following commands are used:

```

./cerf_input ex2_sv_Bump.inp
mpirun -np 8 ./cerf
./cerf2tec

```

The analytical solution to this problem can be obtained from the [Swashes](#) website. In the *doc* directory of **CERF**, the analytical solution of $\eta = h + z_b$ on 2500 points in Tecplot format is provided for reference. The numerical solution can then be compared to the exact solution in the figure below (Figure 3.5).

The numerical solution can also be visualized with *Visit* or *Paraview*:

In figure Figure 3.6, the isovalues of the water level are shown at $t = 100s$. Note that the mesh has been modified for post-processing purposes. The coordinates of the nodes at the base of the hexahedra correspond to z_b , while the nodes on the upper part of the hexahedra correspond to $z_b + h$. This makes it possible to visualize the topography and the water level at the same time.

This computation can also be performed using automatic mesh refinement. To do this, the data file is modified accordingly *ex2_sv_Bump.inp*. A mesh consisting of only 250 blocks is then created, and mesh refinement is allowed only up to level 2. All other computation parameters are kept the same as for the fixed mesh.

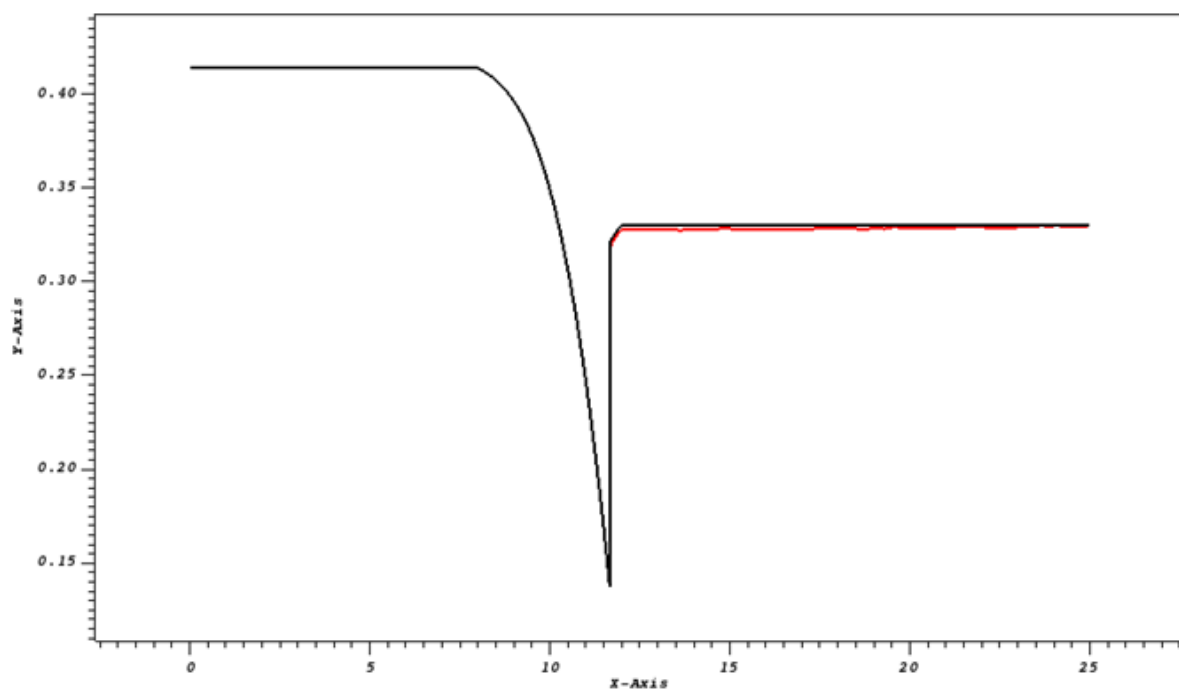


Figure 3.5: Numerical (red) / exact (black) comparison of water level at $t = 100.s$, uniform mesh of 2500 cells

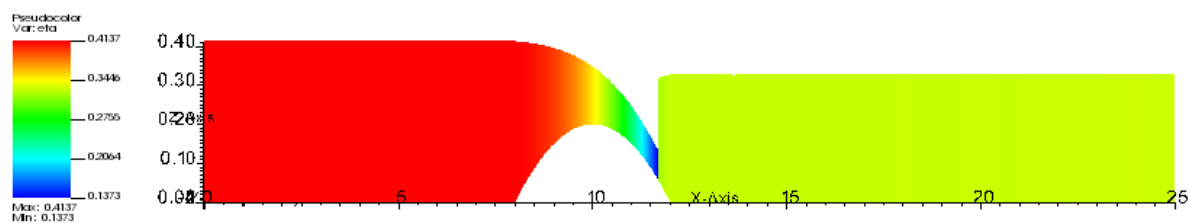


Figure 3.6: Water level isovalues at $t = 100.s$, uniform mesh of 2500 cells

💡 Tip 5: Data file *ex2_sv_Bump_amr.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----

! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference         (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)      (default 20.)
! REFA : Isothermal reference density for Air     (default 1.)
! REFW : Isothermal reference density for Water   (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                    (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level         (default 0.0001)
! FRMO : Friction Model
!       0/none, 1/Manning-Strickler, 2/Darcy-Weisbach (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----

PHYS 0
MODE 1.

!-----
! Reading the MAMesh
! Header MAME
!-----

! code 1 -> creation of a mesh
!
```

```

!           argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!           argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!
!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME      1      0
0. 25.
2500
!-----
! Reading Initial conditions
! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      1      1
!-----
! Reading bathymetry and friction
! Header BATH
!-----

```

```

! code 0 -> Default : flat bottom and no friction

! code 1 -> bathymetry and friction defined by user function (defined in sources/PHY/fct_bath)
!           argument: Function's Number
! code 2 -> bathymetry and friction defined linearly per block
!           argument: 0
! next line: file name
!
! code 3 -> bathymetry and friction defined per block at the highest mesh refinement level
!           argument: 0
! next line: file name
!-----
BATH      1      1

!-----
! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case "shock tube" : Nothing
!
! code 1 -> Description in the case "2d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case "3d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for GMSH mesh
!           argument: number of zones defining a boundary condition
!           number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      2      0

```

```

4
1.e20 0.18 0. 0. 0. 0.
2
0.33 1.e20 0. 0. 0. 0.
999
999
999
999

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS                      (default 001.)
! MARL : MAlximal Refinement Level              (default 000.)
! COPA : Mesh COarsening PArAmeter              0<.. $<1$  (default 0.002)
! REPA : Mesh REfinement PArAmeter              0<.. $<1$  (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!         then for each zone
!         nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 8
MARL 2
NZRA 2
0 0. 25. -100. 100. -100. 100.
2 7.8 12.1 -100. 100. -100. 100.

!-----
! Reading numerical parameter
! Header NUME
!-----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!

```

```

! TINT : Time interval to compute                (default 0.)
! CCFL : cfl condition                          (default 0.9)
! TDOR : Time discretization order 1 ou 2        (default 1.)
! SDOR : Space discretization order 1 ou 2        (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian     (default 0.)
! SAVE : kind of backup, three-digit number      (default 001.)
!         decade : shock tube backup 0/no 1/yes
!         unit    : binary backup    0/no 1/yes
! NPRO : number of probe < 10                    (default 000.)
!         then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT 10.
CCFL .9
SDOR 2.
TDOR 2.
SAVE 01.

```

To run the computation with mesh refinement, the following script is used:

```

#!/bin/bash
./cerf_input ex2_sv_Bump2.inp
./cerf2tec
mv cerfout.dat b_0.dat
mv cerfamr.dat b_mame_0.dat
for i in $(seq 1 20 )
do
mpirun -np 8 ./cerf
./cerf2tec
mv cerfout.dat b_${i}.dat
./cerf_amr
mv cerfamr.dat b_mame_${i}.dat
done

```

As before, the results can be compared with the exact solution.

The numerical solution can also be visualized with *Visit* or *Paraview*:

In both Figure 3.7 and Figure 3.8, it can be seen that mesh refinement made it possible to reduce the number of cells while maintaining good accuracy in the numerical solution.

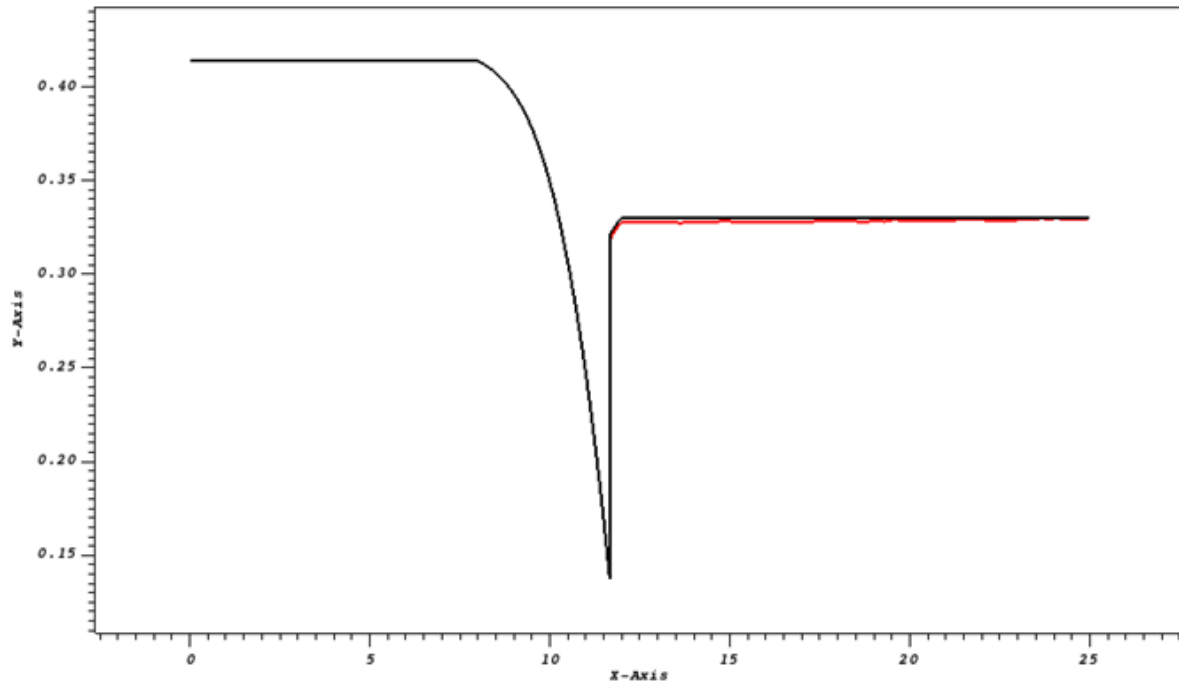


Figure 3.7: Numerical comparison on a fixed mesh (red) / numerical on a dynamic mesh (green) / exact (black) water level at $t = 100.s$

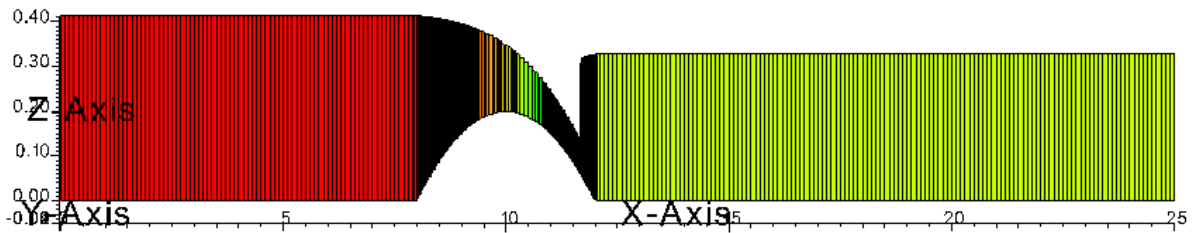


Figure 3.8: Water level isovalues, dynamic mesh of 344 cells at $t = 100.s$

3.1.3 Friction, McDonald test case

We consider here a steady-state test case, with friction, on a non-flat bottom. The semi-analytical solution is taken from Delestre et al. (2013) (case 3.2.1).

On a domain $x \in [0., 1000.]$, we consider a subcritical flow with a constant discharge at the inlet of $q = hu = 2m^2/s$ and a prescribed water depth at the outlet $h_e(1000)$. The domain is initially dry, with $h = 0$ and $u = 0$. A Manning friction with $n = 0.033$ is imposed.

The water depth is given by:

$$h_{ex}(x) = \left(\frac{4}{g}\right)^{1/3} \left(1 + \frac{1}{2} \exp\left(-16 \left(\frac{x}{1000} - \frac{1}{2}\right)\right)\right)$$

Using Equation 1.6, in the steady-state case, the topography can then be recovered by integrating:

$$\frac{\partial z_b}{\partial x} = \left(\frac{q^2}{gh^3} - 1\right) \frac{\partial h}{\partial x} - S_f$$

The computation is carried out on a fixed, regular mesh of 1000 cells. In order to reach a steady state, the computation is run up to $T = 1500s$. The data file corresponding to this test case is provided below Tip 6:

💡 Tip 6: Data file *ex3_sv_McDo.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----
! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!----- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
```

```

! PREF : isothermal pressure of reference          (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)       (default 20.)
! REFA : Isothermal reference density for Air      (default 1.)
! REFW : Isothermal reference density for Water    (default 1000.)
! SHAR : isothermal interface sharpening parameter
!           recommended value 0.01                  (default 0.)
!----- for Shallow water model
! WLTR : Value to truncate the Water level         (default 0.0001)
! FRMO : Friction Model
! 0/none, 1/Manning-Strickler, 2/Darcy-Weisbach    (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----
PHYS 0
MODE 1.
FRMO 1.

!-----
! Reading the MasteR MESH
! Header  MAME
!-----
! code 1 -> creation of a mesh
!
!           argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!           argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!
!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME      1      0
0. 1000.
1000
!-----
! Reading Initial conditions

```

```

! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      2      1
-1. 1001. -100. 100. -100. 100.
0. 0. 0.

!-----
! Reading bathymetry and friction
! Header BATH
!-----
! code 0 -> Default : flat bottom and no friction

! code 1 -> bathymetry and friction defined by user function (defined in sources/PHY/fct_bath)
!           argument: Function's Number
! code 2 -> bathymetry and friction defined linearly per block
!           argument: 0  reading the file
!           argument:-1  creation of a file containing the coordinates of the vertices of the
! next line: file name
!-----

BATH      1      2
!-----
! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case  "shock tube" : Nothing

```

```

!
! code 1 -> Description in the case "2d box" (at least 1 line per type) :
!         kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case "3d box" (at least 1 line per type) :
!         kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for GMSH mesh
!         argument: number of zones defining a boundary condition
!         number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      2      0
4
1.e20 2. 0.
2
0.74832355831838937 1.e20 0. 0.
999
999
999
999

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS (default 001.)
! MARL : MArimal Refinement Level (default 000.)
! COPA : Mesh COarsening Parameter 0<.. $1$  (default 0.002)

```

```

! REPA : Mesh REfinement PArameter          0<.. $<1$  (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!       then for each zone
!       nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 1
MARL 0
NZRA 1
0 0. 1000. -100. 100. -100. 100.

!-----
! Reading numerical parameter
! Header  NUME
!-----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute              (default 0.)
! CCFL : cfl condition                        (default 0.9)
! TDOR : Time discretization order 1 ou 2      (default 1.)
! SDOR : Space discretization order 1 ou 2     (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian  (default 0.)
! SAVE : kind of backup, three-digit number    (default 001.)
!       decade : shock tube backup 0/no 1/yes
!       unit    : binary backup    0/no 1/yes
! NPRO : number of probe < 10                 (default 000.)
!       then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT 50.
CCFL .9
SDOR 1.
TDOR 1.
SAVE 01.

```

To run the computation, in order to visualize the evolution towards the steady state, the computation time is split into 30 steps using the script below:

💡 Script *ex3_sv_McDo.sh*

```
#!/bin/bash
./cerf_input ex3_sv_McDo.inp
./cerf2tec
mv cerfout1d.dat md_0.dat
for i in $(seq 1 30 )
do
./cerf
./cerf2tec
mv cerfout1d.dat md_${i}.dat
done
```

The analytical solution to this problem can be obtained from the [Swashes](#) website. In the *doc* directory of **CERF**, the analytical solution on 10000 points in Tecplot format is provided for reference. The numerical solution can then be compared to the exact solution in the figure below (Figure 3.9).

3.2 Examples with the Eulerian two-fluid model

In this chapter, we present the implementation of a few classic test cases for two-fluid flows. The cases covered are as follows:

- Solving a Riemann problem
- Dam break
- A cylinder falling onto a body of water
- ...

3.2.1 Riemann problem

In order to validate the implementation of the model, we consider the classic “shock tube” test case with the isothermal two-fluid model. We use the test case from Frédéric Golay and Helluy (2007). The domain is defined by $x \in [-2., 2.]$ and $t \in [0, 0.25]$. The initial conditions are given by:

$$\overrightarrow{w}_p(x, 0) = \begin{cases} \overrightarrow{w}_{p_L} = \langle \rho_L, u_L, p_L \rangle^T = \langle 0.0046, 0, 1.5 \cdot 10^5 \rangle^T & \text{if } x < 0 \\ \overrightarrow{w}_{p_R} = \langle \rho_R, u_R, p_R \rangle^T = \langle 0.1446, 0., 1. \cdot 10^5 \rangle^T & \text{if } x > 0 \end{cases}$$

We consider a uniform, fixed 1D mesh of 400 cells over a single domain. The data file corresponding to this test case is provided below:

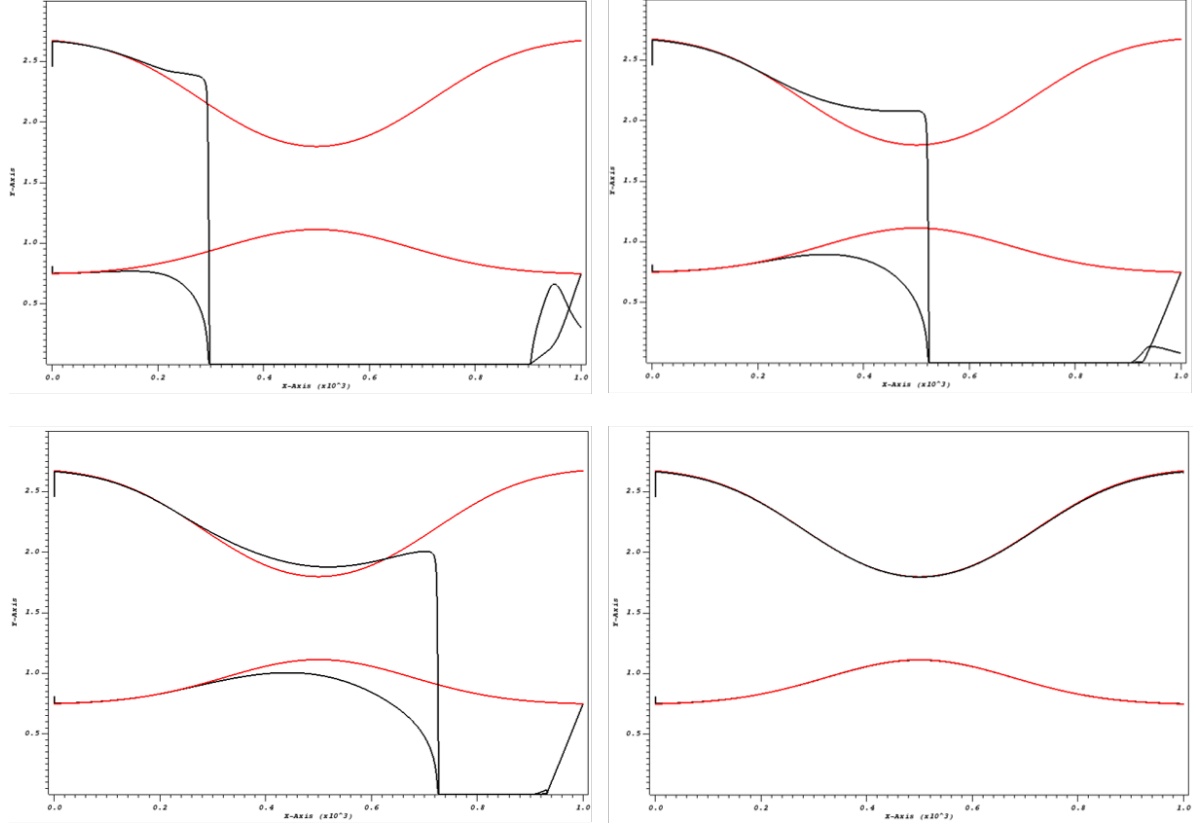


Figure 3.9: Numerical (black) / exact (red) comparison of the water level h (top) and velocity u (bottom), with a uniform mesh of 1000 cells at first order in space and time, at times $t = 100s$ (top left), $t = 200s$ (top right), $t = 300s$ (bottom left), $t = 1500s$ (bottom right)

💡 Tip 7: Data file *ex1_bf_Riemann.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----

! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference         (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)      (default 20.)
! REFA : Isothermal reference density for Air     (default 1.)
! REFW : Isothermal reference density for Water   (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                    (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level        (default 0.0001)
! FRMO : Friction Model
!       0/none, 1/Manning-Strickler, 2/Darcy-Weisbach (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----

PHYS 0
MODE 2.

!-----
! Reading the MAMaster MESH
! Header MAME
!-----

! code 1 -> creation of a mesh
```

```

!
!           argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!           argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!
!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
NAME      1      0
-10. 10.
400
!-----
! Reading Initial conditions
! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      0      0
1125. 0. 1.5e5
1. 0. 1.e5
!-----

```

```

! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case "shock tube" : Nothing
!
! code 1 -> Description in the case "2d box" (at least 1 line per type) :
!         kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case "3d box" (at least 1 line per type) :
!         kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for GMSH mesh
!         argument: number of zones defining a boundary condition
!         number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      0      0

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS (default 001.)
! MARL : MArimal Refinement Level (default 000.)
! COPA : Mesh COarsening PARAMeter 0<..<1 (default 0.002)
! REPA : Mesh REfinement PARAMeter 0<..<1 (default 0.02)

```

```

! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!       then for each zone
!       nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 1
MARL 0
NZRA 1
0 -10. 10. -100. 100. -100. 100.

!-----
! Reading numerical parameter
! Header  NUME
!-----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute (default 0.)
! CCFL : cfl condition (default 0.9)
! TDOR : Time discretization order 1 ou 2 (default 1.)
! SDOR : Space discretization order 1 ou 2 (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian (default 0.)
! SAVE : kind of backup, three-digit number (default 001.)
!       decade : shock tube backup 0/no 1/yes
!       unit : binary backup 0/no 1/yes
! NPRO : number of probe < 10 (default 000.)
!       then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT 0.25
CCFL .9
SDOR 2.
TDOR 2.
SAVE 11.

```

To run the computation, since there is only one domain to process over a single time interval:

```

./cerf_input ex1_bf_Riemann.inp
./cerf

```

Since the data file *ex1_bf_Riemann.inp* requests output in 1D format (parameter **SAVE**=11, section **NUME**), 1D output files have been generated. Files for the numerical solution in density, velocity, and pressure *'n_r_001'*, *'n_u_001'*, and *'n_p_001'*, and files for the exact solution in density, velocity, and pressure: *'e_r_001'*, *'e_u_001'*, and *'e_p_001'*. These files can be visualized, for example, with gnuplot:

```
gnuplot
gnuplot> set term x11
gnuplot> plot 'e_r_001' w l, 'n_r_001' w l
gnuplot> plot 'e_u_001' w l, 'n_u_001' w l
gnuplot> plot 'e_p_001' w l, 'n_p_001' w l
gnuplot> quit
```

The numerical solution can thus be compared to the exact solution.

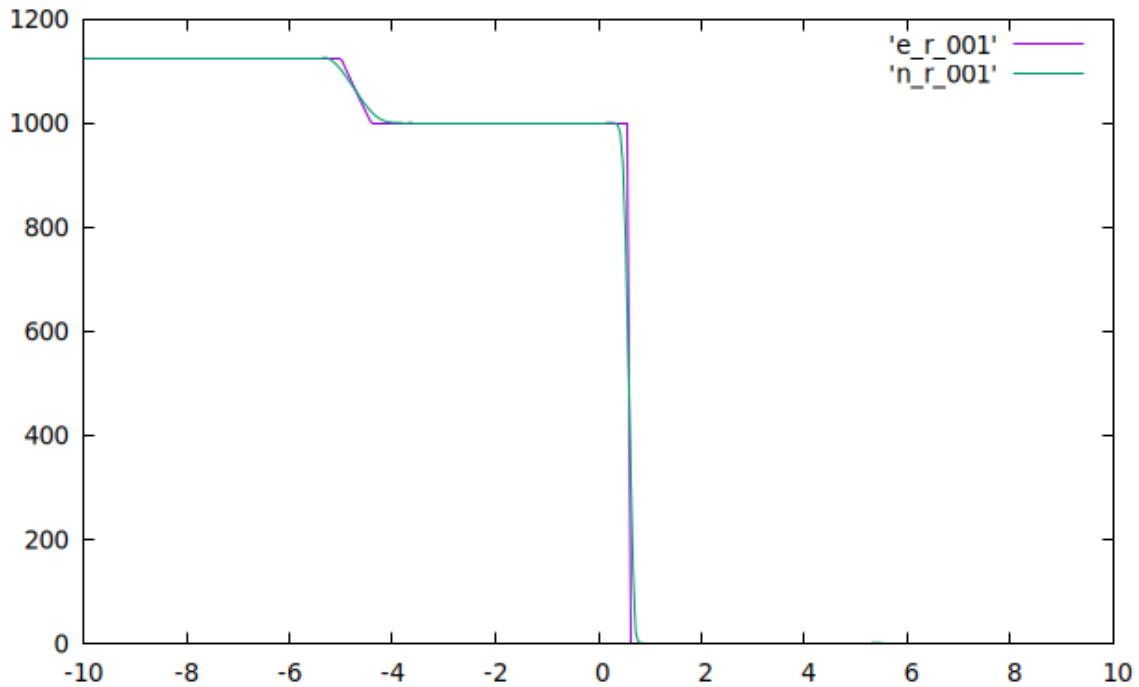


Figure 3.10: Numerical/exact comparison of density at $t = 0.25s$, uniform mesh of 400 cells

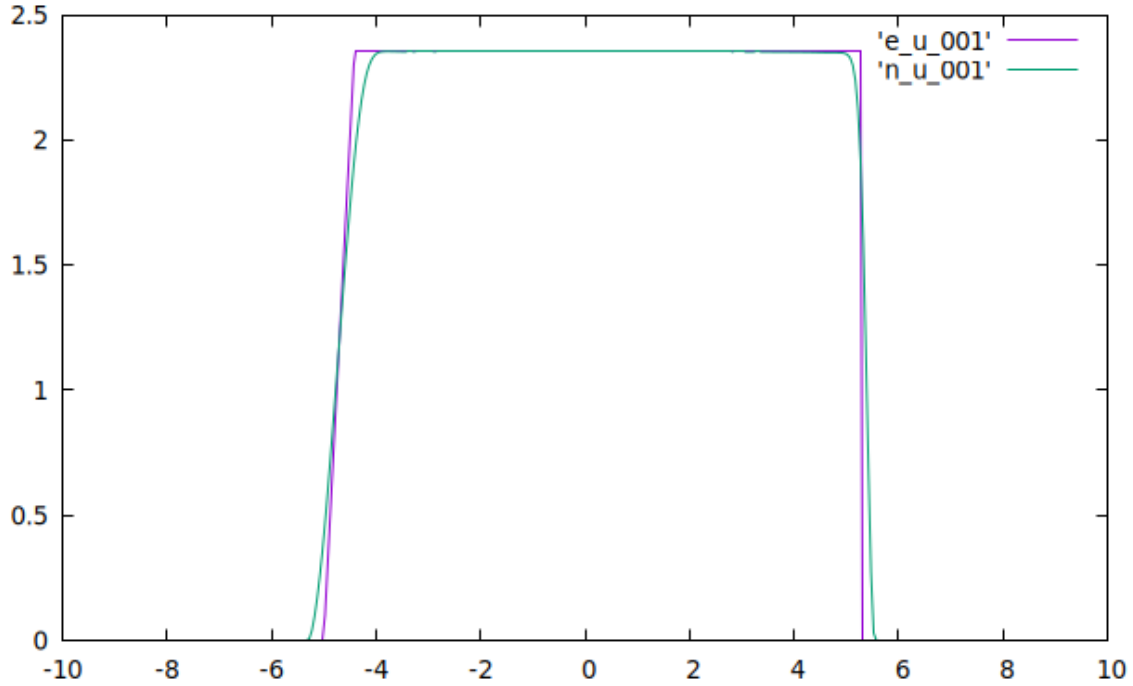


Figure 3.11: Numerical/exact comparison of velocity at $t = 0.25s$, uniform mesh of 400 cells

3.2.2 Dam break

We consider here a 2D dam-break test case. This is a widely used and well-documented case. From an experimental standpoint, one can rely for example on the experiments carried out by Martin and Moyce (1952). We consider a computational domain $x \in [0., 4.]$ and $y \in [0., 3.]$. The domain boundaries are characterized by “mirror”-type or zero-Neumann boundary conditions. At the initial instant, the domain consists of air, except for a region defined by $x \in [0., 2.]$ and $y \in [0., 3.]$ where water is present. The master mesh consists of 40×30 blocks, which are refined to level 4 around the air/water interface. To improve the quality of the interface, the computation is penalized using an interface stiffening coefficient of 0.02.

The data file corresponding to this test case is provided below:

💡 Tip 8: Data file *ex2_bf_Dam.inp*

```
!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
```

```

! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----
! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference         (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)      (default 20.)
! REFA : Isothermal reference density for Air     (default 1.)
! REFW : Isothermal reference density for Water   (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                    (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level        (default 0.0001)
! FRMO : Friction Model
! 0/none, 1/Manning-Strickler, 2/Darcy-Weisbach   (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----
PHYS 0
MODE 3.
GPES 2
SHAR 0.02
!-----
! Reading the Master MESH
! Header MAME
!-----
! code 1 -> creation of a mesh
!
!       argument: 0 type "1D shock tube"
! next line: xmin,xmax
! next line: nx
!
!       argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax

```

```

! next line: nx,ny
!
!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME      1      1
0. 4.  0. 3.
40 30
!-----
! Reading Initial conditions
! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      2      2
-1. 4.1 -1. 3.1 -1. 1.
1. 0. 0. 0.  1.e5 0.
-1. 1. -1. 2. -1. 1.
1000. 0. 0. 0.  1.e5 0.
!-----
! Reading boundary condition
! (maximum 10 for the moment)
! Header COND
!-----
!

```



```

! code 0 -> Description in the case  "shock tube" : Nothing
!
! code 1 -> Description in the case  "2d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case  "3d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for  GMSH mesh
!           argument: number of zones defining a boundary condition
!           number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables  (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      1      0
1
1
1
1

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS                                (default 001.)
! MARL : MAximal Refinement Level                          (default 000.)
! COPA : Mesh COarsening PArameter                        0<..<1 (default 0.002)
! REPA : Mesh REfinement PArameter                        0<..<1 (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)

```

```

!           then for each zone
!           nrb,xmin,xmax,ymin,ymax,zmin,zmax
!-----
MESH 0
NBDS 8
MARL 4
COPA 0.9
REPA 3
NZRA 3
0 -1. 4.1 -1. 3.1 -1. 1.
4 -1. 1.5 -1. 2.25 -1. 1.
0 -1. .75 -1. 1.75 -1. 1.

!-----
! Reading numerical parameter
! Header  NUME
!-----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute                (default 0.)
! CCFL : cfl condition                            (default 0.9)
! TDOR : Time discretization order 1 ou 2          (default 1.)
! SDOR : Space discretization order 1 ou 2          (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian      (default 0.)
! SAVE : kind of backup, three-digit number        (default 001.)
!         decade : shock tube backup 0/no 1/yes
!         unit    : binary backup    0/no 1/yes
! NPRO : number of probe < 10                      (default 000.)
!         then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT 0.01
CCFL .9
SDOR 2.
TDOR 2.
SAVE 1.

```

The expected velocities are on the order of a few meters per second. Therefore, since the blocks have a size of $0.1m$, the time interval between each remeshing can be estimated to be $0.01s$.

To speed up the computation, the mesh is decomposed into 8 domains (or equivalently 8 MPI processes). To run a computation up to $T = 0.8s$, the following script can be used:

```
#!/bin/bash
./cerf_input ex2_bf_Dam.inp
./cerf2tec
mv cerfout.dat d_0.dat
mv cerfamr.dat d_mame_0.dat
for i in $(seq 1 80 )
do
mpirun -np 8 ./cerf
./cerf2tec
mv cerfout.dat d_${i}.dat
./cerf_amr
mv cerfamr.dat d_mame_${i}.dat
done
```

In the images below Figure 3.12, the density of the mixture is shown (blue for water and red for air). The interface can be seen to be “*sharp*”, i.e., spread over 3-4 cells.

3.2.3 A cylinder falling onto a body of water

We consider here a 2D test case of a solid object falling onto a water surface. This fluid-structure interaction problem is treated here using CERF’s “rigid-fluid” tool (Frederic Golay (2018)). This is a widely used and well-documented case (see for example H. Sun and Faltinsen (2006), P. Sun, Ming, and Zhang (2015)). From an experimental standpoint, one can rely for example on the experiments carried out by Greenhow and Lin (2015). We consider a computational domain $x \in [0., 2.]$ and $y \in [0., 2.]$. The domain boundaries are characterized by “mirror”-type or zero-Neumann boundary conditions. At the initial instant, the domain consists of air, except for a region defined by $x \in [0., 2.]$ and $y \in [0., 0.65]$ where water is present. The rigid solid is defined by a mesh in .OBJ format. The cells inside this envelope behave as a free rigid solid, subject to gravity, with a density of $500kg/m^3$. The master mesh consists of 40×40 blocks, which are refined to level 4 around the air/water interface and around the cylinder, whose center is positioned $0.5m$ from the surface of the water, as illustrated in figure Figure 3.13. To improve the quality of the interface, the computation is penalized using an interface stiffening coefficient of 0.01.

The data file corresponding to this test case is provided below:

💡 Tip 9: Data file *ex3_bf_cyl.inp*

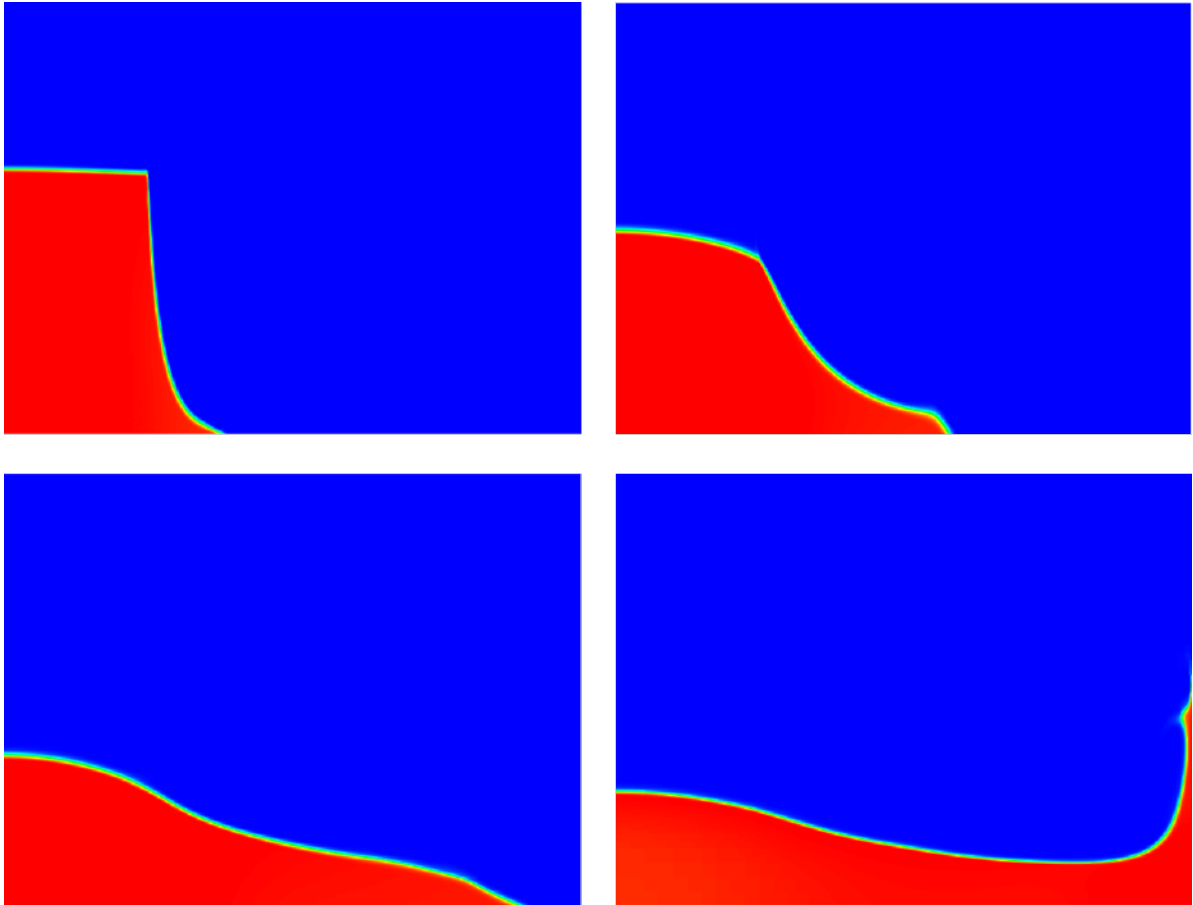


Figure 3.12: Dam break, density at $t = 0.2s, 0.4s, 0.6s, 0.8s$

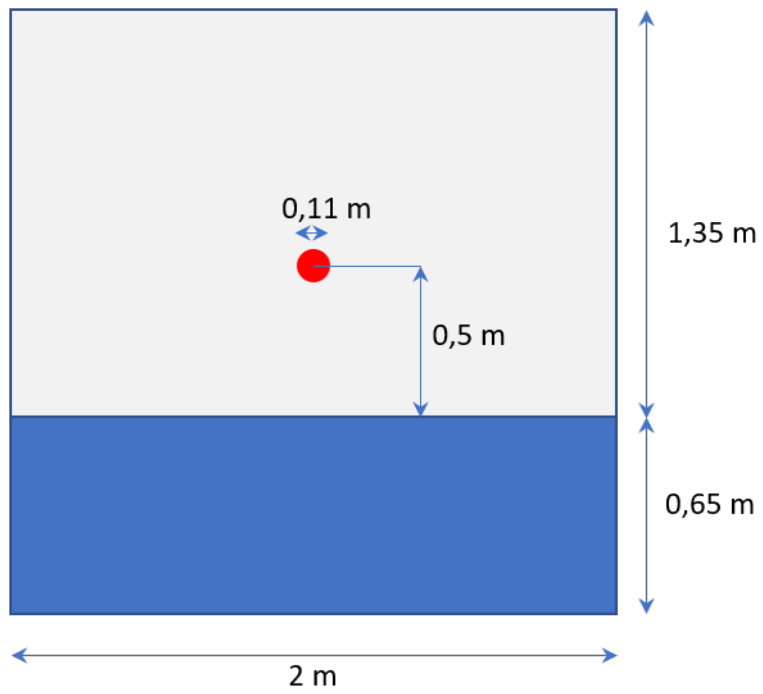


Figure 3.13: Description of the test case: a cylinder falling onto a water surface

```

!-----
! Data file in CERF format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! PHYS MAME INIT BATH COND MESH OBST NUME
!-----

!-----
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a code (1 integer)
! for now we have the same values for all domains
!-----

! MODE : physical model
!       1/ Shallow water
!       2/ Euler isothermal two-phases           (default 0.)
!---- for isothermal Euler model
! GPES : Gravity Orientation 0/none, 1/x, 2/y, 3/z (default 2.)
! PREF : isothermal pressure of reference         (default 1.d5)
! CSSM : Mixture sound velocity (isothermal)      (default 20.)
! REFA : Isothermal reference density for Air     (default 1.)
! REFW : Isothermal reference density for Water   (default 1000.)
! SHAR : isothermal interface sharpening parameter
!       recommended value 0.01                    (default 0.)
!---- for Shallow water model
! WLTR : Value to truncate the Water level        (default 0.0001)
! FRMO : Friction Model
! 0/none, 1/Manning-Strickler, 2/Darcy-Weisbach   (default 0.)
! BAGR : Bathymetry gradient 0/ Given , 1/ numerical approximation
!-----

PHYS 0
MODE 2.
GPES 2
SHAR 0.01

!-----
! Reading the MAsTer MESH
! Header  MAME
!-----

! code 1 -> creation of a mesh
!
!       argument: 0 type "1D shock tube"
! next line: xmin,xmax

```

```

! next line: nx
!
!           argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax
! next line: nx,ny
!
!           argument: 2 type "3d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nx,ny,nz
!
! code 2 -> Reading an ascii file format GMSH V2.2
! next line: file name
!-----
MAME      1      1
0. 2.  0. 2.
40 40
!-----
! Reading Initial conditions
! Header INIT
!-----
! code 0 -> Definition for a shock tube
!           primitive variables on left
!           primitive variables on right
!           argument: 0
! code 1 -> user function
!           argument: Function's Number
! code 2 -> Definition using areas
!           argument: number of areas
!           and for each area 2 lignes
!           - xmin,xmax,ymin,ymax,zmin,zmax
!           - nvar values ( primitive variables)
! Shallow water model: water level, x velocity, y velocity
! Air-water flow model: density, x velocity, y velocity,z velocity, pressure
!-----
INIT      2      2
-1. 2.1 -1. 2.1 -1. 1.
1. 0. 0. 0.  1.e5 0.
-1. 2. -1. 0.65 -1. 1.
1000. 0. 0. 0.  1.e5 0.
!-----
! Reading boundary condition

```

```

! (maximum 10 for the moment)
! Header COND
!-----
!
! code 0 -> Description in the case "shock tube" : Nothing
!
! code 1 -> Description in the case "2d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax
!
! code 2 -> Description in the case "3d box" (at least 1 line per type) :
!           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
!
! code 3 -> Definition per zone for GMSH mesh
!           argument: number of zones defining a boundary condition
!           number of zone, kind of boundary condition
!
! kind of boundary condition:
! 0 outlet (we copy)
! 1 mirror
! 2 Dirichlet condition (if 1.e20 then copy), then next line nvar values
! 3 user's function , then next line function number
! 4 Dirichlet condition on conservative variables (if 1.e20 then copy),
!   then next line nvar values
! 999 Non used
!-----
COND      1      0
1
1
1
1

!-----
! Reading mesh parameters in order to define the mesh according to
! the Master mesh is the one represented by the domain "0".
! Header MESH
!-----
!
! code 0 -> default value
!
! 1 keyword (4 characters) and a real
! NBDS : NumBer of DomainS (default 001.)
! MARL : MArximal Refinement Level (default 000.)

```



```

! COPA : Mesh COarsening PArameter          0<.. $<1$  (default 0.002)
! REPA : Mesh REfinement PArameter          0<.. $<1$  (default 0.02)
! FDRA : Function defining the mesh refinement to be applied
! NZRA : Number of zones where initial blocks are refined (default 1.)
!         then for each zone
!         nrb,xmin,xmax,ymin,ymax,zmin,zmax
! -----
MESH 0
NBDS 10
MARL 4
COPA 1
REPA 2
NZRA 4
0 -1. 2.1 0. 2.1 -1. 1
4 -1. 2.1 0.6 0.7 -1. 1
0 -1. 2.1 0. 0.6 -1. 1
4 0.9 1.1 1.0 1.3 -1. 1

! -----
! Reading rigid obstacle parameters
! Header OBST
! -----
!
! code 0 -> default value
!         argument: number of obstacles
! 1 line per obstacle: file name, density, kind of motion
! ( -1: fixed, 0: free, >0 : function number imposing the movement
!
! -----
OBST 0 1
ex3_bf_cyl.obj 500. 0

! -----
! Reading numerical parameter
! Header NUME
! -----
! 1 keyword (4 characters) then a real
! code 0 -> default value
!
! TINT : Time interval to compute          (default 0.)
! CCFL : cfl condition                     (default 0.9)
! TDOR : Time discretization order 1 ou 2  (default 1.)

```

```

! SDOR : Space discretization order 1 ou 2          (default 1.)
! LIM1 : kind of limiter 0/Barth, 1/Cartesian      (default 0.)
! SAVE : kind of backup, three-digit number        (default 001.)
!       decade : shock tube backup 0/no 1/yes
!       unit    : binary backup    0/no 1/yes
! NPRO : number of probe < 10                      (default 000.)
!       then coordinates x,y,z of the probe (1 probe per line)
!-----
NUME 0
TINT 0.01
CCFL .9
SDOR 2.
TDOR 2.
TIME 1
SAVE 1.

```

The expected velocities are on the order of a few meters per second. Therefore, since the blocks have a size of $0.05m$, the time interval between each remeshing can be estimated to be $0.01s$. To speed up the computation, the mesh is decomposed into 10 domains (or equivalently 10 MPI processes). To run a computation up to $T = 2s$, the following script can be used:

```

#!/bin/bash
./cerf_input ex3_bf_cyl.inp
./cerf2tec
mv cerfout.dat c_000.dat
mv cerfamr.dat c_mame_000.dat
cp ex3_bf_cyl.obj cs_000.obj

for i in $(seq 1 9 )
do
mpirun -np 10 ./cerf
./cerf2tec
mv cerfout.dat c_00$i.dat
./cerf_amr
mv cerfamr.dat c_mame_00$i.dat
mv solide_001.obj cs_00$i.obj
done
for i in $(seq 10 99 )
do
mpirun -np 10 ./cerf

```

```

./cerf2tec
mv cerfout.dat c_0$i.dat
./cerf_amr
mv cerfamr.dat c_mame_0$i.dat
mv solide_001.obj cs_0$i.obj
done
for i in $(seq 100 200 )
do
mpirun -np 10 ./cerf
./cerf2tec
mv cerfout.dat c_$i.dat
./cerf_amr
mv cerfamr.dat c_mame_$i.dat
mv solide_001.obj cs_$i.obj
done

```

In the images below, the density of the mixture is shown (blue for water and red for air). The interface can be seen to be “*sharp*”, i.e., spread over 3-4 cells, and the dynamic mesh can be seen to “follow” the regions of interest closely. The first figure Figure 3.14, at $t = 0.3s$, corresponds to the instant when the cylinder touches the water surface at a velocity of about $3m/s$, which is consistent with theory ($\sqrt{2gh}$) and with experiment. The second figure Figure 3.15, at $t = 0.45s$, illustrates the generation of the wave created as the cylinder enters the water.

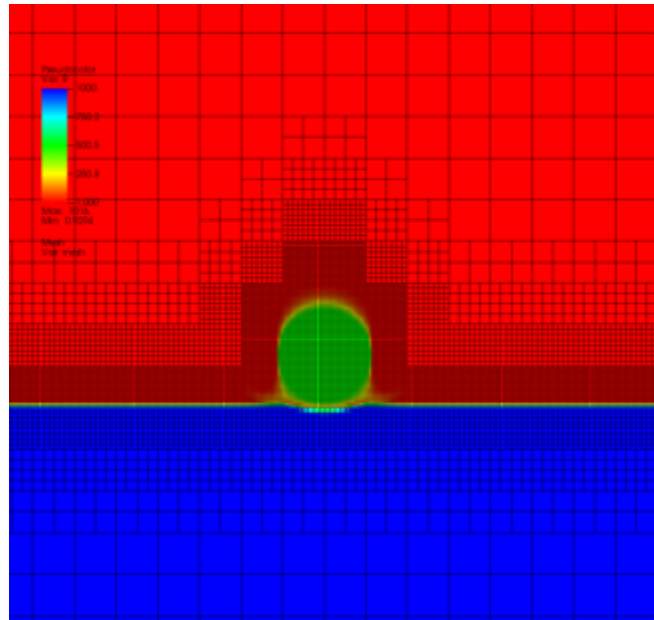


Figure 3.14: Cylinder fall, density and mesh at $t = 0.3s$

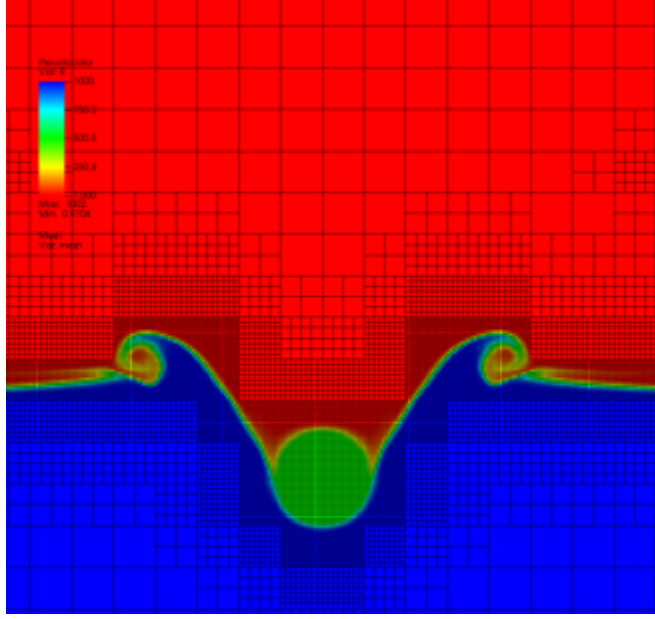


Figure 3.15: Cylinder fall, density and mesh at $t = 0.45s$

In the figure below Figure 3.16, the vertical trajectory of the cylinder over time is compared with the experimental results of Greenhow and Lin (2015). Good agreement is observed between the two approaches.

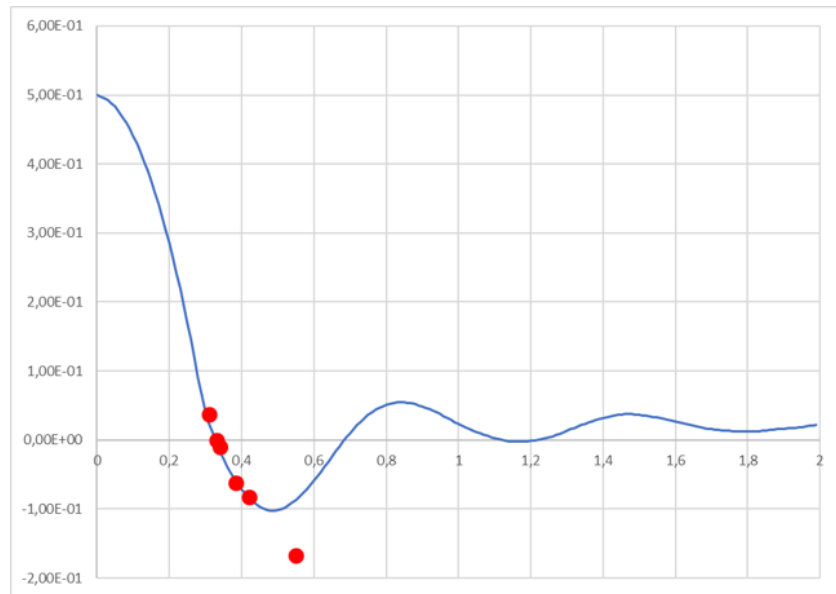


Figure 3.16: Vertical trajectory of the cylinder over time: comparison between the numerical simulation (solid line) and the experiment (dots)

Références

- Altazin, Thomas. 2017. “Un Modèle d’interaction Fluide-Structure En Régime Compressible Faible Mach.” PhD thesis. <http://www.theses.fr/2017TOUL0021/document>.
- Audusse, Emmanuel, François Bouchut, Marie-Odile Bristeau, Rupert Klein, and Benoît Perthame. 2004. “A Fast and Stable Well-Balanced Scheme with Hydrostatic Reconstruction for Shallow Water Flows.” *SIAM Journal on Scientific Computing* 25 (6): 2050–65. <https://doi.org/10.1137/S1064827503431090>.
- Chorin, Alexandre Joel. 1967. “A Numerical Method for Solving Incompressible Viscous Flow Problems.” *Journal of Computational Physics* 2 (1): 12–26. [https://doi.org/https://doi.org/10.1016/0021-9991\(67\)90037-X](https://doi.org/https://doi.org/10.1016/0021-9991(67)90037-X).
- Coquerelle, M., and G.-H. Cottet. 2008. “A Vortex Level Set Method for the Two-Way Coupling of an Incompressible Fluid with Colliding Rigid Bodies.” *Journal of Computational Physics* 227 (21): 9121–37. <https://doi.org/https://doi.org/10.1016/j.jcp.2008.03.041>.
- Croisille, Jean-Pierre. 1990. “Contribution à l’étude Théorique Et à l’approximation Par Éléments Finis Du Système Hyperbolique de La Dynamique Des Gaz Multidimensionnelle Et Multiespèces.” PhD thesis. <http://www.theses.fr/1990PA066095>.
- Delestre, Olivier, Carine Lucas, Pierre-Antoine Ksinant, Frédéric Darboux, Christian Laguerre, T.-N.-Tuoi Vo, François James, and Stéphane Cordier. 2013. “SWASHES: A Compilation of Shallow Water Analytic Solutions for Hydraulic and Environmental Studies.” *International Journal for Numerical Methods in Fluids* 72 (3): 269–300. <https://doi.org/https://doi.org/10.1002/fld.3741>.
- Ersoy, Mehmet, Frédéric Golay, and Lyudmyla Yushchenko. 2013. *Open Mathematics* 11 (8): 1392–1415. <https://doi.org/doi:10.2478/s11533-013-0252-6>.
- Faccanoni, Gloria, and Anne Mangeney. 2013. “Exact Solution for Granular Flows.” *International Journal for Numerical and Analytical Methods in Geomechanics* 37 (10): 1408–33. <https://doi.org/https://doi.org/10.1002/nag.2124>.
- Golay, Frederic. 2018. “Monolithic Fluid Structure Interaction Model, Application to Water Entry Problem.” In *Topical Problems of Fluid Mechanics 2018*. Prague, Czech Republic: Institute of Thermomechanics AS CR v.v.i. <https://doi.org/10.14311/TPFM.2018.016>.
- Golay, Frédéric. 2009. “Numerical Entropy Production and Error Indicator for Compressible Flows.” *Comptes Rendus. Mécanique* 337 (4): 233–37. <https://doi.org/10.1016/j.crme.2009.04.004>.
- Golay, Frédéric, and Philippe Helluy. 2007. “Numerical Schemes for Low Mach Wave Breaking.” *International Journal of Computational Fluid Dynamics* 21 (2): 69–86. <https://doi.org/10.1080/10618560701343382>.

- Greenhow, Martin, and Lin. 2015. “Greenhow m. And Lin w-m.”nonlinear Free Surface Effects: Experiments and Theory” MIT Internal Report 83-19 (1983),” June. <https://doi.org/10.13140/RG.2.1.4129.2964>.
- Helluy, Philippe, Golay, Frédéric, Caltagirone, Jean-Paul, Lubin, Pierre, Vincent, Stéphane, Drevard, Deborah, Marcer, Richard, et al. 2005. “Numerical Simulations of Wave Breaking.” *ESAIM: M2AN* 39 (3): 591–607. <https://doi.org/10.1051/m2an:2005024>.
- Lax, Peter. 1971. “Shock Waves and Entropy.” In *Contributions to Nonlinear Functional Analysis*, edited by Eduardo H. Zarantonello, 603–34. Academic Press. <https://doi.org/https://doi.org/10.1016/B978-0-12-775850-3.50018-2>.
- Marche, Fabien. 2007. “Derivation of a New Two-Dimensional Viscous Shallow Water Model with Varying Topography, Bottom Friction and Capillary Effects.” *European Journal of Mechanics - B/Fluids* 26 (1): 49–63. <https://doi.org/https://doi.org/10.1016/j.euromechflu.2006.04.007>.
- Martin, J. C., and W. J. Moyce. 1952. “Part IV. An Experimental Study of the Collapse of Liquid Columns on a Rigid Horizontal Plane.” *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 244 (882): 312–24. <http://www.jstor.org/stable/91519>.
- Pons, Kevin. 2018. “Dynamique Des Écoulements à Surface Libre Dans Les Plages Sableuses.” Thèse de doctorat, IMATH, Université de Toulon: Université de Toulon.
- Poussel, Camille. 2024. “Dynamique Des Écoulements à Surface Libre Dans Les Plages Sableuses.” Thèse de doctorat, IMATH, Université de Toulon: Université de Toulon.
- Puppo, Gabriella. 2004. “Numerical Entropy Production for Central Schemes.” *SIAM Journal on Scientific Computing* 25 (4): 1382–1415. <https://doi.org/10.1137/S1064827502386712>.
- Saint-Venant, Barré de. 1871. “AJC (1871b).” Théorie Et Equations générales Du Mouvement Non Permanent Des Eaux, Avec Application Aux Crues Des Rivières Et à l’introduction Des Marées Dans Leur Lit (2ème Note).” *Comptes Rendus Des séances de l’Académie Des Sciences*, 237–40.
- Sun, Hui, and Odd M. Faltinsen. 2006. “Water Impact of Horizontal Circular Cylinders and Cylindrical Shells.” *Applied Ocean Research* 28 (5): 299–311. <https://doi.org/https://doi.org/10.1016/j.apor.2007.02.002>.
- Sun, Pengnan, Furen Ming, and Aman Zhang. 2015. “Numerical Simulation of Interactions Between Free Surface and Rigid Body Using a Robust SPH Method.” *Ocean Engineering* 98: 32–49. <https://doi.org/https://doi.org/10.1016/j.oceaneng.2015.01.019>.
- Thomas Altazin, Frédéric Golay, Mehmet Ersoy, and Lyudmyla Yushchenko. 2016. “Numerical Investigation of BB-AMR Scheme Using Entropy Production as Refinement Criterion.” *International Journal of Computational Fluid Dynamics* 30 (3): 256–71. <https://doi.org/10.1080/10618562.2016.1194977>.
- Toro, E. F. 1997. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer. <https://books.google.fr/books?id=6QFAAQAAIAAJ>.
- Varra, Giada, Veronica Pepe, Renata Della Morte, and Luca Cozzolino. 2024. “A Novel Efficient and Robust Treatment of the Friction Source Term in 2D Shallow Water Inundation Models.” *Journal of Hydrology* 634: 131045. <https://doi.org/10.1016/j.jhydrol.2024.131045>.