TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Numerical investigation of the gas jet formation immediately after opening a champagne bottle

Author

Lukas Wagner, BSc

Student ID number: 01426990

Supervisor

Ao.Univ.Prof. Dipl.-Ing. Dr.techn.

Stefan Braun

Institut für Strömungsmechanik und Wärmeübertragung

Vienna, October 2021

Contents

1 Abstract

The goal of this thesis is to deliver a numerical simulation concerning the emissive behavior of pressurized CO_2 gas inside a Champagne bottle in the first few milliseconds after the cork popping. This system was proposed and experimentally performed by [1]. The mentioned goal, the embedding of existing literature and the overall approach regarding the proposed system are introduced in section 2. Chapter 3 describes the theoretical background needed to create the simulation, such as the Euler equations of a compressible fluid in cylindrical coordinates in their non-dimensional form. The next section explains the numerical discretization via finite volume methods to update the solution for each time step and introduces the CLAWPACK software package, which provides the framework of this thesis' Python and Fortran programs. The last part of chapter 4 determines the different settings of the simulation, for example explaining the Roe solver, and finally section 5 illustrates and explains the characteristics of the flow without any obstacle and secondly the effects of an incorporated cork on the fluid flow and vise versa.

2 Discharge Flow Behavior out of a Cylindrical Bottleneck

This thesis is based on the experimental work of [1] and provides the numerical simulation for the emission of gas in the first few microseconds after the popping of a champagne bottle. In the mentioned experiment six 75 cl champagne rosé bottles with 12.5% ethanol were used. 72 hours before the cork popping, four bottles were stored at 20 °C and two at 30 °C to examine the effects on bottles with different temperatures and therefore also varying pressures. During storage some dissolved CO_2 gas in liquid phase transitioned into gas phase in accordance with *Henry's law*, [2]. The CO_2 gas is instantly mixed with H_2O vapor, formed due to the exact same reason as for the CO_2. While the pressure of gas-phase CO_2 in the sealed bottle p_{CO_2} is 7.5 and 10.2 bar for the 20 and 30 °C bottle, the pressure of H_2O vapor p_{H_2O} is only 0.0203 and 0.0368 bar respectively. Because of the small net effect of water vapor on the overall emissive behavior, only the CO_2 gas will be considered in this thesis.

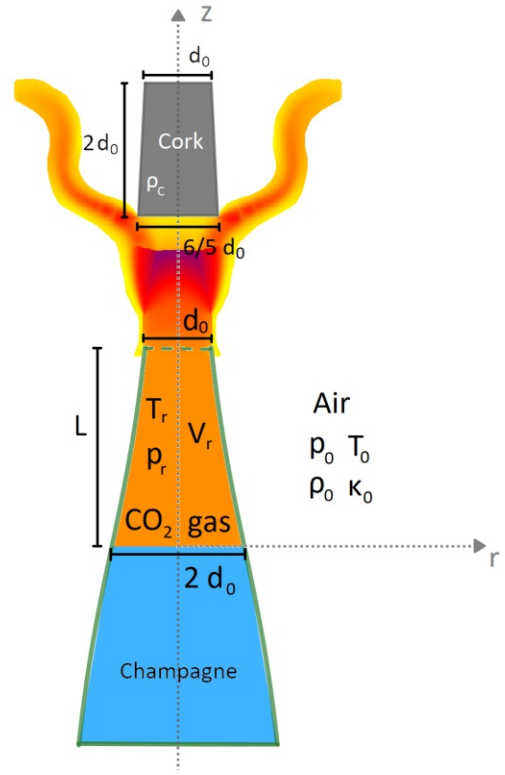

Figure 1: An illustration of the given system with its corresponding reference quantities.

After the bottle opening, the gas mixture undergoes an adiabatic expansion, leading to a drastic temperature decrease and velocity increase of the gas. While the temperature drop results in the formation of a blue haze for the 20 °C bottles due to *Rayleigh scattering* and a gray-white fog for the 30 °C bottles due to *Mie-scattering*, for this thesis the effects of the sudden velocity increase are considered more thoroughly. The emitting gas speeds up in a way that the velocity reaches Mach numbers $M > 1$, leading to the creation of shock waves in the form of *Mach disks*. This phenomenon and its temporal progression are described by the Euler equations of a compressible fluid, considering axial symmetry, which are solved numerically via *Godunov's finite volume method* with the implementation of the *Roe solver*.

3 Physical-Mathematical Modeling

As mentioned in the previous section, the CO_2 gas is the major contributor, regarding the emissive behavior of the fluid, due to its high gas pressure compared to the one for gaseous H_2O. This results in a two-phase mixture, initially being CO_2 inside and air outside of the bottle. To drastically simplify the given problem and therefore also the necessary equations, the CO_2 is exchanged with pressurized air, so that ultimately a single-phase system is proposed.

To mathematically describe the physical problem, a set of conservation laws, material, thermal and caloric equations must be given, so that the number of existing equations matches the number of unknown physical quantities, [3]. In this case the desired quantities are: the fluid's mass density ρ, three velocity components v^i due to the system's given dimensions, pressure p and energy density E ($N = 6$). The latter is defined as

$$E = \rho\Big(e + \frac{v^i v^i}{2}\Big)\,, \tag{1}$$

where ρe is the internal energy density.

While the conservation laws and material equations will be considered in section 3.2, the thermal and caloric equations of state describe the type of fluid and are introduced in this section. For the expected magnitude of the pressures and temperatures the assumption of an ideal gas is sufficient and therefore the ideal gas equation is used as the thermal equation of state,

$$p = \frac{\rho R T}{\mathcal{M}}\,, \tag{2}$$

where R is the universal gas constant, T the absolute temperature and $\mathcal{M}$ the molar mass of the fluid.

The following set of caloric equations of state holds true for an ideal gas and results in

$$e = \frac{f}{2}\,\frac{p}{\rho} = \frac{f}{2}\,\frac{RT}{M}\,, \qquad h = e + \frac{p}{\rho} = \Big(\frac{f}{2} + 1\Big)\,\frac{RT}{M}\,, \tag{3}$$

where ρh is the internal enthalpy density and f are the degrees of freedom ($f = 5$ for air), [22].

Another major quantity, which will be the only source of information concerning the type of fluid in the final equations, is the *heat-capacity ratio* κ

$$\kappa = \frac{c_p}{c_\mathcal{V}} = \frac{\left(\frac{\partial h}{\partial T}\right)_p}{\left(\frac{\partial e}{\partial T}\right)_\mathcal{V}}\,, \tag{4}$$

where $\mathcal{V} = 1/\rho$ is the specific volume, c_p and $c_\mathcal{V}$ are the heat capacities at constant pressure and volume, respectively.

Inserting (3) into (4), a relation between the heat-capacity ratio and the degrees of freedom can be found.

$$\kappa = \frac{f+2}{f} \tag{5}$$

With the definition of κ, e and h can be written as

$$e = \frac{1}{\kappa - 1}\ \frac{p}{\rho} + const\,, \qquad h = \frac{\kappa}{\kappa - 1}\ \frac{p}{\rho} + const\,, \tag{6}$$

where *const* is set to be 0.

Therefore E can be calculated to

$$E = \frac{p}{\kappa - 1} + \rho\,\frac{v^i v^i}{2}\,. \tag{7}$$

The relation between E and all other quantities in (7) reduces the number of unknown quantities by one ($N = 5$), so that p is chosen to be a dependent quantity.

The speed of sound c is given by

$$c = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_s} = \sqrt{\kappa\,\frac{p}{\rho}}\,, \tag{8}$$

where s is the entropy.

The following equation describes the total enthalpy density ρH.

$$H = h + \frac{v^i v^i}{2} \tag{9}$$

Defining the second half of (6) as a function of E and inserting it into (9), yields

$$H = \kappa\,\frac{E}{\rho} - (\kappa - 1)\,\frac{v^i v^i}{2}\,. \tag{10}$$

Note, that the proposed system is time-dependent, therefore the physical quantities depend on three spatial dimensions x_i, which define the position vector r^m, and the time t. However, for better readability, these parameters will never be explicitly mentioned in equations, unless the dependency is of any importance. All vectors are denoted by a single and matrices by two superscripts. If two indices are the same, Einstein's summation convention is used.

3.1 Dimensional Analysis

The whole set of equations and unknown quantities, as well as the parameters are dimension-afflicted, which means that for each different system the set of differential equations, introduced in section 3.2, has to be calculated individually, even if its initial state only varies in one constant. The latter would be the case, when designing a model for a physical object with the same proportions, but a usually smaller size. This issue can be solved by dimensional analysis. Hereby all physical quantities $\tilde{b}$ are split into a non-dimensional part b and a reference-quantity b_0, while the second one always stays constant ($\tilde{b} \rightarrow b\, b_0$).

For r^m and t a proper reference length d_0 and time $\tau_0 = d_0/c_0$ are chosen, which suit the given system, whereas for the unknown quantities, defined as the vector q^i, the reference quantities resemble the initial conditions of the environment (see appendix).

$$\tilde{q}^i(t) = \begin{pmatrix} \tilde{\rho} \\ \tilde{\rho}\,\tilde{v}^i \\ \tilde{E} \end{pmatrix}, \qquad \tilde{q}^i(t=0) \rightarrow \begin{pmatrix} 1\cdot\ \rho_0 \\ 0\cdot \rho_0\, c_0 \\ 1\cdot\ E_0 \end{pmatrix} = \begin{pmatrix} \rho_0 \\ 0 \\ E_0 \end{pmatrix} \tag{11}$$

The non-dimensional functions are equal for all systems with the same proportions and dependencies and are either 1 or 0 for the environment at $t = 0$, as seen in the second vector of (11). 0 in the second column shows that the initial conditions consider a static fluid, while 1 symbolizes that there must be a given mass and energy density present from the beginning.

The equations are then written in non-dimensional form, transforming E, c and H to

$$E = p + \kappa_0(\kappa_0 - 1)\,\rho\,\frac{v^i v^i}{2}, \tag{12}$$

$$c = \sqrt{\frac{p}{\rho}} = \sqrt{T}, \tag{13}$$

$$H = \frac{E}{\rho} - (\kappa_0 - 1)^2 \cdot \frac{v^i v^i}{2}, \tag{14}$$

where κ is already defined as a non-dimensional constant, therefore $\tilde{\kappa} \rightarrow \kappa_0$.

H can also be written in terms of velocities,

$$H = c^2 + (\kappa_0 - 1)\,\frac{v^i v^i}{2} \tag{15}$$

3.2 Governing Equations

The overall goal is to find a set of equations, which describes the spatial and temporal evolution of a viscous fluid in the presence of a stress tensor σ^{ij} and a heat flux Q^i. This set has to comply with physical laws conserving the quantities mass, momentum and energy. The approach is completely based on continuum mechanics, is independent of the fluid's molecular structure and finally leads to the *Navier-Stokes-equations*.

At first, σ^{ij} is defined as a continuous function with respect to the *strain-rate tensor* D^{ij}, considering spatial homogeneity. A Newtonian fluid is characterized by a linear dependency of σ^{ij} with respect to D^{ij} and results in its material equation

$$\sigma^{ij} = \left(-p + \bar{\mu}\, \nabla^k v^k \right) \delta^{ij} + 2\mu\, D^{ij}\,, \tag{16}$$

where $\bar{\mu}$ is the volume and μ the dynamic viscosity of the fluid, ∇^k the Nabla operator and δ^{ij} the *Kronecker-delta*.

The Navier-Stokes-equations are called the *continuity equation* (17), the *equation of motion* (18) and the *energy equation* (19), each considering the conservation of mass, momentum and energy, respectively, [3].

$$\mathrm{Sr}\,\frac{\partial \rho}{\partial t} + v^i\,\nabla^i \rho + \rho\,\nabla^i v^i = 0\,, \tag{17}$$

$$\mathrm{Sr}\,\rho\,\frac{\partial v^i}{\partial t} + \rho\, v^j \nabla^j v^i = -\mathrm{Eu}\,\nabla^i p + \frac{1}{\mathrm{Re}}\Big(\nabla^i\big(\bar{\mu}\,\nabla^k v^k\big) + \nabla^j\big(2\mu\, D^{ij}\big)\Big) + \frac{1}{\mathrm{Fr}^2}\,\rho e_g^i\,, \tag{18}$$

$$\rho\left(\mathrm{Sr}\,\frac{\partial h}{\partial t} + v^i \nabla^i h\right) - \mathrm{Eu}\;\mathrm{Ec}\left(\mathrm{Sr}\,\frac{\partial p}{\partial t} + v^i \nabla^i p\right) = \frac{\mathrm{Ec}}{\mathrm{Re}}\,\Phi - \frac{1}{\mathrm{Re}\;\mathrm{Pr}}\,\nabla^i Q^i\,, \tag{19}$$

where e_g^i is the unit vector pointing in the direction of an external force, Φ is the dissipation function and Sr, Eu, Fr, Re, Ec, Pr are the system's non-dimensional groups:

1. Strouhal number
2. Euler number
3. Froude number
4. Reynolds number
5. Eckert number
6. Prandtl number

Note, that (17)-(19) are nonlinear and that the Navier-Stokes-equations are written in non-dimensional form. The magnitudes of the corresponding non-dimensional groups allow the simplification of the given conservation laws, so that ultimately the Euler equations of a compressible fluid can be deduced.

Order of Magnitude Estimates:

The non-dimensional groups contained in the Navier-Stokes-equations are fractions of reference quantities which depend on the given system. These numbers are named after their inventors:

- Strouhal number ... $\mathrm{Sr} = \dfrac{d_0}{c_0\,\tau_0} = 1$
- Euler number ... $\mathrm{Eu} = \dfrac{p_0}{\rho_0\,c_0^2} = \dfrac{1}{\kappa_0}$
- Froude number ... $\mathrm{Fr} = \dfrac{c_0}{\sqrt{g_0\,d_0}} \approx 817 \quad \rightarrow \quad \dfrac{1}{\mathrm{Fr}^2} \approx 10^{-6}$
- Reynolds number ... $\mathrm{Re} = \dfrac{c_0\,d_0\,\rho_0}{\mu_0} \approx 4.07 \cdot 10^5 \quad \rightarrow \quad \dfrac{1}{\mathrm{Re}} \approx 10^{-6}$
- Eckert number ... $\mathrm{Ec} = \dfrac{c_0^2}{c_{p_0}\,T_0} = \kappa_0 - 1$
- Prandtl number ... $\mathrm{Pr} = \dfrac{\mu_0\,c_{p_0}}{\lambda_0} \approx 0.700$

Since $1/\mathrm{Fr}^2$ and $1/\mathrm{Re}$ are very small compared to the other values, gravitational and viscous effects are neglected in the present study. Due to these simplifications, the Navier-Stokes-equations reduce to the Euler equations for an inviscid, adiabatic, compressible gas flow:

$$\mathrm{Sr}\,\frac{\partial \rho}{\partial t} + v^i\,\nabla^i\rho + \rho\,\nabla^i v^i = 0\,, \tag{20}$$

$$\mathrm{Sr}\,\rho\,\frac{\partial v^i}{\partial t} + \rho\,v^j\nabla^j v^i + \mathrm{Eu}\,\nabla^i p = 0\,, \tag{21}$$

$$\rho\left(\mathrm{Sr}\,\frac{\partial h}{\partial t} + v^i\nabla^i h\right) - \mathrm{Eu}\;\mathrm{Ec}\left(\mathrm{Sr}\,\frac{\partial p}{\partial t} + v^i\nabla^i p\right) = 0\,. \tag{22}$$

Because material equations are used to describe the dependency of any heat flux or viscous stress on the temperature or velocity gradient, they are not considered in this thesis, [3]. Another reason for their redundancy is the already achieved number of required equations ($N = 5$), with the consideration of (21) being 3 separate equations.

The Euler equations in the form of (20)-(22) still require two modifications to be solvable for a broad spectrum of problems. The first one is expressing the equations in divergence form. This expression allows the definition of a speed matrix, which finally results in an eigenvalue problem. The second requirement is the definition of a proper coordinate system, in the present case the cylindrical coordinates.

3.3 Hyperbolic Differential Equations

Hyperbolic partial differential equations arise wherever a physical process can be described by a wave motion. This phenomenon is the case in disciplines like Newtonian mechanics, thermodynamics, electrodynamics, optics and many more. The unknown functions describing the system are again indicated by q^i and can be calculated with an important class of homogeneous hyperbolic equations called *conservation laws*, [4].

The general divergence form of these laws denoted in Einstein notation is

$$\frac{\partial q^i(r^m,t)}{\partial t} + \nabla^j f^{ij}(q^k(r^m,t)) = 0\,, \tag{23}$$

where f^{ij} is the flux matrix.

Modifying the total derivative $\nabla^j f^{ij}$ by using the chain rule, yields

$$\frac{\partial q^i(r^m,t)}{\partial t} + \frac{\partial f^{ij}(q^k(r^m,t))}{\partial q^k}\ \nabla^j q^k = 0\,. \tag{24}$$

One of the main tasks of this thesis is to transform equations in the form of (24) into a *quasilinear divergence form* like

$$\frac{\partial q^i(r^m,t)}{\partial t} + \sum_{l=1}^{3} A_l^{ij}\ \frac{\partial q^j(r^m,t)}{\partial x_l} = \Psi^i(q^k(r^m,t),\, r^m,t) \tag{25}$$

for the special case of a compressible Euler fluid in cylindrical coordinates.

Ψ^i represents a source term and $A_l^{ij} = \partial f_l^i / \partial q^j$ is the Jacobean or speed-matrix of the flux term f_l^i.

The total flux is defined as the sum of all flux terms: $f^i = \sum_l f_l^i$

3.4 Euler Equations

The Euler equations describe the motion and behavior of inviscid fluids, in this thesis assuming no inner friction (viscosity $\eta = 0$) and an adiabatic expansion. The continuity equation (26), the equation of motion (27) and the energy equation (28) are derived from the dimensional form of (20)-(22) and transformed into the form of (23), resulting in

$$\frac{\partial \rho}{\partial t} + \nabla^i\big(\rho v^i\big) = 0\,, \tag{26}$$

$$\frac{\partial}{\partial t}(\rho v^i) + \nabla^j\big(\delta^{ij} p + \rho v^i v^j\big) = 0\,, \tag{27}$$

$$\frac{\partial E}{\partial t} + \nabla^i\Big(v^i(E+p)\Big) = 0\,. \tag{28}$$

The general Euler equations (26) to (28) are coordinates-independent. In order to transform these equations, the basis unity vectors and the corresponding Nabla operator must be known. The basis $\mathcal{B}$ in cylindrical coordinates is defined as

$$\mathcal{B} = \left\{ \hat{e}_r^i = \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \\ 0 \end{pmatrix}\,,\ \hat{e}_\phi^i = \begin{pmatrix} -\sin(\phi) \\ \cos(\phi) \\ 0 \end{pmatrix}\,,\ \hat{e}_z^i = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}. \tag{29}$$

Every vector inside this space can be written as a linear combination of these unit vectors. Therefore the position and velocity vectors are

$$r^m = r\hat{e}_r^m\,, \qquad v^i = u\hat{e}_r^i + v\hat{e}_\phi^i + w\hat{e}_z^i\ . \tag{30}$$

The general Nabla operator is defined as

$$\nabla^i = \frac{\partial}{\partial r^i} = \sum_l \frac{1}{h_l}\,\hat{e}_l^i\,\frac{\partial}{\partial x_l}\,, \tag{31}$$

where $h_l = \left|\frac{\partial r^m}{\partial x_l}\right|$.

For cylindrical coordinates (31) becomes

$$\nabla^i = \hat{e}_r^i\,\frac{\partial}{\partial r} + \frac{1}{r}\,\hat{e}_\phi^i\,\frac{\partial}{\partial \phi} + \hat{e}_z^i\,\frac{\partial}{\partial z}\,. \tag{32}$$

The main assumption for the given problem of the gas discharge through a bottle neck is axial symmetry. All quantities solely depend on r and z, but not on ϕ. Therefore the system can be depicted in only two dimensions and the derivative of any function with respect to $\frac{\partial}{\partial\phi}$, as well as the velocity component v in the ϕ-direction become zero. Only the unit vectors still depend on ϕ and therefore $\frac{\partial}{\partial\phi}\hat{e}_r^i = \hat{e}_\phi^i$. This fact will be important for the calculation of the source term Ψ^i outlined below.

The overall number of unknown quantities therefore decreases by one ($N = 4$), as well as the number of given equations (the second equation of (27), considering the ϕ-component, yields: $0 = 0$). All other conservation laws in cylindrical coordinates are calculated by inserting v^i from (30) and ∇^i from (32) into (26)-(28).

$$\frac{\partial\rho}{\partial t} + \left(\hat{e}_r^i\frac{\partial}{\partial r} + \frac{1}{r}\hat{e}_\phi^i\frac{\partial}{\partial\phi} + \hat{e}_z^i\frac{\partial}{\partial z}\right)\left(\rho\left(u\hat{e}_r^i + w\hat{e}_z^i\right)\right) = 0 \tag{33}$$

$$\begin{aligned}\frac{\partial}{\partial t}\left(\rho\left(u\hat{e}_r^i + w\hat{e}_z^i\right)\right) + \left(\hat{e}_r^i\frac{\partial}{\partial r} + \frac{1}{r}\hat{e}_\phi^i\frac{\partial}{\partial\phi} + \hat{e}_z^i\frac{\partial}{\partial z}\right)p + \\ \left(\hat{e}_r^j\frac{\partial}{\partial r} + \frac{1}{r}\hat{e}_\phi^j\frac{\partial}{\partial\phi} + \hat{e}_z^j\frac{\partial}{\partial z}\right)\left(\rho\left(u\hat{e}_r^i + w\hat{e}_z^i\right)\left(u\hat{e}_r^j + w\hat{e}_z^j\right)\right) = 0\end{aligned} \tag{34}$$

$$\frac{\partial E}{\partial t} + \left(\hat{e}_r^i\frac{\partial}{\partial r} + \frac{1}{r}\hat{e}_\phi^i\frac{\partial}{\partial\phi} + \hat{e}_z^i\frac{\partial}{\partial z}\right)\left(\left(u\hat{e}_r^i + w\hat{e}_z^i\right)\left(E + p\right)\right) = 0 \tag{35}$$

After expanding all products by multiplication and solving the derivatives, the actual form of the *continuity equation* (36), the *equation of motion* in the r-direction (37) and the z-direction (38), as well as the *energy equation* (39) can be seen.

$$\frac{\partial\rho}{\partial t} + \frac{\partial}{\partial r}(\rho u) + \frac{\partial}{\partial z}(\rho w) = -\frac{\rho u}{r} \tag{36}$$

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial r}(p + \rho u^2) + \frac{\partial}{\partial z}(\rho u w) = -\frac{\rho u^2}{r} \tag{37}$$

$$\frac{\partial}{\partial t}(\rho w) + \frac{\partial}{\partial z}(p + \rho w^2) + \frac{\partial}{\partial r}(\rho u w) = -\frac{\rho u w}{r} \tag{38}$$

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial r}\big(u(E + p)\big) + \frac{\partial}{\partial z}\big(w(E + p)\big) = -\frac{u(E + p)}{r} \tag{39}$$

To finally be able to transform (36)-(39) into the shape of (25), q^i, Ψ^i and $f^i = f_r^i + f_z^i$ must be identified.

$$q^i = \begin{pmatrix} \rho \\ \rho u \\ \rho w \\ E \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix}, \qquad \Psi^i = -\frac{1}{r}\begin{pmatrix} \rho u \\ \rho u^2 \\ \rho u w \\ u(E+p) \end{pmatrix} = -\frac{1}{r}\begin{pmatrix} q_2 \\ q_2^2/q_1 \\ q_2 q_3/q_1 \\ \frac{q_2}{q_1}\big(q_4 + p(q^i)\big) \end{pmatrix}$$

$$f_r^i = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho u w \\ u(E+p) \end{pmatrix} = \begin{pmatrix} q_2 \\ p(q^i) + q_2^2/q_1 \\ q_2 q_3/q_1 \\ \frac{q_2}{q_1}\big(q_4 + p(q^i)\big) \end{pmatrix}, \qquad f_z^i = \begin{pmatrix} \rho w \\ \rho u w \\ p + \rho w^2 \\ w(E+p) \end{pmatrix} = \begin{pmatrix} q_3 \\ q_2 q_3/q_1 \\ p(q^i) + q_3^2/q_1 \\ \frac{q_3}{q_1}\big(q_4 + p(q^i)\big) \end{pmatrix} \tag{40}$$

$p(q^i)$ can be calculated from (7) to give

$$p(q^i) = (\kappa - 1)E - \rho\frac{v^i v^i}{2} = (\kappa - 1)q_4 - \frac{q_2^2 + q_3^2}{2q_1}\,. \tag{41}$$

With the definition of f_r^i and f_z^i, the corresponding Jacobean matrices can be calculated to yield

$$A_r^{ij} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{\partial p}{\partial q_1} - \left(\frac{q_2}{q_1}\right)^2 & \frac{\partial p}{\partial q_2} + 2\,\frac{q_2}{q_1} & \frac{\partial p}{\partial q_3} & \frac{\partial p}{\partial q_4} \\ -\frac{q_2 q_3}{q_1^2} & \frac{q_3}{q_1} & \frac{q_2}{q_1} & 0 \\ \frac{q_2}{q_1}\left(\frac{\partial p}{\partial q_1} - \frac{p+q_4}{q_1}\right) & \frac{1}{q_1}\left(q_2\frac{\partial p}{\partial q_2} + p + q_4\right) & \frac{q_2}{q_1}\frac{\partial p}{\partial q_3} & \frac{q_2}{q_1}\left(\frac{\partial p}{\partial q_4} + 1\right) \end{pmatrix}, \tag{42}$$

$$A_z^{ij} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\frac{q_2 q_3}{q_1^2} & \frac{q_3}{q_1} & \frac{q_2}{q_1} & 0 \\ \frac{\partial p}{\partial q_1} - \left(\frac{q_3}{q_1}\right)^2 & \frac{\partial p}{\partial q_2} & \frac{\partial p}{\partial q_3} + 2\,\frac{q_3}{q_1} & \frac{\partial p}{\partial q_4} \\ \frac{q_3}{q_1}\left(\frac{\partial p}{\partial q_1} - \frac{p+q_4}{q_1}\right) & \frac{q_3}{q_1}\frac{\partial p}{\partial q_2} & \frac{1}{q_1}\left(q_3\frac{\partial p}{\partial q_3} + p + q_4\right) & \frac{q_3}{q_1}\left(\frac{\partial p}{\partial q_4} + 1\right) \end{pmatrix}. \tag{43}$$

A_r^{ij} and A_z^{ij} look rather similar. This is the reason why the eigenvalues are the same, when exchanging the velocity components u and w, which is shown in section 4.1 .

Performing dimensional analysis transforms the source and flux terms to

$$\Psi^i = -\frac{1}{r}\begin{pmatrix} \rho u \\ \rho u^2 \\ \rho u w \\ u\big(E + (\kappa_0 - 1)\, p\big) \end{pmatrix}, \qquad (44)$$

$$f_r^i = \begin{pmatrix} \rho u \\ p/\kappa_0 + \rho u^2 \\ \rho u w \\ u\big(E + (\kappa_0 - 1)\, p\big) \end{pmatrix}, \quad f_z^i = \begin{pmatrix} \rho w \\ \rho u w \\ p/\kappa_0 + \rho w^2 \\ w\big(E + (\kappa_0 - 1)\, p\big) \end{pmatrix}.$$

These relations finally lead to the non-dimensional form of the Jacobean matrices.

$$A_r^{ij} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ (\kappa_0 - 1)\,\frac{u^2+w^2}{2} - u^2 & (3 - \kappa_0)\, u & -(\kappa_0 - 1)\, w & \frac{1}{\kappa_0} \\ -\,u\, w & w & u & 0 \\ \kappa_0 u\big((\kappa_0 - 1)^2\,\frac{u^2+w^2}{2} - H\big) & \kappa_0\big(H - (\kappa_0 - 1)^2 u^2\big) & -\kappa_0(\kappa_0 - 1)^2 u\, w & \kappa_0 u \end{pmatrix} \qquad (45)$$

$$A_z^{ij} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -\,u\, w & w & u & 0 \\ (\kappa_0 - 1)\,\frac{u^2+w^2}{2} - w^2 & -(\kappa_0 - 1)\, u & (3 - \kappa_0)\, w & \frac{1}{\kappa_0} \\ \kappa_0 w\big((\kappa_0 - 1)^2\,\frac{u^2+w^2}{2} - H\big) & -\kappa_0(\kappa_0 - 1)^2 u\, w & \kappa_0\big(H - (\kappa_0 - 1)^2 w^2\big) & \kappa_0 w \end{pmatrix} \qquad (46)$$

Note, that (42) and (43) have to be modified by multiplying $1/\kappa_0$ in row 2 or 3 and $(\kappa_0 - 1)$ in row 4 with every p to result in (45) and (46). The reason for that is because f_r^i and f_z^i have been changed by dimensional analysis too.

3.5 Boundary and Initial Conditions

Since viscous effects are neglected in the present study, the no-slip condition cannot be satisfied at solid walls. Instead, the no-penetration condition at solid boundaries is applied, [3].

$$(v^i - v_w^i)\,\hat{n}^i = 0\,, \qquad (v^i - v_w^i)\,\hat{t}^i \neq 0\,, \qquad \hat{n}^i\,\hat{t}^i = 0 \tag{47}$$

v_w^i is the velocity of the wall, $\hat{n}^i$ and $\hat{t}^i$ are the normal and tangent unit vectors.

The last requirement for solving any set of differential equations, describing unsteady processes, are initial conditions. These define the initial state of the system or rather the value of q^i at $t = 0$, as briefly mentioned in section 3.1.

4 Numerical Treatment

4.1 Riemann Problem

The Riemann problem is defined as the combination of the given equations and very special initial conditions, consisting of a piecewise constant function with at least one jump discontinuity, [4]. Since (25) has a quasi-linear form, the general solution must be a linear superposition of plane waves moving in the direction of the corresponding unit vector $\hat{n}^m$, which can be any linear combination of the basis vectors: $\hat{n}^m = n_r \hat{e}_r^m + n_z \hat{e}_z^m$. In addition, each wave propagates at constant speed s.

Therefore the solution looks like

$$q^i(r^m, t) = \bar{q}^i(\hat{n}^m\, r^m - st)\,. \tag{48}$$

Applying the first derivative with respect to time and the spatial dimensions yields

$$\begin{aligned} \frac{\partial q^i(r^m,t)}{\partial t} &= -s\,\bar{q}'^i(\hat{n}^m\, r^m - st)\,, \\ \frac{\partial q^j(r^m,t)}{\partial x_l} &= \hat{n}^m \frac{\partial r^m}{\partial x_l}\ \bar{q}'^j(\hat{n}^m\, r^m - st)\,. \end{aligned} \tag{49}$$

For the r- and z-directions, $\hat{n}^m \frac{\partial r^m}{\partial x_l}$ equals n_r and n_z, respectively.

Therefore the homogeneous part of (25) simplifies to

$$\left(n_r A_r^{ij} + n_z A_z^{ij}\right) \bar{q}'^j(\hat{n}^m\, r^m - st) = s\ \bar{q}'^i(\hat{n}^m\, r^m - st)\,. \tag{50}$$

(50) requires that both A_r^{ij} and A_z^{ij} are diagonalizable with real eigenvalues $\lambda_{r\,p}$, $\lambda_{z\,p}$ and eigenvectors $r^i_{r\,p}$, $r^i_{z\,p}$ (=*hyperbolicity condition*), so that ultimately q^i can be obtained from them.

In this thesis the non-dimensional eigenvalues and eigenvectors are calculated as follows.

$$\lambda_{r\,1} = u - c \;:\quad r^i_{r\,1} = \begin{pmatrix} 1 \\ u - c \\ w \\ \kappa_0\big(H - (\kappa_0 - 1)uc\big) \end{pmatrix}$$

$$\lambda_{r\,2} = u \;:\quad r^i_{r\,2} = \begin{pmatrix} 1 \\ u \\ w \\ \kappa_0\big(H - c^2\big) \end{pmatrix}, \quad \lambda_{r\,3} = u \;:\quad r^i_{r\,3} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \kappa_0(\kappa_0 - 1)w \end{pmatrix} \tag{51}$$

$$\lambda_{r\,4} = u + c \;:\quad r^i_{r\,4} = \begin{pmatrix} 1 \\ u + c \\ w \\ \kappa_0\big(H + (\kappa_0 - 1)uc\big) \end{pmatrix}$$

$$\lambda_{z\,1} = w - c \;:\quad r^i_{z\,1} = \begin{pmatrix} 1 \\ u \\ w - c \\ \kappa_0\big(H - (\kappa_0 - 1)wc\big) \end{pmatrix}$$

$$\lambda_{z\,2} = w \;:\quad r^i_{z\,2} = \begin{pmatrix} 1 \\ u \\ w \\ \kappa_0\big(H - c^2\big) \end{pmatrix}, \quad \lambda_{z\,3} = w \;:\quad r^i_{z\,3} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \kappa_0(\kappa_0 - 1)u \end{pmatrix} \tag{52}$$

$$\lambda_{z\,4} = w + c \;:\quad r^i_{z\,4} = \begin{pmatrix} 1 \\ u \\ w + c \\ \kappa_0\big(H + (\kappa_0 - 1)wc\big) \end{pmatrix}$$

The general form of (50) allows the definition of q^i to be some linear combination of the eigenvectors and therefore a superposition of waves propagating at different velocities, denoted by the eigenvalues, [4]:

$$q^i(r^m, t) = \sum_{p=1}^{4} \Big(w_{r\,p}(\hat{n}^m\, r^m - \lambda_{r\,p}\, t)\; r^i_{r\,p} + w_{z\,p}(\hat{n}^m\, r^m - \lambda_{z\,p}\, t)\; r^i_{z\,p} \Big), \tag{53}$$

where the functions $w_p(r^m, t) = w_p(\hat{n}^m\, r^m - \lambda_p\, t)$ are called the *characteristic variables*.

(53) can further be written as a sum of matrix-multiplications of $w^j = (w_1, \dots, w_4)$ with the transformation matrices $T^{ij} = (r_1^i, \dots, r_4^i)$,

$$q^i(r^m, t) = T_r^{ij}\, w_r^j(r^m, t) + T_z^{ij}\, w_z^j(r^m, t)\,. \tag{54}$$

The Jacobean matrices can be transformed into their diagonal matrices Λ^{kl} as well,

$$A^{ij} = T^{ik}\Lambda^{kl}T^{lj^{-1}}\,, \qquad \Lambda^{kl} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_4 \end{pmatrix}\,. \tag{55}$$

(55) is needed to validate the proposed form of q^i, so that an eigenvalue problem must arise again when inserting (54) and (55) into the homogeneous part of (25),

$$\begin{aligned} T_r^{ij}\,\frac{\partial w_r^j}{\partial t} + T_r^{ik}\Lambda_r^{kl}T_r^{lj^{-1}}\;\; T_r^{jm}\,\frac{\partial w_r^m}{\partial r} &= 0\,,\\ T_z^{ij}\,\frac{\partial w_z^j}{\partial t} + T_z^{ik}\Lambda_z^{kl}T_z^{lj^{-1}}\;\; T_z^{jm}\,\frac{\partial w_z^m}{\partial z} &= 0\,. \end{aligned} \tag{56}$$

Simplifying (56) yields

$$\begin{aligned} \frac{\partial w_r^i}{\partial t} + \Lambda_r^{ij}\,\frac{\partial w_r^j}{\partial r} &= 0\,,\\ \frac{\partial w_z^i}{\partial t} + \Lambda_z^{ij}\,\frac{\partial w_z^j}{\partial z} &= 0\,, \end{aligned} \tag{57}$$

which further reveals two eigenvalue problems independent from each other. The advantage of (57) compared with (50) is that its problem can be written as a set of scalar equations, when considering the properties of diagonal matrices.

$$\begin{aligned} \frac{\partial w_{r\,p}}{\partial t} + \lambda_{r\,p}\,\frac{\partial w_{r\,p}}{\partial r} &= 0\\ \frac{\partial w_{z\,p}}{\partial t} + \lambda_{z\,p}\,\frac{\partial w_{z\,p}}{\partial z} &= 0 \end{aligned} \tag{58}$$

The solutions of (58) are the already known characteristic variables in the form of

$$w_p(r^m, t) = \mathring{w}_p(\hat{n}^m\, r^m - \lambda_p\, t) = l_p^j\ \mathring{q}^j(\hat{n}^m\, r^m - \lambda_p\, t)\,, \tag{59}$$

where l_p^j are the left eigenvectors or the row-entries of the inverse transformation matrices $T^{ij^{-1}}$. The circle over some letters denotes the initial value of that quantity $\mathring{q}^i = q^i(t = 0)$.

Finally, q^i can be written in terms of its initial value as

$$q^i(r^m, t) = \sum_{p=1}^{4} \left(l_{r\,p}^j\, \mathring{q}^j(\hat{n}^m\, r^m - \lambda_{r\,p}\, t)\, r_{r\,p}^i + l_{z\,p}^j\, \mathring{q}^j(\hat{n}^m\, r^m - \lambda_{z\,p}\, t)\, r_{z\,p}^i \right). \tag{60}$$

Domain of Dependence and Range of Influence

(60) shows, that $q^i(r^m, t)$ only depends on $\mathring{q}^i$ at, especially in this thesis, four particular points $\hat{n}^m\, r^m - \lambda_p\, t$ for a given $\hat{n}^m$. This set of points is called the *domain of dependence* of the point (r^m, t). The value of $\mathring{q}^i$ at other points has no influence on the value of q^i at (r^m, t). This is only the case for linear hyperbolic equations and is the result from the fact that information propagates at finite speed. This fact is later discussed in relation to the *CFL condition* in section 4.5.5.

Another way to represent the domain of dependence is to focus on a single point r_0^m at time $t = 0$ and consider the influence of $\mathring{q}^i(r_0^m)$ on the solution $q^i(r^m, t)$. The initial condition will again only affect the solution along the characteristic lines $\hat{n}^m\, r_0^m + \lambda_p\, t$. This set is called the *range of influence* of the point r_0^m, [4].

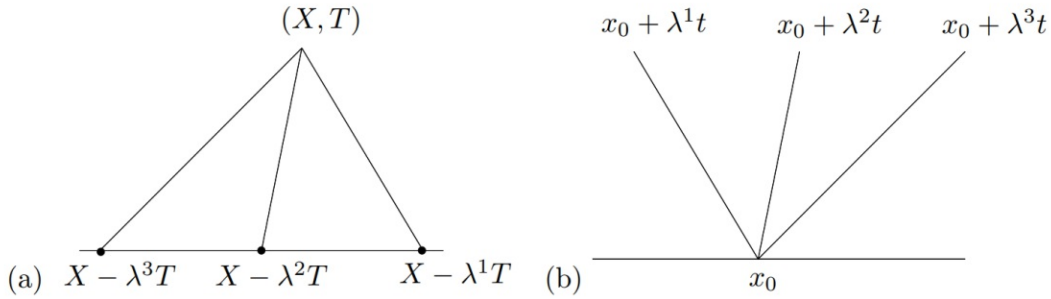

Figure 2: For a typical hyperbolic system of three equations with $\lambda_1 < 0 < \lambda_2 < \lambda_3$,
(a) domain of dependence of the point (X, T),
(b) range of influence of the point x_0, [4].

Discontinuous Solutions

(60) can even be used, when the initial data $\mathring{q}^i$ is neither smooth nor continuous at some points. If $\mathring{q}^i$ has a singularity at r_0^m, at least one characteristic variable $w_p(r_0^m, 0)$ will have one at that point as well. These singularities can then lead to ones in the solution q^i at some or even all points $\hat{n}^m\, r_0^m + \lambda_p\, t$. Conversely, if $\mathring{q}^i$ is smooth in a small region around the line $\hat{n}^m\, r^m - \lambda_p\, t$, then the solution q^i must also be smooth in the neighborhood of the point (r^m, t), [4].

When considering special initial data that is constant everywhere, except for a single jump discontinuity, $\mathring{q}^i$ can be split into a continuous left and right side. This split can be performed separately for each dimension and in the following will therefore only be mentioned once,

$$\mathring{q}^i(r^m) = \begin{cases} \hat{n}^m\, r^m < 0: & q_l^i \\ \hat{n}^m\, r^m > 0: & q_r^i \end{cases}. \tag{61}$$

q_l^i and q_r^i can be defined in terms of a linear combination of the corresponding characteristic variables as well,

$$q_l^i = \sum_{p=1}^{4} w_{l\,p}\; r_p^i\,, \qquad q_r^i = \sum_{p=1}^{4} w_{r\,p}\; r_p^i\,. \tag{62}$$

The discontinuity propagates with speed λ_p and therefore w_p is split into

$$w_p(r^m, t) = \begin{cases} \hat{n}^m\, r^m - \lambda_p\, t\; < 0: & w_{l\,p} \\ \hat{n}^m\, r^m - \lambda_p\, t\; > 0: & w_{r\,p} \end{cases}. \tag{63}$$

Finally q^i consists of two solutions, one left and the other one right from the singularity,

$$q^i(r^m, t) = \sum_{p:\, \hat{n}^m r^m < \lambda_p t}^{4} w_{l\,p}\; r_p^i \;+ \sum_{p:\, \hat{n}^m r^m > \lambda_p t}^{4} w_{r\,p}\; r_p^i\,. \tag{64}$$

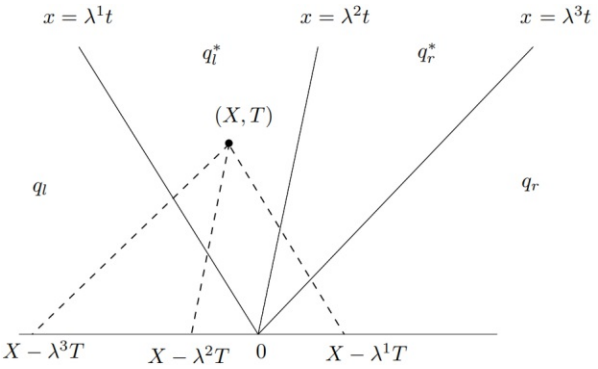

Figure 3: Construction of the solution to the Riemann problem at (X, T) for a system of 3 equations, [4].

Note, that the solution is constant between two lines of $\hat{n}^m r^m - \lambda_p t = 0$, as seen in figure 3. The p-th jump from $w_{l\,p}$ to $w_{r\,p}$ is defined as

$$\left(w_{r\,p} - w_{l\,p}\right)\, r_p^i = \alpha_p\, r_p^i \quad . \tag{65}$$

Being a scalar multiple of r_p^i with the constant α_p, the jump is an eigenvector of the Jacobean matrix A^{ij} (=*Rankine–Hugoniot jump condition*). Because of (65), the total jump from q_l^i to q_r^i can be written as

$$q_r^i - q_l^i = \sum_{p=1}^{4} \alpha_p\, r_p^i = \sum_{p=1}^{4} W_p^i\,, \tag{66}$$

where W_p^i is the p-th wave in the solution of the Riemann problem.

The solution q^i can then be written in terms of the waves in two different forms

$$\begin{aligned} q^i(r^m, t) &= q_l^i \;+ \sum_{p:\,\hat{n}^m r^m > \lambda_p t}^{4} W_p^i \quad , \\ q^i(r^m, t) &= q_r^i \;- \sum_{p:\,\hat{n}^m r^m < \lambda_p t}^{4} W_p^i \quad . \end{aligned} \tag{67}$$

The first equation of (67) can also be depicted with the use of the *Heaviside step function* H:

$$q^i(r^m, t) = q_l^i + \sum_{p=1}^{4} H(\hat{n}^m r^m - \lambda_p t)\; W_p^i\,. \tag{68}$$

The Phase Plane

The splitting of $q_r^i - q_l^i$ can be viewed in *state space*, where each vector q^i is represented by a point in a, for this case, four-dimensional space. For clearer explanation, only a system of two equations will be considered in this section. Therefore the state space can be described by the so-called *phase plane*. Since $q_r^i - q_l^i$ is an eigenvector of A^{ij}, the line from q_l^i to q_r^i must be parallel to the eigenvector r_1^i or r_2^i (see figure 4(a)). These lines give the set of all possible states (=*Hugoniot loci*), that can be connected to q_l^i by the waves W_1^i or W_2^i. This result is the same for q_r^i.

For a general Riemann problem with two equations and arbitrary q_l^i and q_r^i, the solution consists of two discontinuities traveling with speeds λ_1 and λ_2 and a new constant state q_m^i in between them (see figure 4(b)), [4],

$$q_m^i = w_{r\,1}\; r_1^i + w_{l\,2}\; r_2^i\,, \tag{69}$$

so that $q_m^i - q_l^i = (w_{r\,1} - w_{l\,1})\; r_1^i$ and $q_r^i - q_m^i = (w_{r\,2} - w_{l\,2})\; r_2^i$.

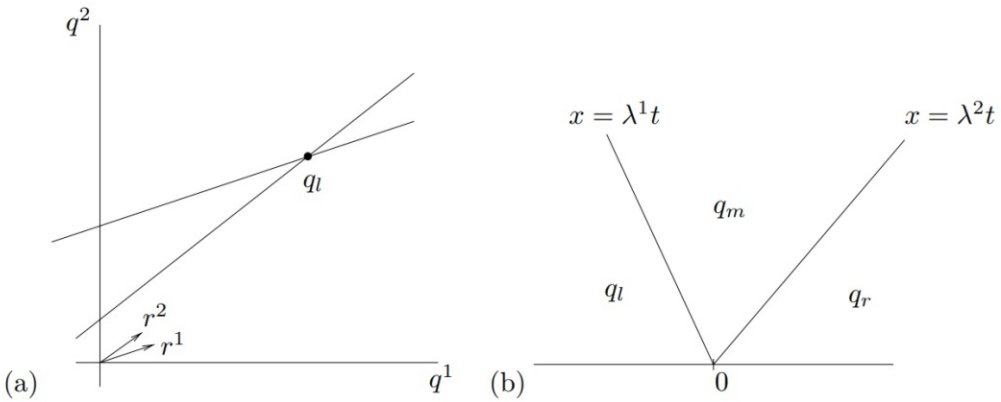

Figure 4: (a) The Hugoniot loci of the state q_l^i.
(b) Solution to the Riemann problem in the x–t plane, [4].

Considering a system with four dimensions again, (66) can be used to obtain a piecewise linear path from q_l^i to q_r^i in four-dimensional state space.

4.2 Finite Volume Methods

A finite volume method subdivides the system's spatial domain into small, but finite volumes, the so-called grid cells, and considers an approximation to the integral of q^i over each of these volumes. This kind of method also deals with discrete time steps by updating all values, using approximations to the flux through the boundaries of the given domain.

The average value of q^i over the m-th cell at time t_n is defined as

$$Q^i(r_m^i, t_n) \approx \frac{1}{\Delta V(r_m^i)} \int\limits_{\Delta V(r_m^i)} q^i(r^i, t_n) \ \mathrm{d}V \,. \tag{70}$$

For better clarity, from (70) on every vector is going to be denoted by the index i, including the position vector r^i.

$\Delta V(r_m^i)$ is the volume of the m-th cell and only depends on the position of the cell center r_m^i, assuming a static grid. For a constant ΔV the cells are evenly spaced, which is the definition of a uniform grid. If q^i is a smooth function, the integral in (70) agrees with its value at the cell center to $\mathcal{O}(\Delta V(r_m^i)^2)$, [4].

The integral form of the conservation laws from (23) generally gives

$$\frac{\mathrm{d}}{\mathrm{d}t} \int\limits_{\Delta V_m} q^i(r^i, t) \ \mathrm{d}V = f_{in}^i\big(q^i(r_m^i, t)\big) - f_{out}^i\big(q^i(r_m^i, t)\big) \,, \tag{71}$$

where f_{in}^i is the total flux into and f_{out}^i out of the m-th cell, and $\Delta V_m = \Delta V(r_m^i)$.

The integral form can then be used to develop an explicit time-marching algorithm. Integrating (71) over a single time step from t_n to t_{n+1} yields

$$\int\limits_{\Delta V_m} \Big(q^i(r^i, t_{n+1}) - q^i(r^i, t_n)\Big) \,\mathrm{d}V = \int\limits_{t_n}^{t_{n+1}} \Big(f_{in}^i\big(q^i(r_m^i, t)\big) - f_{out}^i\big(q^i(r_m^i, t)\big)\Big) \,\mathrm{d}t \,. \tag{72}$$

Rearranging (72), dividing by ΔV_m and using the definition of (70) gives

$$Q^i(r_m^i, t_{n+1}) = Q^i(r_m^i, t_n) - \frac{\Delta t_n}{\Delta V_m}\big(F_{out}^i(r_m^i, t_n) - F_{in}^i(r_m^i, t_n)\big) \,, \tag{73}$$

where $F^i(r_m^i, t_n)$ is defined as some approximation to the average flux over the time step $\Delta t_n = t_{n+1} - t_n$,

$$F^i(r^i_m, t_n) \approx \frac{1}{\Delta t_n} \int_{t_n}^{t_{n+1}} f^i\big(q^i(r^i_m, t)\big)\, \mathrm{d}t\,. \tag{74}$$

The average flux can be approximated based on the values of the surrounding cells. This leads to a fully discrete method.

In one dimension the incoming average flux can be described only by the value Q^i of the given cell and its left neighbor and the outgoing flux by the value of the given cell and its right neighbor,

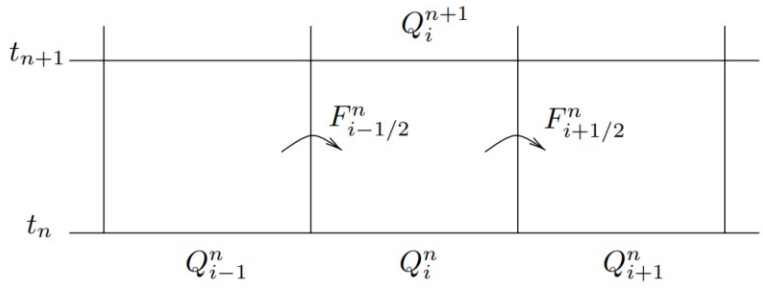

Figure 5: Illustration of a finite volume method for updating the cell average $Q^n_i = Q(x_i, t_n)$ by fluxes at the cell edges $F^n_{i-1/2} = F_{in}(x_i, t_n)$, $F^n_{i+1/2} = F_{out}(x_i, t_n)$, shown in x–t space, [4].

$$\begin{aligned} F^i_{in}(x_m, t_n) &= \mathcal{F}^i\big(Q^i(x_{m-1}, t_n), Q^i(x_m, t_n)\big)\,, \\ F^i_{out}(x_m, t_n) &= \mathcal{F}^i\big(Q^i(x_m, t_n), Q^i(x_{m+1}, t_n)\big)\,, \end{aligned} \tag{75}$$

where $\mathcal{F}^i$ is some numerical flux vector.

In this one-dimensional case the updated vector $Q^i(x_m, t_{n+1})$ only depends on three values: $Q^i(x_{m-1}, t_n)$, $Q^i(x_m, t_n)$ and $Q^i(x_{m+1}, t_n)$.

The sum of all flux differences cancels out, except for the fluxes at the domain's edges. Therefore all values of Q^i are conserved, except in the outermost cells of the domain, which must be solved by boundary conditions, [4].

Godunov's Method

Godunov's method revolutionized the field of computational fluid dynamics by providing a solution for systems of equations, where information propagates in different directions and where discontinuities, like shock waves, can be present.

The method consists of the following three steps:

- Definition of a piecewise polynomial function $\tilde{q}^i(r^i, t_n)$ for all r^i at time t_n,
- Definition of the in- and outgoing fluxes,
- Application of the flux-differencing formula from (73).

The simplest form of $\tilde{q}^i$ is denoted by constants inside the cells and can therefore solely be represented by the average value of q^i,

$$\tilde{q}^i(r^i, t_n) = Q^i(r^i_m, t_n), \quad r^i \in \Delta V_m \,. \tag{76}$$

For the case of hyperbolic systems that are not in conservation form, $\tilde{q}^i$ will typically have at least one discontinuity inside every cell. For example, a linear system of three one-dimensional equations with eigenvalues $\lambda_1 < 0 < \lambda_2 < \lambda_3$ has three discontinuities in the m-th grid cell at the points $x_{m-1/2} + \lambda_2 \, \Delta t_n$, $x_{m-1/2} + \lambda_3 \, \Delta t_n$ and $x_{m+1/2} + \lambda_1 \, \Delta t_n$.

Instead of trying to work with $\tilde{q}^i$ directly, the new cell averages are calculated by considering the waves of the Riemann problem and their corresponding speeds. The first ones can be expressed by transforming (66) into its numerical, one-dimensional form

$$Q^i(x_m, t_n) - Q^i(x_{m-1}, t_n) = \sum_p \mathcal{W}^i_p(x_{m-1/2}, t_n) \,, \tag{77}$$

where $\mathcal{W}^i_p$ is the numerical p-th wave at the left boundary of the m-th cell.

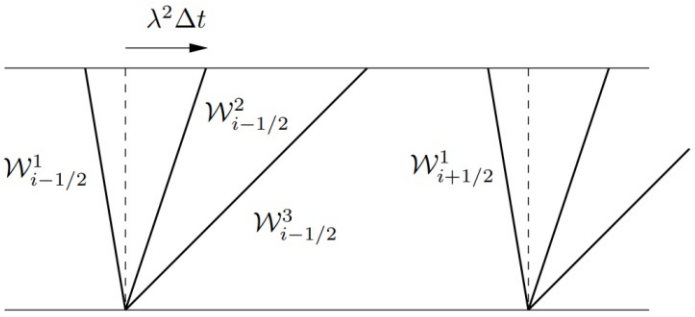

Figure 6: The wave movement of a system with three one-dimensional equations. $\mathcal{W}^p_{i-1/2} = \mathcal{W}_p(x_{i-1/2}, t_n)$, $\lambda^2 = \lambda_2$, [4].

The Riemann problem depicted in figure 6 is solved at each cell interface and the wave structure is used to determine $Q^i(x, t_{n+1})$. After the passing of the time step Δt_n, the wave $\mathcal{W}_2^i$, for example, has moved the distance $\lambda_2 \, \Delta t_n$. This movement results in the decrease of the average value of q^i by the amount

$$- \frac{\lambda_2 \, \Delta t_n}{\Delta x_m} \; \mathcal{W}_2^i(x_{m-1/2}, t_n) \, , \tag{78}$$

where $\Delta x_m = x_{m+1/2} - x_{m-1/2}$ is the length of the m-th cell. The minus sign arises because $\mathcal{W}_2^i$ measures the jump from right to left.

The new cell average can be found by simply adding up these independent effects,

$$Q^i(x_m, t_{n+1}) = Q^i(x_m, t_n) - \frac{\Delta t_n}{\Delta x_m} \sum_p \Big(\lambda_{+\,p} \, \mathcal{W}_p^i(x_{m-1/2}, t_n) + \lambda_{-\,p} \, \mathcal{W}_p^i(x_{m+1/2}, t_n) \Big) \, , \tag{79}$$

where $\lambda_{+\,p} = \max(\lambda_p, 0)$ and $\lambda_{-\,p} = \min(\lambda_p, 0)$.

For a better approach regarding the physical properties of the waves, the net effects of all right-traveling waves from the left edge of the m-th cell are defined as $A_+^{ij} \, \Delta Q^j(x_{m-1/2}, t_n)$ and the left-traveling waves from the right edge as $A_-^{ij} \, \Delta Q^j(x_{m+1/2}, t_n)$. These net effects are called *wave fluctuations*.

Using the diagonal form of the Jacobean matrix from (55) and defining Λ_+^{kl} as the diagonal matrix of all $\lambda_{+\,p}$ and Λ_-^{kl} of all $\lambda_{-\,p}$, A_+^{ij} and A_-^{ij} are stated as

$$\begin{gathered} A^{ij} = T^{ik} \Lambda^{kl} \, T^{lj^{-1}} = T^{ik} \big(\Lambda_+^{kl} + \Lambda_-^{kl} \big) \, T^{lj^{-1}} = A_+^{ij} + A_-^{ij} \, , \\ A_+^{ij} = T^{ik} \Lambda_+^{kl} \, T^{lj^{-1}} \, , \quad A_-^{ij} = T^{ik} \Lambda_-^{kl} \, T^{lj^{-1}} \, . \end{gathered} \tag{80}$$

The splitting of A^{ij} is essential for describing the wave fluctuations in terms of the total amount of right- or left-traveling waves,

$$A^{ij} \, \Delta Q^j = T^{ik} \Lambda^{kl} \, T^{lj^{-1}} \; \sum_p \mathcal{W}_p^j = \sum_p r_p^i \, \lambda_p \, l_p^j \;\; \alpha_p \, r_p^j = \sum_p \lambda_p \, \mathcal{W}_p^i \, . \tag{81}$$

This leads to the final form of the fluctuations

$$\begin{aligned} A_+^{ij}\,\Delta Q^j(x_{m-1/2},t_n) &= \sum_p \lambda_{+\,p}\,\mathcal{W}_p^i(x_{m-1/2},t_n)\,, \\ A_-^{ij}\,\Delta Q^j(x_{m+1/2},t_n) &= \sum_p \lambda_{-\,p}\,\mathcal{W}_p^i(x_{m+1/2},t_n)\,. \end{aligned} \tag{82}$$

As a consequence (79) becomes

$$Q^i(x_m,t_{n+1}) = Q^i(x_m,t_n) - \frac{\Delta t_n}{\Delta x_m}\left(A_+^{ij}\,\Delta Q^j(x_{m-1/2},t_n) + A_-^{ij}\,\Delta Q^j(x_{m+1/2},t_n)\right). \tag{83}$$

To transform (83) into (73), the in- and out-going average fluxes for a general linear problem must be defined as

$$\begin{aligned} F_{in}^i(x_m,t_n) &= f^i\big(Q^i(x_m,t_n)\big) - A_+^{ij}\,\Delta Q^j(x_{m-1/2},t_n)\,, \\ F_{out}^i(x_m,t_n) &= f^i\big(Q^i(x_m,t_n)\big) + A_-^{ij}\,\Delta Q^j(x_{m+1/2},t_n)\,, \end{aligned} \tag{84}$$

where f^i is the flux term defined in section 3.3.

For a conservative system the in-going average flux must cancel out the out-going one at the cell interface $x_{m-1/2}$ inside the domain, hence

$$\begin{gathered} F_{out}^i(x_{m-1},t_n) - F_{in}^i(x_m,t_n) = 0\,, \\ A_-^{ij}\,\Delta Q^j(x_{m-1/2},t_n) + A_+^{ij}\,\Delta Q^j(x_{m-1/2},t_n) = f^i\big(Q^i(x_m,t_n)\big) - f^i\big(Q^i(x_{m-1},t_n)\big)\,. \end{gathered} \tag{85}$$

The difference between the fluxes is split into a left-going fluctuation updating $Q^i(x_{m-1})$ and a right-going one updating $Q^i(x_m)$. This method is called *flux-difference splitting* and is often used for the calculation of (83).

4.3 Installation and Setup of CLAWPACK

The CLAWPACK software package, [5], delivers a wide range of algorithms, which apply to a variety of hyperbolic systems, by providing the appropriate Riemann solver along with the basic framework for creating the user's Fortran or Python program. It also includes templates and examples, which were very helpful in designing this thesis' program. Further information can be obtained from [5].

In this thesis the installation of CLAWPACK is done on an Ubuntu-subsystem of a Windows PC because it was only possible to run the software package directly on a Linux kernel. Therefore the following section can be skipped when already working on any Linux distribution.

Installation of the Ubuntu-Subsystem

For Windows 10 Ubuntu can be installed from the Microsoft Store. Under default configuration the whole environment is saved inside the folder:

*C:\Users*user*\AppData\Local\ Packages\CanonicalGroupLimited.*
UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs .

After choosing a username and a password, the system must be updated with the command: *sudo apt update*.
Then, Python's package installer *pip*, the Fortran compiler *gfortran* and the mathematical package for Python, called *NumPy*, must all be downloaded via the following commands.

sudo apt install gfortran
sudo apt install python3-pip
sudo apt install python3-numpy

Note, that for this thesis CLAWPACK was compiled with *python3* as well as *gfortran*. For later versions of Python or other Fortran compilers, all three commands must be changed accordingly.

Installation of CLAWPACK

CLAWPACK's current version (in this case v5.8.0) can be downloaded as a tar-file from [6]. After copying the file to anywhere inside the environment folder, the command *sudo tar -xzf clawpack-v5.8.0.tar.gz* unpacks the file into a folder with the same name. Opening it with *cd clawpack-v5.8.0*, reveals the data structure of the software. Inside this folder the following command finally installs CLAWPACK, so that it can be used inside every Python script: *sudo python3 setup.py config_fc –fcompiler=gfortran install*

Installation of Pycharm

PyCharm is an integrated development environment (=IDE) and therefore simplifies the process of editing, compiling and error finding. While for a native Linux machine the setup of Pycharm is rather straightforward, additional configurations need to be applied for the Ubuntu-subsystem. Therefore this section is only necessary for the virtual environment too.

At first, the professional version of the IDE must be downloaded from [7], due to its ability to use the subsystem for compilation. After installation, a new project can be created. Under the tab *Pure Python*, Ubuntu can be chosen by selecting *Previously configured interpreter*, clicking on the three dots and choosing *WSL* (=Windows subsystem for Linux). Under the *Python interpreter path*, in this case, the *python3.8* file inside the already configured */usr/bin* folder must be selected. The default *path mapping*, where all projects are stored, is */mnt/c/Users/*user*/PycharmProjects*. This can later be changed under *File→Settings...→Project:*name*→Python Interpreter*.

To finally be able to run a Python program inside the Ubuntu subsystem, the *shell path* must be changed under *File→Settings...→Tools→Terminal* from *cmd.exe* to *wsl.exe*, [8].

Graphical Output

To visualize the resulting data, the Python package *Matplotlib* is used. Additionally, a display server for Windows, like *Xming*, is needed to be able to see the graphs created by Ubuntu. After downloading Xming from [9], Matplotlib can be installed via the following command: *sudo apt install python3-matplotlib*

Finally, the desired Python script can be compiled with the following command sequence:

*cd ∼/clawpack-v5.8.0/ ; sudo python3 setup.py config_fc –fcompiler=gfortran install ; cd /mnt/c/Users/*user*/PycharmProjects ; DISPLAY=:0 python3 *program*.py*

The second command only has to be called when modifying the Riemann solver and *DISPLAY=:0* is needed to address the display server. Note, that all expressions surrounded by stars must be adjusted accordingly, that the commands should be typed in manually because they cannot be pasted correctly in the Linux shell and that Matplotlib must not be installed via the pip package installer.

Custom Riemann Solver

CLAWPACK already comes with a variety of different Riemann solvers, including Euler-equation-solvers in two spatial dimensions which, especially the hlle-solver, will be the basis of this thesis' custom Riemann solver. To use this one, a few steps must be performed to let CLAWPACK recognize the solver.

- Place the Riemann solver inside the folder */clawpack-v5.8.0/riemann/src*
- Add the solver to the list in */clawpack-v5.8.0/riemann/riemann/setup.py*
- Add the solver to the list in */clawpack-v5.8.0/riemann/riemann/__init__.py* , [10].

While the basic framework of the problem is written in Python3, the Riemann solvers are usually Fortran90 files. This allows to combine the convenience of Python classes with the computational efficiency of the Fortran language. This is done via the CLAWPACK package *PyClaw*.

4.4 PyClaw's Framework

PyClaw's Framework consists of the following classes:

- Domain
- Solver
- State
- Solution
- Controller

4.4.1 Domain

PyClaw's geometry package contains the classes used to define the space, in which the solution lives. The base class for all other geometry-related quantities is called the *domain*. This object can contain multiple *patches*, which can have different coordinate mappings, or that are used to construct complex domain shapes. However, in this thesis only one patch is used and is therefore indistinguishable from the domain. Another simpler, but not so elegant way to construct a complex domain structure, is to define custom boundary conditions and simply disregard the space outside the constructed borders.

The patch can include multiple *grids*, and these in turn consist of *dimension* objects, which define the start- and end-point of the given dimension, as well as the number of grid cells in it, [11].

In the most complex case a domain could look as depicted in figure 7.

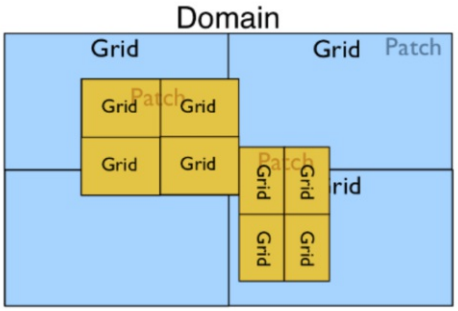

Figure 7: Complex domain structure, [11].

Again, in this thesis only one grid is used, therefore possessing the same dimensions as the domain. The grid is then automatically defined by the parameters of the base class.

A grid mapping can be used to describe a non-rectangular domain or to locally adjust the spacing of some grid cells. This method can be helpful when dealing with non-Cartesian coordinate systems. In spite of the thesis' cylindrical coordinates, the domain has axial symmetry and can therefore be described by a two-dimensional Cartesian system, where source terms describe the components of mixed spatial dimensions, as seen in section 3.4 . Therefore a grid mapping is not considered in this case.

4.4.2 Riemann Solver

The Riemann Solver uses the given set of hyperbolic equations and computes the waves and speeds at every cell interface, defined by the system's domain, to update the solution of the problem for a given time step. The majority of computational work is done by a so-called *normal solver*, that solves the one-dimensional problem in the direction perpendicular to a cell interface. The extension of this numerical method for higher dimensions is called *dimensional splitting*, [12].

Furthermore, the solver-class is responsible for the implementation of high resolution limiters, source terms and boundary conditions. The latter ones are realized via auxiliary fields and updating these fields before each time step is done by the solver as well.

The normal or one-dimensional solver consists of 11 input and 4 output arguments.

Input Parameters

The following input parameters must be allocated to the numerical Riemann solver:

- ixy ... determines the used dimension (here: either the r- or z-direction)
- mx ... the total cell number of the chosen dimension
- $maxm$ ... the maximum total cell number of all dimensions
- $meqn$ ... the number of equations used (here: 4)
- $mwaves$ ... the number of waves
- $maux$ ... the number of auxiliary fields used
- mbc ... the number of ghost cells at each boundary
- ql, $qr(meqn, 1 - mbc : maxm + mbc)$ ...
  the solutions of the problem on the left or right side of the cells (see figure 8)
- $auxl$, $auxr(maux, 1 - mbc : maxm + mbc)$ ...
  the auxiliary fields' values on the left or right side of the cells

Output Parameters

The following output parameters are returned by the numerical Riemann solver:

- $s(mwaves, 1 - mbc : maxm + mbc)$ ... the speeds of the waves s_p
- $wave(meqn, mwaves, 1 - mbc : maxm + mbc)$ ... the waves $\mathcal{W}_p^i$
- $amdq(meqn, 1 - mbc : maxm + mbc)$ ... the left-going fluctuations $A_-^{ij}\,\Delta Q^j$
- $apdq(meqn, 1 - mbc : maxm + mbc)$ ... the right-going fluctuations $A_+^{ij}\,\Delta Q^j$

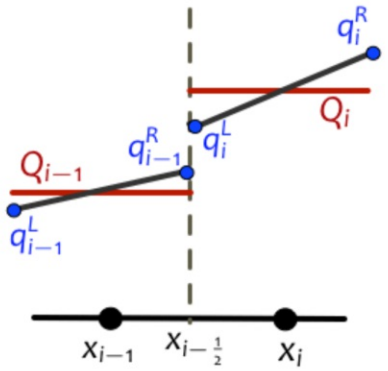

Figure 8: Various cell values at the cell centers x_{i-1}, x_i and interface $x_{i-\frac{1}{2}}$, [12].

While generally $ql(:,i) \neq qr(:,i)$ and therefore both values differ from the cell average $Q(:,i)$, in this thesis at least the auxiliary fields stay constant through the i-th cell: $auxl(:,i) = auxr(:,i) = aux(:,i)$. A single colon denotes that the numerical operation is performed over all given entries of the replaced index.

Dimensional Splitting

As the name already suggests, for this method a multidimensional problem is simply split into a sequence of one-dimensional problems. In this thesis the problem along the z-axis is solved first by applying the flux-differencing formula (73) only in one dimension. Therefore F^i_{in} and F^i_{out} are also only average fluxes in the z-direction,

$$\tilde{Q}^i(r^i_{m_z,m_r}, t_n) = Q^i(r^i_{m_z,m_r}, t_n) - \frac{\Delta t_n}{\Delta V_m}\left(\mathcal{F}^i(r^i_{m_z+1/2,m_r}, t_n) - \mathcal{F}^i(r^i_{m_z-1/2,m_r}, t_n)\right), \quad (86)$$

where $\mathcal{F}^i(r^i_{m_z-1/2,m_r}, t_n) = F^i_{in}(r^i_{m_z,m_r}, t_n)$ and $\mathcal{F}^i(r^i_{m_z+1/2,m_r}, t_n) = F^i_{out}(r^i_{m_z,m_r}, t_n)$.

The difference between (86) and (73) is that the cell's index m is split into a two-dimensional array (m_z, m_r) and the result is not denoted by the average value $Q^i(r^i_m, t_{n+1})$, but $\tilde{Q}^i(r^i_{m_z,m_r}, t_n)$. To get the updated value of Q^i, the flux-differencing formula must be used again, but this time with average fluxes in the r-direction: $\tilde{F}^i_{in}$, $\tilde{F}^i_{out}$,

$$Q^i(r^i_{m_z,m_r}, t_{n+1}) = \tilde{Q}^i(r^i_{m_z,m_r}, t_n) - \frac{\Delta t_n}{\Delta V_m}\left(\tilde{\mathcal{F}}^i(r^i_{m_z,m_r+1/2}, t_n) - \tilde{\mathcal{F}}^i(r^i_{m_z,m_r-1/2}, t_n)\right), \quad (87)$$

where $\tilde{\mathcal{F}}^i(r^i_{m_z,m_r-1/2}, t_n) = \tilde{F}^i_{in}(r^i_{m_z,m_r}, t_n)$ and $\tilde{\mathcal{F}}^i(r^i_{m_z,m_r+1/2}, t_n) = \tilde{F}^i_{out}(r^i_{m_z,m_r}, t_n)$.

4.4.3 State

The state object stores the data of the fields that exist on a given patch, or in this case on the whole domain. These fields include the average value Q^i and auxiliary fields, which are used to describe the given boundary conditions and some relevant parameters. The latter ones are also used inside the normal solver and in this thesis describe the velocity of a solid object. The arguments to the constructor must therefore include the domain, the number of equations and the number of auxiliary fields, [13].

The state also interacts with the solver by providing certain constants that occur inside the given equations. In this thesis the only constant will be the heat capacity ratio κ_0.

4.4.4 Solution

The solution object is a container for the data of the state and domain classes and defines the problem's solution over the whole time interval. For a system with multiple grids and different states, the solution could look like figure 9.

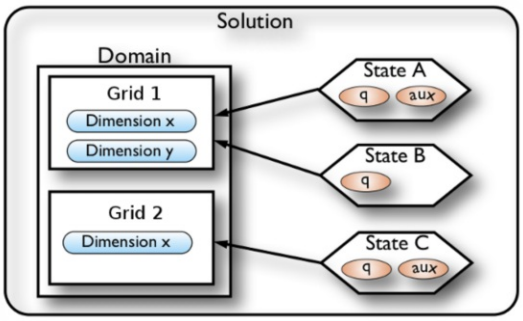

Figure 9: Solution of a complex grid-state-structure, [14].

4.4.5 Controller

The controller class provides a convenient way for running the simulation and configuring the specifications of the corresponding data output. The latter includes the format and the overall output style of the generated figures, which are created by the *Matplotlib* package, [15].

The only two input parameters are the solver and the solution object. Internal parameters are used to specify the number of figures *num_output_times* and the final time *tfinal*, where the simulation should end (assuming $t_{start} = 0$). If no output is desired *keep_copy* is set to *False*, otherwise the *setplot*-class determines the general structure and appearance of the plot.

The graphical output in *setplot* is done with another CLAWPACK package, called *VisClaw*. For every different plot-setup a *new_plotfigure* object must be initialized. In this thesis four *new_plotfigure* objects are created to display the mass density, pressure, Mach number and temperature of the fluid inside the given domain.

These objects include the following instances:

- *new_plotaxes* ... defines attributes like title, aspect ratio and axes' limits
- *figno* ... the number of the corresponding figure
- *afteraxis* ... a powerful class that can include or change aspects of the plot which cannot be accessed by any VisClaw instance directly.
- *new_plotitem* ... the type of figure (in this case a two-dimensional heat-map, hence *new_plotitem(plot_type=2d_pcolor)*)
- *plot_var* ... the data to plot
- *add_colorbar* ... if the heat map should include a color-bar
- *pcolor_cmap* ... the color gradient of the bar
- *pcolor_cmin*, *pcolor_cmax* ... the limits of the bar

Finally, the *run()* command runs the simulations and *plot()* creates the figures via *Xming* or the already integrated graphics software of Linux itself.

4.5 Configuration of the Numerical Simulation

While the general description of PyClaw's framework was already given in section 4.4, this one focuses on the configurations used in this thesis. The chosen settings include:

- Solver
- Geometry
- Source Term
- Boundary Conditions
- Spatial and Temporal Resolution
- Order of Precision

All other options, unless mentioned otherwise, are set by the default classes, functions or parameters of the CLAWPACK software and therefore will not be discussed any further.

4.5.1 Roe Solver

Linearized Riemann Solvers

A practical way to simplify the numerical simulation of nonlinear systems is the application of *Linearized Riemann Solvers*. Hereby the quasi-linear form of (25) is discretised by using the matrix $\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big)$, which is some approximation to the Jacobean matrix A^{ij} valid in a neighborhood of the data $Q^i(x_{m-1})$ and $Q^i(x_m)$. This slight modification defines the linearized problem locally at each cell interface, [4].

In order to properly define $\hat{A}^{ij}$, it must fulfill certain criteria similar to the ones for the original matrix:

- It is diagonalizable with real eigenvalues $\hat{\lambda}_p(x_{m-1/2})$ and vectors $\hat{r}^i_p(x_{m-1/2})$.
- It is hyperbolic
- It is consistent with the original conservation laws:
  $\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big) \rightarrow A^{ij}$, when $Q^i(x_{m-1}), Q^i(x_m) \rightarrow \bar{q}^i$

Condition one can still fail to be satisfied, even if both $A^{ij}\big(Q^i(x_{m-1})\big)$ and $A^{ij}\big(Q^i(x_m)\big)$ have real eigenvalues and eigenvectors. Furthermore, in smooth regions, which separate the isolated contact discontinuities, the variation in Q^i is of the order of $\mathcal{O}(\Delta x)$ and the Jacobean matrix is nearly constant: $A^{ij}\big(Q^i(x_{m-1})\big) \approx A^{ij}\big(Q^i(x_m)\big)$.

In its simplest form the new Jacobean matrix can be defined as

$$\hat{A}^{ij} = A^{ij}\big(\hat{Q}^i(x_{m-1/2})\big)\,, \qquad \hat{Q}^i(x_{m-1/2}) = \frac{Q^i(x_{m-1}) + Q^i(x_m)}{2}\,, \tag{88}$$

where $\hat{Q}^i(x_{m-1/2})$ is some average of $Q^i(x_{m-1})$ and $Q^i(x_m)$, in (88) simply corresponding to the mean value.

This definition creates a single state $\hat{q}^i_m$ with only one wave and the *Hugoniot loci* needed to find the exact Riemann solution are just straight lines pointing in the directions of the eigenvectors of $\hat{A}^{ij}$ (see Fig 10(a)). However, near shocks $Q^i(x_{m-1})$ and $Q^i(x_m)$ are far apart in state space and the true nonlinear structure is very different from the eigenstructure of $\hat{A}^{ij}$ in (88). Both values of Q^i should therefore be connected by a single 2-shock. In this case *Roe linearization* can offer an elegant way to compute the exact solutions, [4].

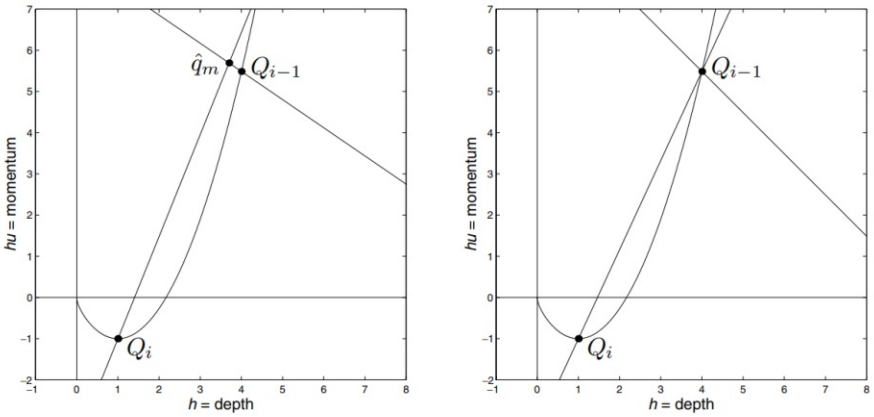

Figure 10: Calculation of the solutions $Q_i = Q^i(x_m)$, $Q_{i-1} = Q^i(x_{m-1})$, connected by a 2-shock in an example governed by the shallow water equations, [4], using
(a) the average $\hat{Q}^i$ from (88),
(b) the Roe average, [4].

Roe Linearization

The additional requirement for *Roe linearization* states, that if $Q^i(x_{m-1})$ and $Q^i(x_m)$ are connected by a single wave in the true Riemann solution, then $\mathcal{W}_1^i$ should also be an eigenvector of $\hat{A}^{ij}$. If this statement is true, then the linearized Riemann solution also consists of this single wave and agrees with the exact solution, [4].

This condition can be phrased mathematically by using the one-dimensional form of (71) and assuming that q^i is constant inside each cell, the following expression can be obtained

$$s\big(Q^i(x_m) - Q^i(x_{m-1})\big) = f^i\big(Q^i(x_m)\big) - f^i\big(Q^i(x_{m-1})\big)\,, \tag{89}$$

where $s = \Delta x_m/\Delta t_n$ is the wave speed and the flux vector f^i represents both the in- and out-going fluxes: $f^i\big(Q^i(x_m)\big) = f^i_{in}\big(Q^i(x_m)\big)$, $f^i\big(Q^i(x_{m-1})\big) = f^i_{out}\big(Q^i(x_{m-1})\big)$.

The variation of Q^i is also the solution to the linearized Riemann problem, hence

$$\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big)\left(Q^i(x_m) - Q^i(x_{m-1})\right) = s\left(Q^i(x_m) - Q^i(x_{m-1})\right). \tag{90}$$

Combining (89) with (90) yields

$$\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big)\,\Big(Q^i(x_m)-Q^i(x_{m-1})\Big)=f^i\big(Q^i(x_m)\big)-f^i\big(Q^i(x_{m-1})\big)\,. \tag{91}$$

(91) is another essential requirement for the conservative nature of the system. Physical quantities are conserved if the *flux-difference-splitting* formula (85) is valid and this in turn is only satisfied for the linearized Riemann solver if the condition (91) is fulfilled.

One way to obtain a matrix satisfying (91) is by integrating $\hat{A}^{ij}$ over a suitable path in state space between $Q^i(x_{m-1})$ and $Q^i(x_m)$. q^i can therefore be parameterized like

$$q^i(\eta)=Q^i(x_{m-1})+\big(Q^i(x_m)-Q^i(x_{m-1})\big)\,\eta\,, \tag{92}$$

for $0\leq\eta\leq 1$. The flux difference can then be written as the line integral

$$\begin{aligned} f^i\big(Q^i(x_m)\big)-f^i\big(Q^i(x_{m-1})\big)&=\int_0^1\frac{\mathrm{d}f^i\big(q^i(\eta)\big)}{\mathrm{d}\eta}\,\mathrm{d}\eta\\ =\int_0^1\frac{\mathrm{d}f^i\big(q^i(\eta)\big)}{\mathrm{d}q^j}\,\frac{\mathrm{d}q^j}{\mathrm{d}\eta}\,\mathrm{d}\eta&=\left(\int_0^1\frac{\mathrm{d}f^i\big(q^i(\eta)\big)}{\mathrm{d}q^j}\,\mathrm{d}\eta\right)\Big(Q^j(x_m)-Q^j(x_{m-1})\Big)\,, \end{aligned} \tag{93}$$

since $\mathrm{d}q^i/\mathrm{d}\eta=\big(Q^i(x_m)-Q^i(x_{m-1})\big)$ is constant and can be pulled out of the integral.

Inserting (91) into (93) results in the already proposed claim

$$\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big)=\int_0^1\frac{\mathrm{d}f^i\big(q^i(\eta)\big)}{\mathrm{d}q^j}\,\mathrm{d}\eta\,. \tag{94}$$

The main issue for solving (94) is to evaluate this integral in closed form for most nonlinear problems. Roe was the first author to circumvent this problem for the Euler equations by introducing a *parameter vector* $z^i(q^i)$, with the change of variables finally leading to an integral that is much easier to evaluate, [4]. The mapping is assumed to be invertible, meaning that $q^i(z^i)$ is also known and that the flux vector f^i can also be written as a function of z^i, or more precisely: $f^i=f^i(q^i(z^i))$.

Rather than integrating along the path of (92), the new one is defined as

$$z^i(\eta)=Z^i(x_{m-1})+\big(Z^i(x_m)-Z^i(x_{m-1})\big)\,\eta\,, \tag{95}$$

where Z^i is the average value of z^i of the m-th cell: $Z^i(x_m)=z^i\big(Q^i(x_m)\big)$.

With the new definition of the line path in (95), two line integrals can be constructed. The first one resembles the flux formula in (93), but using the new parameter z^i,

$$f^i\big(Q^i(x_m)\big) - f^i\big(Q^i(x_{m-1})\big) = \left(\int_0^1 \frac{\mathrm{d}f^i\big(z^i(\eta)\big)}{\mathrm{d}z^j}\,\mathrm{d}\eta \right) \Big(Z^j(x_m) - Z^j(x_{m-1}) \Big)\,. \tag{96}$$

The second line integral describes the relation between Q^i and Z^i and looks like

$$Q^i(x_m) - Q^i(x_{m-1}) = \left(\int_0^1 \frac{\mathrm{d}q^i\big(z^i(\eta)\big)}{\mathrm{d}z^j}\,\mathrm{d}\eta \right) \Big(Z^j(x_m) - Z^j(x_{m-1}) \Big)\,. \tag{97}$$

Defining the line integrals in (96) and (97) as *linearization matrices* yields

$$\begin{aligned} f^i\big(Q^i(x_m)\big) - f^i\big(Q^i(x_{m-1})\big) &= \hat{C}^{ij}\big(Z^i(x_{m-1/2})\big)\ \big(Z^j(x_m) - Z^j(x_{m-1})\big)\,, \\ Q^i(x_m) - Q^i(x_{m-1}) &= \hat{B}^{ij}\big(Z^i(x_{m-1/2})\big)\ \big(Z^j(x_m) - Z^j(x_{m-1})\big)\,. \end{aligned} \tag{98}$$

With the introduction of $\hat{B}^{ij}$ and $\hat{C}^{ij}$, $\hat{A}^{ij}$ can be calculated to

$$\hat{A}^{ij}\big(Q^i(x_{m-1/2})\big) = \hat{C}^{ij}\big(Z^i(x_{m-1/2})\big)\ \hat{B}^{ij^{-1}}\big(Z^i(x_{m-1/2})\big)\,. \tag{99}$$

To find a simple solution for the linearization matrices, z^i must be chosen in a way that both $\mathrm{d}q^i/\mathrm{d}z^j$ and $\mathrm{d}f^i/\mathrm{d}z^j$ have components that are polynomials with respect to z^i. Therefore the path in (95) will also only consist of polynomials, hence it is easy to integrate, [4].

Roe Solver for the Euler Equations

For the Euler equations Roe proposed the following parameter z^i, [16]:

$$z^i = \sqrt{\rho} \begin{pmatrix} 1 \\ u \\ w \\ H \end{pmatrix} \tag{100}$$

Therefore the non-dimensional flux vectors f_r^i and f_z^i in (44) can be transformed to

$$f_r^i = \begin{pmatrix} z_1 z_2 \\ z_2^2 + \frac{p(z^i)}{\kappa_0} \\ z_2 z_3 \\ \frac{z_2}{z_1}\left(E(z^i) + (\kappa_0 - 1)\, p(z^i)\right) \end{pmatrix}, \quad f_z^i = \begin{pmatrix} z_1 z_3 \\ z_2 z_3 \\ z_3^2 + \frac{p(z^i)}{\kappa_0} \\ \frac{z_3}{z_1}\left(E(z^i) + (\kappa_0 - 1)\, p(z^i)\right) \end{pmatrix}, \tag{101}$$

with E and p being functions depending on z^i:

$$E(z^i) = z_1 z_4 + (\kappa_0 - 1)^2 \frac{z_2^2 + z_3^2}{2}, \quad p(z^i) = z_1 z_4 - (\kappa_0 - 1) \frac{z_2^2 + z_3^2}{2}. \tag{102}$$

Considering the eigenvalues and eigenvectors of A_r^{ij} and A_z^{ij} in section 4.1, similar results can be achieved. Therefore the eigenvalue problem must only be solved for one of the Jacobean matrices and the solutions can easily be transformed to obtain the quantities for the other matrix. In this case the eigenvalue problem will be solved for A_r^{ij}, or more precisely for the linearized form $\hat{A}_r^{ij}$.

The main goal of this section is to evaluate the linearization matrices $\hat{B}_r^{ij}$ and $\hat{C}_r^{ij}$. Therefore $\mathrm{d}f_r^i/\mathrm{d}z^j$ and $\mathrm{d}q^i/\mathrm{d}z^j$ must be calculated first,

$$\frac{\mathrm{d}f_r^i}{\mathrm{d}z^j} = \begin{pmatrix} z_2 & z_1 & 0 & 0 \\ \frac{z_4}{\kappa_0} & \frac{\kappa_0+1}{\kappa_0} z_2 & -\frac{\kappa_0-1}{\kappa_0} z_3 & \frac{z_1}{\kappa_0} \\ 0 & z_3 & z_2 & 0 \\ 0 & \kappa_0 z_4 & 0 & \kappa_0 z_2 \end{pmatrix}, \tag{103}$$

$$q^i = \begin{pmatrix} z_1^2 \\ z_1 z_2 \\ z_1 z_3 \\ E(z^i) \end{pmatrix} \quad \rightarrow \quad \frac{\mathrm{d}q^i}{\mathrm{d}z^j} = \begin{pmatrix} 2z_1 & 0 & 0 & 0 \\ z_2 & z_1 & 0 & 0 \\ z_3 & 0 & z_1 & 0 \\ z_4 & (\kappa_0 - 1)^2 z_2 & (\kappa_0 - 1)^2 z_3 & z_1 \end{pmatrix}. \tag{104}$$

The linear form of (103) and (104) confirms that z^i was chosen correctly, considering the simple solutions for the path integrals in (96) and (97). Therefore only the following integral must be evaluated for each element of z^i:

$$\int_0^1 z_p(\eta)\, \mathrm{d}\eta = \frac{Z_p(x_{m-1}) + Z_p(x_m)}{2} = \bar{Z}_p, \tag{105}$$

where $\bar{Z}_p = \bar{Z}_p(x_{m-1/2})$, and $\bar{Z}^i = (\bar{Z}_1, ..., \bar{Z}_4)$ is the discretized mean vector of z^i of the interface between the m-th and $(m-1)$-th cell.

This means that the linearization matrices can just be expressed by the relation

$$\hat{C}_r^{ij} = \frac{\mathrm{d}f_r^i}{\mathrm{d}z^j}\left(\bar{Z}^i\right), \quad \hat{B}_r^{ij} = \frac{\mathrm{d}q^i}{\mathrm{d}z^j}\left(\bar{Z}^i\right). \tag{106}$$

Inverting $\hat{B}_r^{ij}$ yields

$$\hat{B}_r^{ij^{-1}} = \begin{pmatrix} \frac{1}{2\bar{Z}_1} & 0 & 0 & 0 \\ -\frac{\bar{Z}_2}{2\bar{Z}_1^2} & \frac{1}{\bar{Z}_1} & 0 & 0 \\ -\frac{\bar{Z}_3}{2\bar{Z}_1^2} & 0 & \frac{1}{\bar{Z}_1} & 0 \\ (\kappa_0-1)^2\,\frac{\bar{Z}_2^2+\bar{Z}_3^2}{\bar{Z}_1^3} - \frac{\bar{Z}_4}{2\bar{Z}_1^2} & -(\kappa_0-1)^2\,\frac{\bar{Z}_2}{\bar{Z}_1^2} & -(\kappa_0-1)^2\,\frac{\bar{Z}_3}{\bar{Z}_1^2} & \frac{1}{\bar{Z}_1} \end{pmatrix}. \tag{107}$$

Using (99), $\hat{A}_r^{ij}$ can finally be calculated to

$$\hat{A}_r^{ij} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ (\kappa_0-1)\,\frac{\hat{u}^2+\hat{w}^2}{2} - \hat{u}^2 & (3-\kappa_0)\,\hat{u} & -(\kappa_0-1)\,\hat{w} & \frac{1}{\kappa_0} \\ -\hat{u}\hat{w} & \hat{w} & \hat{u} & 0 \\ \kappa_0\hat{u}\left((\kappa_0-1)^2\,\frac{\hat{u}^2+\hat{w}^2}{2} - \hat{H}\right) & \kappa_0\left(\hat{H} - (\kappa_0-1)^2\,\hat{u}^2\right) & -\kappa_0(\kappa_0-1)\,\hat{u}\hat{w} & \kappa_0\hat{u} \end{pmatrix}, \tag{108}$$

which yields the same result as in (45), but with the *Roe-averages*:

$$\begin{aligned} \hat{u} = \frac{\bar{Z}_2}{\bar{Z}_1} &= \frac{\sqrt{\rho(x_{m-1})}\,u(x_{m-1}) + \sqrt{\rho(x_m)}\,u(x_m)}{\sqrt{\rho(x_{m-1})} + \sqrt{\rho(x_m)}}\,, \\ \hat{w} = \frac{\bar{Z}_3}{\bar{Z}_1}\,, &\quad \hat{H} = \frac{\bar{Z}_4}{\bar{Z}_1}\,. \end{aligned} \tag{109}$$

Therefore all eigenvalues and eigenvectors can be obtained by simply exchanging all quantities in (51) with their corresponding Roe-averages, whereas (15) is used to calculate $\hat{c}$.

The HLLE Solver

In some situations linearized Riemann solvers can fail completely, giving non-physical solutions such as negative pressures or densities in the Euler equations. This can be the case for data that is near a vacuum state or where strong expansions arise, which both occur in this thesis' scenario. The key requirement for avoiding these non-physical effects is for the Riemann solver to be *positively conservative*. This means that the mass and energy density must always remain positive for any physical initial data, [4].

While Godunov's method with the exact Riemann solver is positively conservative, the authors of [17] were able to prove this requirement for the *HLLE solver* as well. This simple approximate Riemann solver, named after its inventors Harten, Lax, van Leer and Einfeldt, estimates the smallest and largest wave speeds arising in the Riemann solution, called $s_1(x_{m-1/2})$ and $s_2(x_{m-1/2})$. Therefore a new middle state $\hat{Q}^i(x_{m-1/2})$ between the waves is created, as illustrated in figure 4(b). The waves are then defined as

$$\mathcal{W}_1^i(x_{m-1/2}) = \hat{Q}^i(x_{m-1/2}) - Q^i(x_{m-1})\,, \qquad \mathcal{W}_2^i(x_{m-1/2}) = Q^i(x_m) - \hat{Q}^i(x_{m-1/2})\,. \tag{110}$$

Extending (89) for two waves yields

$$\begin{aligned} s_1(x_{m-1/2})\left(\hat{Q}^i(x_{m-1/2}) - Q^i(x_{m-1})\right) + s_2(x_{m-1/2})\left(Q^i(x_m) - \hat{Q}^i(x_{m-1/2})\right) = \\ f^i\big(Q^i(x_m)\big) - f^i\big(Q^i(x_{m-1})\big)\,. \end{aligned} \tag{111}$$

Therefore $\hat{Q}^i$ can be calculated to

$$\hat{Q}^i(x_{m-1/2}) = \frac{f^i\big(Q^i(x_m)\big) - f^i\big(Q^i(x_{m-1})\big) - s_2(x_{m-1/2})\,Q^i(x_m) + s_1(x_{m-1/2})\,Q^i(x_{m-1})}{s_1(x_{m-1/2}) - s_2(x_{m-1/2})}\,. \tag{112}$$

The only parameters left for definition are the wave speeds s_p. In the original HLL method s_1 and s_2 were chosen to be some lower and upper bounds on all characteristic speeds λ_p that arise in the true Riemann solution. Einfeldt [18], however, defined the speeds as follows

$$\begin{aligned} s_1(x_{m-1/2}) &= \min_p \left(\min\left(\lambda_p(x_m), \hat{\lambda}_p(x_{m-1/2})\right)\right), \\ s_2(x_{m-1/2}) &= \max_p \left(\max\left(\lambda_p(x_{m+1}), \hat{\lambda}_p(x_{m-1/2})\right)\right), \end{aligned} \tag{113}$$

or especially for this thesis, knowing that $u-c$ must be the smallest and $u+c$ the biggest value,

$$
\begin{aligned}
s_1(x_{m-1/2}) &= \min\left(u(x_m) - c(x_m),\, \hat{u}(x_{m-1/2}) - \hat{c}(x_{m-1/2})\right), \\
s_2(x_{m-1/2}) &= \max\left(u(x_{m+1}) + c(x_{m+1}),\, \hat{u}(x_{m-1/2}) + \hat{c}(x_{m-1/2})\right).
\end{aligned}
\tag{114}
$$

Compared to the HLL solver, Einfeldt's adaptation (the HLLE solver) gives sharper results for shock waves in general and for data connected only by a single shock wave, the approximate solution exactly agrees with the true solution, due to the properties of the Roe solver.

4.5.2 Geometry of the Bottle and the Cork

The shapes of the bottle and the moving cork are defined by the initial values $Q^i(t_0)$ and the auxiliary fields, which are responsible for managing the boundary conditions. This section describes the functions needed to define the initial values. The bottle opening's diameter $d_0 = 1.8\,\mathrm{cm}$, [1], is chosen as a reference length, so that in the non-dimensional case it is equal to one and all other sizes are adjusted accordingly. The form of the bottle resembles a polynomial of second order, with the assumption of the starting point being two times bigger than the opening and that the function's minimum is at the bottle's ending point (see figure 1),

$$
\begin{gathered}
f_B(x) = ax^2 + bx + c\,, \qquad f_B(0) = 1\,, \qquad f_B(L) = \frac{1}{2}\,, \qquad \frac{\mathrm{d}f_B(L)}{\mathrm{d}x} = 0 \\
\rightarrow \qquad f_B(x) = \frac{x^2}{2L^2} - \frac{x}{L} + 1
\end{gathered}
\tag{115}
$$

L is the non-dimensional length of the bottle and is determined by the given volume of the bottleneck $V_r = 25\,\mathrm{ml}$, [1]. For the derivation the volume integral of a cylinder with varying width is used,

$$
V = \int_0^{2\pi} \mathrm{d}\phi \int_0^{L} \mathrm{d}x \int_0^{f(x)} r\,\mathrm{d}r = \pi \int_0^{L} \mathrm{d}x\; f_B^2(x)\,. \tag{116}
$$

At first, the dimensional volume must be calculated. Multiplying f_B and L by the reference length d_0, (116) results in

$$
V = \frac{7\pi}{15}\, d_0^3\, L\,. \tag{117}
$$

Since $V = V_r$ is a constant, L finally results in

$$L = \frac{15}{7\pi} \frac{V_r}{d_0^3} \,. \tag{118}$$

The same procedure is then used for the cork, only exchanging f_B with f_C in (116), which is defined as a linear function with the bottle opening's radius at the end point and a slightly wider diameter at the start, due to its elastic expansion after the cork popping. The cork's length is assumed to be twice its end diameter,

$$\begin{aligned} f_C(x) = ax + b \,, \qquad f_C(0) = \frac{3}{5} \,, \qquad f_C(2) = \frac{1}{2} \\ \rightarrow \qquad f_C(x) = \frac{3}{5} - \frac{x}{20} \,. \end{aligned} \tag{119}$$

(116) is then evaluated for the function of the cork, whereas this time the cork's volume is the desired quantity,

$$V_C = \frac{91\pi}{150} d_0^3 \,. \tag{120}$$

Since the spatial domain of the system is represented by a grid of finite cells, the bottle's and cork's geometries must also be discretized. Therefore two functions are needed to define the cells, occupied by the bottle and cork, in the r- and z-direction respectively. The dimension z lies on the x-axis and r on the y-axis.

The function *Ind_x*(x_{st}, x_{end}) returns the indices of the starting- and ending points on the x-axis. This is done by calculating the absolute distance between each cell interface $x_{m-1/2}$ and the point x_{st} or x_{end} for increasing m. If the mentioned distance reaches its minimum value, the corresponding index is assigned to the starting or the ending point and is returned. While for the smaller point the initial index starts at 0, for the bigger one m_{init} can already be set to the previously calculated index for x_{st}. The only disadvantage is that the size of objects is altered slightly. However, this effect decreases with increasing total cell numbers and every other method changes the size as well, as long as the grid is not perfectly aligned with the object's dimensions. In the latter case the recommended method also preserves the true size of the object.

Ind_y(x_{st}, x_{end},*func*) on the other hand has an arbitrary function as an additional input parameter because the index of the object's wall in the y-direction depends on that function with respect to x. Hereby *func* is calculated for every cell interface between x_{st} and x_{end}. Then the interfaces in the y-direction are scanned along increasing m, starting at the cell in the middle, which corresponds to $y = 0$. Independent of the fact that the total number of cell interfaces N is either even or odd, the scanning starts at $int(N/2)$,

which is exactly $N/2$ for even and $(N-1)/2$ for odd numbers of N. If the position of an interface $y_{m-1/2}$ eventually becomes greater than the function's value at $x_{m-1/2}$, the corresponding index is again assigned to the wall's y-position. The indices for the underside of the bottle and cork are calculated to $m_{st} = N - m_{end} - 1$. A graphical description is depicted in figure 11.

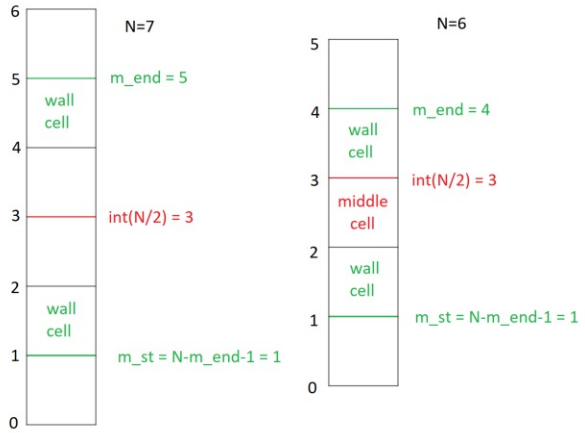

Figure 11: Illustration showing the position of m_{st} and m_{end} to further assign symmetrical wall cells.

Figure 11 shows that the wall cells must be symmetrical around the middle interface for odd and around the middle cell for even numbers of N. This difference explains, why m_{st} can be described by the same formula for both cases.

Filling in the cells with initial values from the bottom to the top wall cells is then done by the function *init_values(value)*. Because all velocities are initially zero, this function only has to be called for ρ and E. At first, all values inside the domain are set to 1, as discussed in section 3.1. Only for cells occupied by the bottle, the values must be higher. Therefore the non-dimensional pressure and temperature are defined as $p = p_r/p_0$ and $T = T_r/T_0$, with p_0, T_0 being the ambient air pressure and temperature as the reference quantities and p_r, T_r being the pressure and temperature inside the bottle (either 7.5 bar at 20 °C or 10.2 bar at 30 °C). Then ρ can be calculated with the non-dimensional form of (2) to $\rho = p/T$ and E with (12), in the case of u and w being 0, to $E = p$.

Finally, the following loop fills in the corresponding values

for i in range(len(j_top)): array[ix+i,j_bot[i]:j_top[i]] = value ,

where *ix* is the index of x_{st}, *j_bot* and *j_top* are the arrays storing the indices for the bottom and top wall of the bottle, returned by *Ind_y*, and *value* is either ρ or E.

The inside of the cork and the wall cells of the bottle are represented by *None* values and are also implemented by *init_values*. These cells must not have any numerical values, so that they do not interact with the fluid, except the ones which are used to fulfill the boundary conditions. The result of this approach can be seen in figure 12.

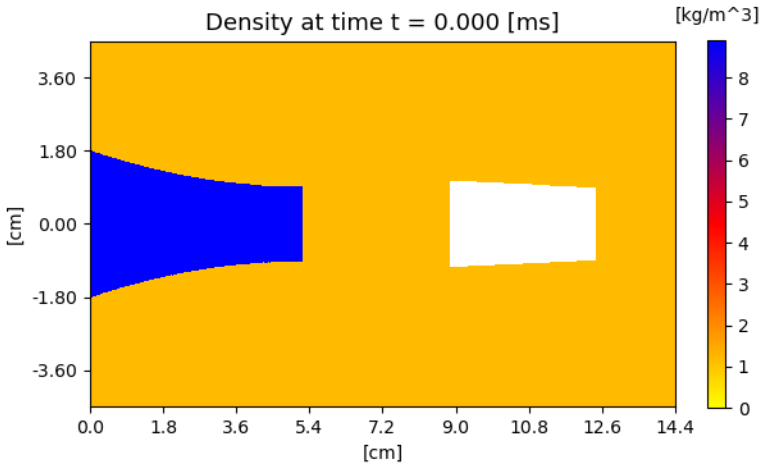

Figure 12: Initial density plot showing the distribution of values for a pressurized bottle (7.5 bar) and a flying cork with a total cell number of 800 for each dimension.

The white area in figure 12 corresponds to the *None* cell entries. These also exist around the bottle, but due to the walls' slim width of only two cells, they are not visible in the illustration. Note, that all graphs below, including figure 12, show dimensional quantities with the corresponding units and that a uniform grid with the same total cell numbers in both directions, but therefore different cell distances is used.

4.5.3 Source Term due to Axial Symmetry

A source term Ψ^i is usually required, when either a fluid source or sink is present. This fact transforms the conservation laws into balance laws. However, as seen in section 3.4, a physical problem must also contain Ψ^i, when its dimensions are reduced due to symmetries. This fact can be understood more thoroughly, when considering the given two-dimensional bottle. After the cork popping, the gas expands in three-dimensional space. Therefore the two-dimensional cross section must inevitably lose some of its otherwise conserved quantities, hence a source term is required.

There are two numerical ways of implementing the source term: the *fractional-step* or *strang-splitting* method. While the latter is more accurate, the algorithms are much more elaborate and the benefits can be neglected for large cell numbers. Therefore the first method will be used for the present work. The necessary assumption for both methods is that the source term solely depends on the unknown quantities: $\Psi^i = \Psi^i(q^i)$, which

fortunately is the case for (44). The fractional-step method is applied by first splitting (25) into two subproblems, that can be solved independently. The first problem describes the homogeneous equation system, that was already discussed in previous sections and the second one deals with the particular equation system

$$\frac{\partial q^i(r^i,t)}{\partial t} = \Psi^i\big(q^i(r^i,t)\big)\,. \tag{121}$$

Since the homogeneous quantities $Q_h^i(t_{n+1})$ have already been solved by (83), these functions are used to find a numerical method for solving (121). The *Runge-Kutta methods* are the most applicable single-step methods, meaning that the solution must only be known one time step prior to the desired one. The general solution of these methods, assuming Ψ^i does not possess an independent parameter of time, looks like

$$Q^i(t_{n+1}) = Q^i(t_n) + \Delta t_n \sum_{j=1} b_j\, k_j^i\,, \qquad k_j^i = \Psi^i\Big(Q^i(t_n) + \Delta t_n \sum_{l=1} a_{jl}\, k_l^i\Big)\,, \tag{122}$$

where b_j and a_{jl} are coefficients from the *Butcher-table*, [19]. Every method is then characterized by its own table. While for implicit methods the two sums in (122) can have the same upper index, resulting in iterative algorithms, for explicit ones the second sum should at most be added up to $l = j-1$. The easiest Runge-Kutta method is the *explicit Euler-method*, where $a_{1\,1} = 0$ and $b_1 = 1$, therefore the solution is just:
$Q^i(t_{n+1}) = Q^i(t_n) + \Delta t_n\, \Psi^i\big(Q^i(t_n)\big)\,.$

The total number of b_j or k_j^i defines the order of the method. In this thesis an algorithm of at least second order is sufficient and therefore the general second order Runge-Kutta method is used. Its Butcher-table is defined as

$$b^j = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\,, \qquad a^{jl} = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{pmatrix}\,. \tag{123}$$

The solution finally results in

$$\begin{gathered} k_1^i = \Psi^i\big(Q^i(t_n)\big)\,, \quad k_2^i = \Psi^i\Big(Q^i(t_n) + \frac{\Delta t_n}{2}\, k_1^i\Big)\,, \\ Q^i(t_{n+1}) = Q^i(t_n) + \Delta t_n\, k_2^i\ . \end{gathered} \tag{124}$$

In terms of the fractional-step method, $Q^i(t_n)$ must be exchanged with $Q_h^i(t_{n+1})$ in (122) and (124) to receive the total solution of the system at t_{n+1}.

Further note, that due to the inverse radial dependence of the source term, the total cell number in the r-direction must be even, therefore the total number of cell interfaces being odd, so that there is no middle cell with the position $r = 0$. This fact would otherwise cause singularities.

4.5.4 Numerical Implementation of the Boundary Conditions

So far $Q^i(x_m)$ could always be updated by considering the in-going flux from $Q^i(x_{m-1})$ and the out-going flux into $Q^i(x_{m+1})$. Unfortunately, the domain is always bounded by a finite number of grid cells and therefore the first and last cell do not possess the required neighboring information. This issue can be circumvented by extending the computational domain to include additional cells on either end, called *ghost cells*, whose values are set at the beginning of each time step. These values solely depend on the boundary conditions and are entirely decoupled from the choice of the numerical method, used for updating Q^i. While for Godunov's method the fluxes depend on two values of Q^i, as seen in (75), and therefore only one ghost cell is needed on either end, for any advanced method like a *high-resolution flux limiter*, discussed in the following section, at least two ghost cells are required, [4].

In this thesis outflow boundaries describe the fluid's behavior at the domain's edges and special internal no-penetration conditions are applied to explain the interaction between the fluid and the stationary bottle or the moving cork.

To let a substance freely flow out of a given volume, the cells outside this region are not allowed to influence the inner cell's values whatsoever. This is ensured by only permitting in-going internal fluxes, but not external ones. By applying a *zero-order extrapolation*, the ghost cells' values for the edges in the x-direction are determined as follows:

$$Q^i(x_j) = Q^i(x_{N_G+1})\,, \quad Q^i(x_{M+j}) = Q^i(x_M)\,, \quad j \in [1, N_G]\,, \tag{125}$$

where the overall first index is 1, M the last internal index to the right and N_G the number of ghost cells. This definition makes it obvious that external cells cannot influence internal ones, when sharing the same value, hence no in-going flux is present.

For any no-penetration condition the key requirement is that the fluid's velocity perpendicular to a given wall must be the speed of that object v_w in the given direction, in a stationary case being zero, so that no substance can pass through it. Because each wall is positioned at a cell interface, the cells' values on the left and right of the wall must somehow be combined to match the desired condition. The easiest way is just taking the average between these cells, meaning that $Q^i(x_{m-1/2}) = \big(Q^i(x_{m-1}) + Q^i(x_m)\big)/2$. This method can be extended to also include cells farther away.

Because $Q_2(x_{m-1/2})$ must be equal to $\rho(x_{m-1/2})\, v_w$ (Q_3 for a wall in the y-direction), the ghost cells for a wall on the right interface can be calculated to

$$Q_2(x_{m+j-1}) = 2\,\rho(x_{m-1/2})\, v_w - Q_2(x_{m-j})\,, \qquad j \in [1, N_G]\,. \tag{126}$$

For the case of a wall to the left, the values of Q_2 in (126) simply have to be exchanged.

All other quantities can be determined by mirroring their values on the cell interface,

$$Q^i(x_{m+j-1}) = Q^i(x_{m-j}) \quad \rightarrow \quad \rho(x_{m-1/2}) = \rho(x_{m-1}) = \rho(x_m), \qquad j \in [1, N_G]\ . \tag{127}$$

Numerically (126) and (127) are only applied if one cell is occupied by the fluid and the exact neighbor is characterized as a wall cell. The first auxiliary field $aux(0, :, :)$ is responsible for this characterization, by setting its value to 1 if the corresponding cell is filled with a fluid and 0 for describing a solid object. The function *aux_init()* performs this algorithm, which is the same one for filling in the initial *None* values. Then the no-penetration condition is called inside the *normal solver* only if the numerical neighboring condition is fulfilled.

For Godunov's method and therefore only one ghost cell on each edge, the gas emission of a bottle, without considering the cork, yields the results depicted in figure 13.

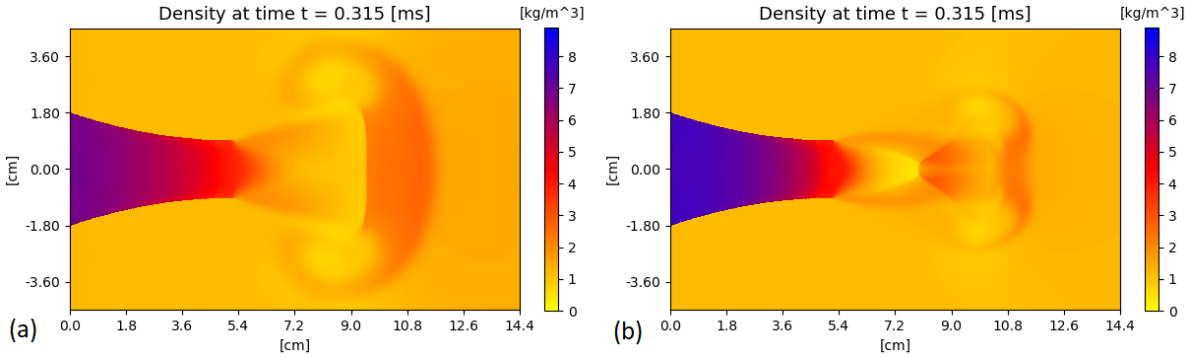

Figure 13: Density plot showing the gas emission of a pressurized bottle (7.5 bar) with a total cell number of 800 for each dimension. The left figure is calculated without a source term, corresponding to a planar flow and the second graph with one, describing a three-dimensional system with axial symmetry.

The left graph in figure 13 illustrates a two-dimensional bottle in Cartesian coordinates and the right graph only a cross section of a three-dimensional bottle in cylindrical coordinates, but with a reduced dimension due to axial symmetry. The left depiction truly lacks an extra dimension and therefore additional space to expand into. This explains, why its fluid is able to expand more rapidly into the given two dimensions than the gas in the second illustration, which also has to expand into the third dimension.

Note, that the ambient quantities for all simulations are $p_0 = 1.013\,\text{bar}$ [20], $\rho_0 = 1.204\,\text{kg/m}^3$ [21] and $T_0 = 20\,^\circ\text{C}$. Furthermore, the fluid inside the bottle is assumed to be air with $\kappa_0 = 7/5$ [22], so that the properties of multiple-phase mixtures, once the internal gas exits the bottle, can be neglected.

4.5.5 CFL-Condition and High Resolution Methods

While for the previously depicted numerical methods, mainly the Godunov method, precision is improved by generally decreasing Δx_m and Δt_n, there are conditions to be fulfilled, so that the numerical solution can even converge towards the true solution in the limit of $\Delta x_m\,, \Delta t_n \to 0$. One of the most important conditions, which can directly be implemented with the given CLAWPACK routines, is the *CFL-condition*, named after Courant, Friedrichs and Lewy. It states that a numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the equation system, [4]. This condition is necessary, but does not guarantee a stable numerical scheme. For figure 2(a) the domain of dependence $\mathcal{D}(X,T)$ must therefore lie within the interval: $X - \lambda_3\,T \leq \mathcal{D}(X,T) \leq X - \lambda_1\,T$.
Refining the grid with a fixed ratio of $r = \Delta t_n/\Delta x_m$, the interval can be transformed to $X - T/r \leq \mathcal{D}(X,T) \leq X + T/r\,T$, with the necessary CFL-condition for a three-point-method, such as Godunov's one, being

$$\nu = \frac{\Delta t_n}{\Delta x_m} \max_p |\lambda_p| \leq 1\,, \tag{128}$$

where ν is the *Courant number*.

If the CFL-condition is not satisfied, a change in the initial data at any point inside the domain of dependence would change the true solution at (X,T), but could have no effect on the numerical solution at this point. In CLAWPACK a desired, as well as a maximum Courant number can be set. While *cfl_max* should always be left at 1, *cfl_desired* should approximately be 1, but not exactly, unless the numerical solution is the same as the true solution. Therefore the desired value is set to default, being 0.9.

Because Godunov's method is only first-order accurate, it can cause numerical diffusion, yielding poor accuracy and smeared results. This method can be improved by inserting an additional *correction term* into (83),

$$\begin{aligned} Q^i(x_m,t_{n+1}) = Q^i(x_m,t_n) &- \frac{\Delta t_n}{\Delta x_m}\left(A^{ij}_+\,\Delta Q^j(x_{m-1/2},t_n) + A^{ij}_-\,\Delta Q^j(x_{m+1/2},t_n)\right) \\ &- \frac{\Delta t_n}{\Delta x_m}\left(\tilde{F}^i_{out}(x_m,t_n) - \tilde{F}^i_{in}(x_m,t_n)\right), \end{aligned} \tag{129}$$

where $\tilde{F}^i$ are the fluxes based on the waves, resulting from the Riemann solution, [4].

These correction terms can be described by considering a second order method such as the *Lax–Wendroff-method*, which is based on the truncated Taylor series expansion

$$q^i(x,t_{n+1}) = q^i(x,t_n) + \Delta t_n \frac{\partial q^i(x,t_n)}{\partial t} + \frac{\Delta t_n^2}{2}\frac{\partial^2 q^i(x,t_n)}{\partial t^2} + \mathcal{O}\big(\Delta t_n^3\big)\,. \tag{130}$$

By inserting the one-dimensional, homogeneous form of (25) and its second derivative with respect to time, being

$$\frac{\partial q^i}{\partial t} = -A^{ij}\frac{\partial q^j}{\partial x}\,, \qquad \frac{\partial^2 q^i}{\partial t^2} = -A^{ij}\frac{\partial^2 q^j}{\partial x\,\partial t} = \big(A^{ij}\big)^2\frac{\partial^2 q^j}{\partial x^2}\,, \tag{131}$$

into (130), yields

$$q^i(x,t_{n+1}) = q^i(x,t_n) - \Delta t_n\, A^{ij}\frac{\partial q^j(x,t_n)}{\partial x} + \frac{\Delta t_n^2}{2}\big(A^{ij}\big)^2\frac{\partial^2 q^j(x,t_n)}{\partial x^2} + \mathcal{O}\big(\Delta t_n^3\big)\,. \tag{132}$$

Keeping only the first three terms and replacing the spatial derivatives with central finite difference approximations, finally results in the *Lax–Wendroff-method*,

$$\begin{aligned} Q^i(x_m,t_{n+1}) = Q^i(x_m,t_n) &- \frac{\Delta t_n}{2\,\Delta x_m}A^{ij}\Big(Q^j(x_{m+1},t_n) - Q^j(x_{m-1},t_n)\Big) \\ &+\frac{1}{2}\left(\frac{\Delta t_n}{\Delta x_m}\right)^2\big(A^{ij}\big)^2\Big(Q^j(x_{m-1},t_n) - 2\,Q^j(x_m,t_n) + Q^j(x_{m+1},t_n)\Big)\,. \end{aligned} \tag{133}$$

Transforming (133) into the form of (129), defines the correction terms as

$$\begin{aligned} \tilde{F}^i_{out}(x_m,t_n) &= \frac{1}{2}\left(A^{ij}_+ - A^{ij}_- - \frac{\Delta t_n}{\Delta x_m}\big(A^{ij}\big)^2\right)\Delta Q^j(x_{m+1/2},t_n)\,, \\ \tilde{F}^i_{in}(x_m,t_n) &= \frac{1}{2}\left(A^{ij}_+ - A^{ij}_- - \frac{\Delta t_n}{\Delta x_m}\big(A^{ij}\big)^2\right)\Delta Q^j(x_{m-1/2},t_n)\,. \end{aligned} \tag{134}$$

In spite of its better accuracy, the Lax–Wendroff-method cannot be used for the present work due to its dispersive nature near discontinuities. Comparing figure 14 with 13(b), a major difference can be seen in regions, where r is very small. The reason for the unrealistic 'stripe' is that the source term has a singularity at $r = 0$ and therefore the Taylor series expansion is not valid for updating over this discontinuity.

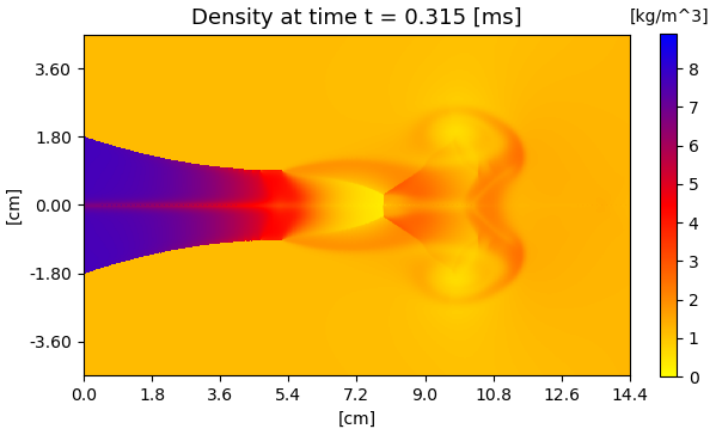

Figure 14: Density distribution $\rho(r, z, t = 0.315\,\text{ms})$ showing the gas emission of a pressurized bottle (7.5 bar) with a total cell number of 800 for each direction. The solution is evaluated with the Lax–Wendroff-method.

High Resolution methods are a way of combining the best features of both the Godunov and the Lax–Wendroff-method. Second-order accuracy is obtained where possible, but not for regions, where the solution does not behave smoothly. Numerically this is performed by the introduction of *flux limiters*, which change the magnitude of the correction terms that are used, depending on how the solution behaves. These limiters must be applied in such a way that the discontinuous portion of the solution does not cause any dispersion, while the smooth part remains accurate. A low-order flux formula $\mathcal{F}_L^i$ such as (84) and a high-order flux $\mathcal{F}_H^i$ like (134) are combined to obtain the general form of a *flux-limiter-method*,

$$\begin{aligned} F_{out}^i(x_m) &= \mathcal{F}_L^i + \Phi(x_{m+1/2})\left(\mathcal{F}_H^i - \mathcal{F}_L^i\right), \\ F_{in}^i(x_m) &= \mathcal{F}_L^i + \Phi(x_{m-1/2})\left(\mathcal{F}_H^i - \mathcal{F}_L^i\right), \end{aligned} \tag{135}$$

where $\mathcal{F}^i = \mathcal{F}^i\big(Q^i(x_{m-1}), Q^i(x_m)\big)$ for F_{in}^i, $\mathcal{F}^i = \mathcal{F}^i\big(Q^i(x_{m+1}), Q^i(x_m)\big)$ for F_{out}^i and $0 \leq \Phi \leq 1$.

Although there are various limiter algorithms, unfortunately no standard method inside the CLAWPACK package is able to solve the issue depicted in figure 14, when the order of the solver, as well as the number of ghost cells are set to two. Only individually these settings result in a smooth solution, the problem being that the latter parameter must be bigger than one in order for the Lax–Wendroff-method to function properly. Therefore the first-order Godunov method, which also requires only a single ghost cell on either side, and the default *TVD minmod limiter* are used for all further simulations.

5 Discharge Flow Analysis

While the general flow out of the pressurized bottle was already discussed in the previous section, this section concentrates on the analysis of the system's quantities as a function of time and space and their dependencies on each other. As discussed in section 3.2, the two main assumptions of the flow are being inviscid and adiabatically expanding after the cork popping. The latter allows the definition of the relation between the fluid's temperature and pressure at the bottle opening at $r = 0$ as a function of time. This relation is derived by defining the total derivative of (3) as

$$\mathrm{d}e = T\,\mathrm{d}s + \frac{p}{\rho^2}\,\mathrm{d}\rho, \qquad \mathrm{d}h = T\,\mathrm{d}s + \frac{1}{\rho}\,\mathrm{d}p, \tag{136}$$

An additional assumption is the isentropic change of state, hence $\mathrm{d}s{=}0$. Inserting (136) into (4) results in the already non-dimensional form

$$\kappa_0 = \frac{\rho}{p}\frac{\mathrm{d}p}{\mathrm{d}\rho} \tag{137}$$

Multiplying by $\mathrm{d}\rho/\rho$, integrating on both sides and using (2), finally yields the desired relation

$$T(t) = T_r \left(\frac{p(t)}{p_r}\right)^{\frac{\kappa_0 - 1}{\kappa_0}} . \tag{138}$$

To validate (138) numerically, two temperature graphs are created and compared with each other by depicting them in one plot. The first one is calculated by combining (12) with the non-dimensional ideal gas equation to

$$T = \frac{E}{\rho} - \kappa_0(\kappa_0 - 1)\,\frac{u^2 + w^2}{2} , \tag{139}$$

and later multiplying it by T_0 to get the dimensional temperature. The second function is evaluated with (138).

The quantities' numerical values are returned by the program at every time step inside the instance *solver.before_step*, which calls the custom function *Flow_calc(solver,state)*. The plotting is then performed outside *VisClaw* with the function *plot_flow(t,Temp, Temp2,Press)*, which in turn uses *Matplotlib*, before the two-dimensional heat-maps are plotted. Figure 15 shows that (138) and (139) are very similar and therefore the assumption of an isentropic expansion at the bottle opening is justified. This fact offers a reliable scenario to test the average error between the two graphs for increasing total cell numbers. Furthermore, the program's overall runtime can also be measured. For this work the numerical simulation was executed on an *Intel* i7 -9700K processor and a working memory of 32 GB.

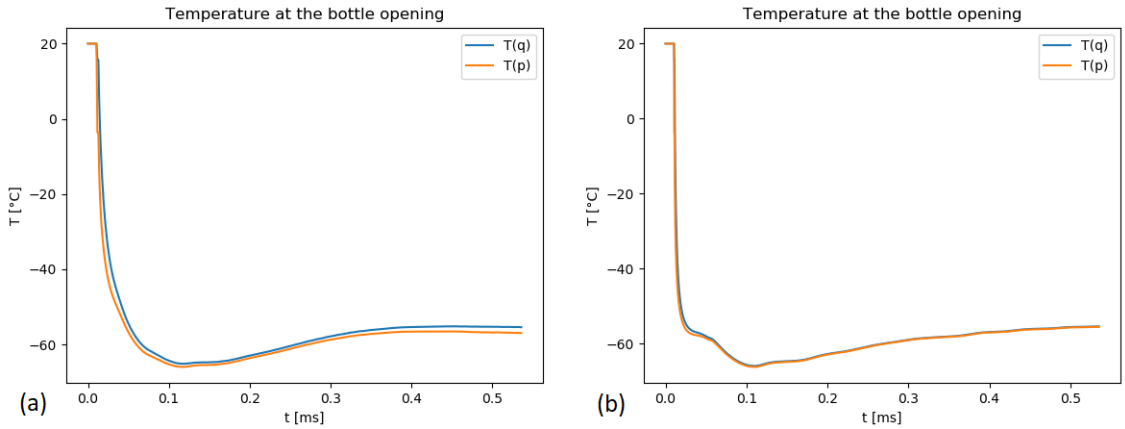

Figure 15: Comparison between (138) $T(p)$ and (139) $T(q^i)$ at the bottle opening with $T_r = 20\,°\text{C}$ for a total cell number of (a) 200 and (b) 800.

total cell number	average error	1^{st} run	2^{nd} run	3^{rd} run	average runtime
200	1.519 °C	9.948 s	10.057 s	10.021 s	10.009 s
400	0.654 °C	68.751 s	68.313 s	68.876 s	68.647 s
600	0.437 °C	237.484 s	237.253 s	237.124 s	237.287 s
800	0.338 °C	496.388 s	495.870 s	496.282 s	496.180 s

Table 1: Average error between (138) $T(p)$ and (139) $T(q^i)$ at the bottle opening with $T_r = 20\,°\text{C}$ and the average runtime of the Python program (see appendix) for increasing total cell numbers.

It can be seen that the overall precision is improved, while the compilation time is exponentially increased when considering bigger total cell numbers and therefore smaller grid step sizes. The spatial distribution of the fluid's temperature at a given point in time is depicted in figure 16.

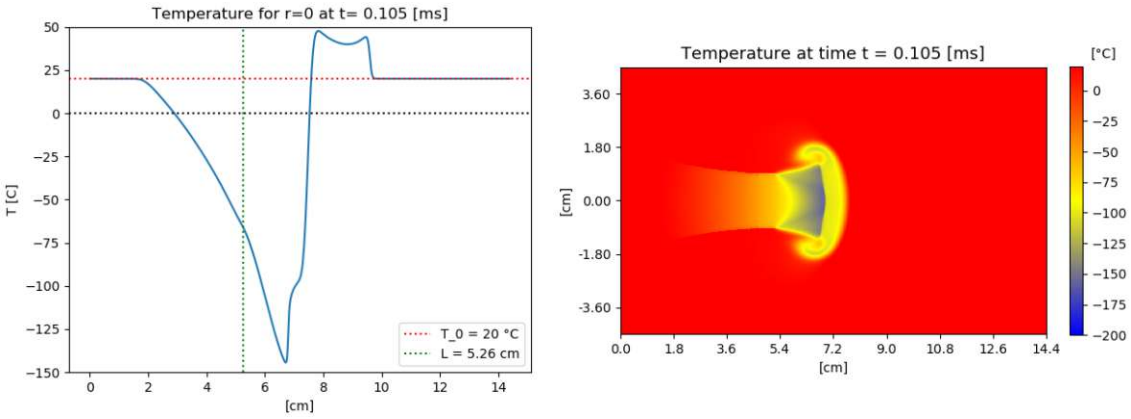

Figure 16: Temperature distribution $T(r, z, t)$ (right) and at the axis $r = 0$ (left) at time $t = 0.105\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

While at the bottle opening $z = 5.26\,\text{cm}$ the temperature never drops below $-70\,°\text{C}$, at around $z = 8\,\text{cm}$ temperatures can reach values even below $-180\,°\text{C}$ (see figure 17), which by far falls below the freezing temperature of CO_2, being $-78.5\,°\text{C}$ at atmospheric pressure, [23]. This fact allows a phase transition of the CO_2 gas and leads therefore to the formation of dry ice particles in front of the bottle, but not inside of it.

Although the simulation is performed by assuming that the fluid is just air, the same phenomenon still occurs when changing the fluid inside the whole domain to be pure CO_2, by setting κ_0 to 9/7, [22].

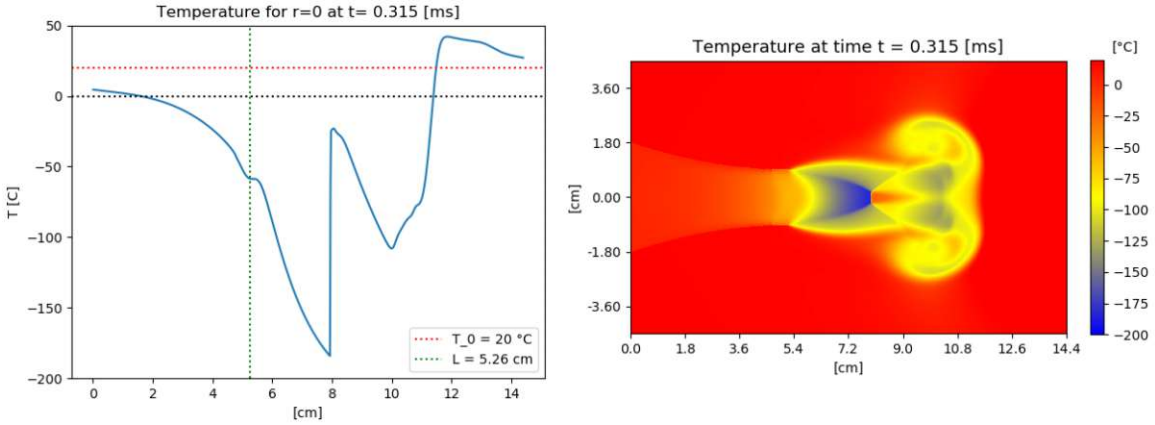

Figure 17: Temperature distribution $T(r, z, t)$ (right) and at the axis $r = 0$ (left) at time $t = 0.315\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

For bottles stored at $30\,°\text{C}$ the minimum temperature even drops further from $T_{min}(T_r = 20\,°\text{C}) = -184.17\,°\text{C}$ to $T_{min}(T_r = 30\,°\text{C}) = -197.19\,°\text{C}$ (see figure 18).

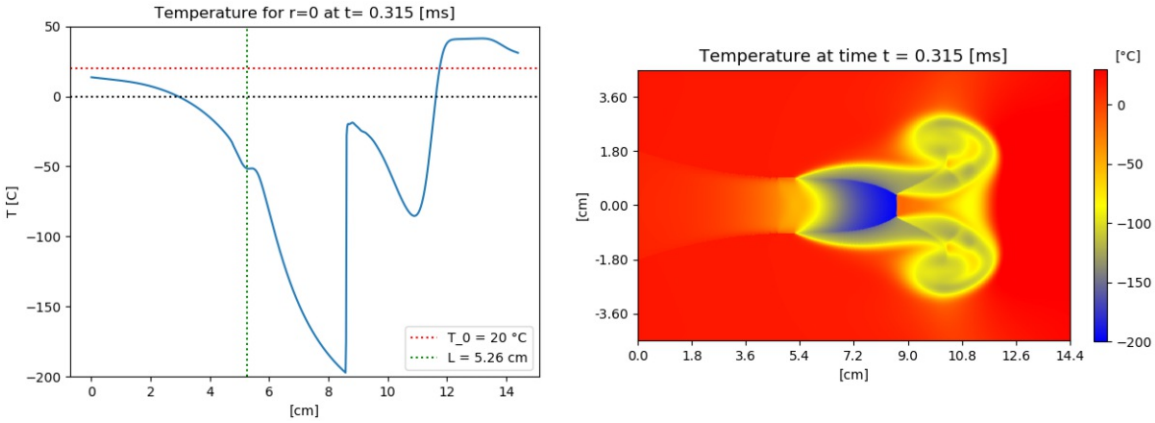

Figure 18: Temperature distribution $T(r, z, t)$ (right) and at the axis $r = 0$ (left) at time $t = 0.315\,\text{ms}$ for a total cell number of 800 and $T_r = 30\,°\text{C}$.

This can be explained by the fact, that a higher initial bottle pressure results in a greater temperature decrease, such as the relation in (138) suggests. However, in reality the fluid is generally warmer due to inner friction, which was not considered in this simulation. Figures 17 and 18 also illustrate the temperature shock discontinuity. This jump appears in distributions of other quantities as well.

5.1 Mach Number Distribution

This section discusses the velocity of a partially supersonic flow, assuming a steady quasi-one-dimensional stream through a bottle with a slightly varying cross section $A = A(z)$. Hereby the main reference quantity is the local *Mach number*, defined as

$$M = \frac{\sqrt{u^2 + w^2}}{c} \,. \tag{140}$$

The previous assumptions lead to $M = w/c$ and allow the integral form of the general Euler equations (26)-(28) and the isentropic equation (137) to be simplified to

$$\frac{\mathrm{d}}{\mathrm{d}z} \ln\left(\dot{m}\right) = \frac{1}{A}\frac{\mathrm{d}A}{\mathrm{d}z} + \frac{1}{w}\frac{\mathrm{d}w}{\mathrm{d}z} + \frac{1}{\rho}\frac{\mathrm{d}\rho}{\mathrm{d}z} = 0 \,, \tag{141}$$

$$\frac{1}{w}\frac{\mathrm{d}w}{\mathrm{d}z} + \frac{1}{\kappa_0 M^2}\frac{1}{p}\frac{\mathrm{d}p}{\mathrm{d}z} = 0 \,, \tag{142}$$

$$\frac{c^2}{\kappa_0 - 1} + \frac{w^2}{2} = \frac{c_0^2}{\kappa_0 - 1} = const \,, \tag{143}$$

$$\frac{1}{p}\frac{\mathrm{d}p}{\mathrm{d}z} - \kappa_0 \frac{1}{\rho}\frac{\mathrm{d}\rho}{\mathrm{d}z} = 0 \,, \tag{144}$$

where $\dot{m}{=}\rho w A$ is the fluid's mass flow, (141)-(143) correspond to the quasi-one-dimensional Euler equations in the same order and (144) to the isentropic equation, [3].

Combining (141), (142) and (144) yields

$$\frac{1}{A}\frac{\mathrm{d}A}{\mathrm{d}z} = \left(M^2 - 1\right)\frac{1}{w}\frac{\mathrm{d}w}{\mathrm{d}z} \,. \tag{145}$$

(145) suggests that inside a bottle with a decreasing cross section ($\mathrm{d}A/\mathrm{d}x < 0$) the fluid should accelerate and the gas should reach $M = 1$ at the bottle opening because here its polynomial shape is defined as $\mathrm{d}A/\mathrm{d}z(z{=}L) = 0$.

Furthermore, (143) can be transformed to result in the non-dimensional relation between the Mach number and the fluid's velocity, when using the one-dimensional form of (140).

$$M(w) = \frac{1}{\sqrt{\frac{1}{w^2} - \frac{\kappa_0 - 1}{2}}} \tag{146}$$

Figure 19 confirms that for both points in time the fluid accelerates and reaches its maximum Mach number after the bottle opening. However, while for the first point in time M reaches 1 at around $z = 5.24\,\text{cm}$, which is just $0.02\,\text{cm}$ away from the end of the bottle, at the later time it reaches its critical value sooner at around $4.76\,\text{cm}$. This suggests that the evolution through time increases the discrepancy between the simulation and the proposed system and therefore shows that the assumptions, especially the steady flow condition, are unjustified.

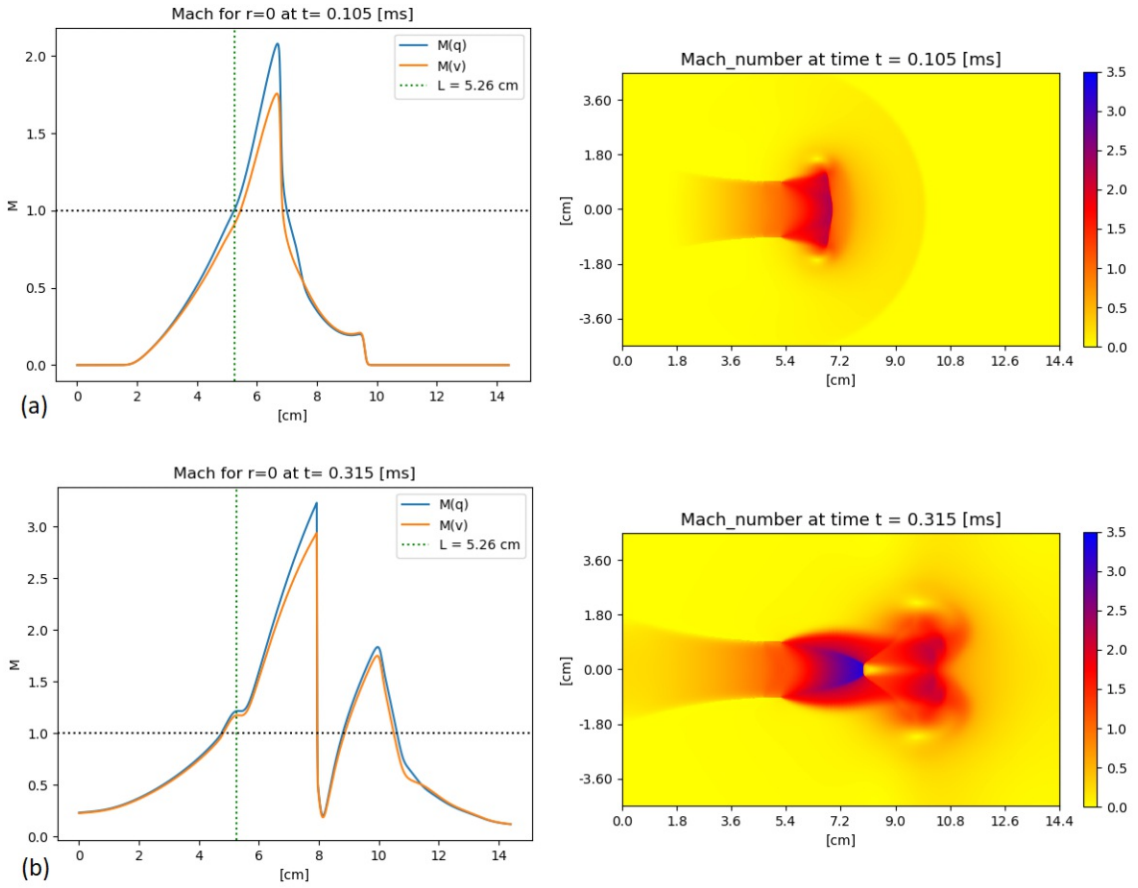

Figure 19: Mach number distribution $M(r, z, t)$ (right) and $M(r = 0, z, t)$ (left) at time $t = 0.105\,\text{ms}$ (top) and $t = 0.315\,\text{ms}$ (bottom) for a total cell number of 800 and $T_r = 20\,°\text{C}$. $M(q^i)$ is calculated with (140) and $M(v)$ with (146).

In spite of the previous fact, the left side of figure 19 shows that (146) can accurately calculate at least the acceleration inside the bottle and the behavior near the discontinuity, but fails at the given maxima. Note, that the similarities between the Mach number and the temperature plots, especially for $t{=}0.315\,\mathrm{ms}$, arise because the speed of sound in its non-dimensional form, which is included in the definition of M, is equal to $\sqrt{T}$.

5.2 Fluid-Cork-Interaction

The characteristics of a flow can vary greatly when a solid object is inserted into the domain. An additional challenge is the description of a moving body propelled by the fluid around it. In this thesis the impermeable object is a cork with a density of $\rho_C = 240\,\mathrm{kg/m^3}$, [24], and a volume already determined by (120). The cork's movement must be updated before each time step and therefore again the *solver.before_step* instance is used, but this time calling the function *update_aux(solver,state)*. As the name suggests, the given function updates the auxiliary fields in a way, that $aux(0,:,:)$ is used for the moving boundary conditions and $aux(1,:,:)$ stores the cork's velocity v_w. After the simulation *plot_mov(t,s,v,a,Dist,Dist2)* provides the plots of the desired quantities.

The algorithm inside *update_aux* starts by calculating the forces pushing onto the left and right wall of the cork. Therefore the pressure of the neighboring cells is determined by transforming (12). The forces are then evaluated by discretizing the following surface integral

$$F(R) = \int_0^{2\pi} \mathrm{d}\phi \int_0^{R} \mathrm{d}r\, r\, p(r)\,, \quad \rightarrow \quad F(N_C) = 2\pi\,\Delta r \sum_{i=int(N/2)}^{N_C} r_i\; p(r_i)\,, \tag{147}$$

where R is the cork's radius, N_C is the biggest cell index still occupied by the cork, Δr the grid distance and r_i the cell center all in the r-direction. The cork's non-dimensional acceleration is calculated by dividing the total force acting on the cork by its mass.

$$a_C = \frac{F_{tot}}{m_C} = \frac{F(N_l) - F(N_r)}{\kappa_0\,\rho_C\,V_C} \tag{148}$$

N_l and N_r are the corresponding uppermost indices for the left and right wall respectively and κ_0 is added due to dimensional analysis.

The cork's velocity v_C and position s_C of the left wall are updated by the transformation of the truncated Taylor series expansion with varying time intervals ($\Delta t \neq const$), [25],

$$\begin{aligned} s_C(t_{n-1}) &= s_C(t_n) - \Delta t_{n-1}\, v_C(t_n) + \frac{\Delta t_{n-1}^2}{2}\, a_C(t_n) + \mathcal{O}(\Delta t_{n-1}^3)\,, \\ s_C(t_{n+1}) &= s_C(t_n) + \Delta t_n\, v_C(t_n) + \frac{\Delta t_n^2}{2}\, a_C(t_n) + \mathcal{O}(\Delta t_n^3)\,, \end{aligned} \tag{149}$$

$$
\begin{aligned}
v_C(t_{n-1}) &= v_C(t_n) - \Delta t_{n-1}\, a_C(t_n) + \frac{\Delta t_{n-1}^2}{2}\, \dot{a}_C(t_n) + \mathcal{O}(\Delta t_{n-1}^3)\,, \\
v_C(t_{n+1}) &= v_C(t_n) + \Delta t_n\, a_C(t_n) + \frac{\Delta t_n^2}{2}\, \dot{a}_C(t_n) + \mathcal{O}(\Delta t_n^3)\,,
\end{aligned}
\tag{150}
$$

where $\dot{a}_C$ denotes the first derivative of a_C.

Multiplying the first equation in (149) with Δt_n, the second one with Δt_{n-1} and finally adding both up, yields

$$
s_C(t_{n+1}) = -s_C(t_{n-1})\,\alpha_n + s_C(t_n)\,(1+\alpha_n) + \frac{a_C(t_n)}{2}\,\Delta t_n(\Delta t_n + \Delta t_{n-1}) + \mathcal{O}(\Delta t^3)\,, \tag{151}
$$

where $\alpha_n = \Delta t_n / \Delta t_{n-1}$.

Applying the same procedure for (150), but this time multiplying the time intervals squared and subtracting the first from the second equation, results in

$$
v_C(t_{n+1}) = v_C(t_{n-1})\,\alpha_n^2 + v_C(t_n)\,(1-\alpha_n^2) + a_C(t_n)\,\Delta t_n(1+\alpha_n) + \mathcal{O}(\Delta t^3)\,. \tag{152}
$$

The thesis' program stores the time steps, s_C, v_C and a_C inside the corresponding arrays t, s, v, a, to which a new entry is appended every time the cork's values are updated. According to (151) and (152), t, s, v must have two initial entries, so that these equations can even be used. On the other hand, a does not have this requirement, therefore whenever $a_C(t_n)$ is calculated, all other values are updated for t_{n+1}.

While the cork's movement is stored before each time step, the auxiliary field $aux(0,:,:)$ is only changed, whenever the index of the left wall, calculated with $Ind\_x(s_C(t_{n+1}))$, increases by 1. If this is the case, the cells previously occupied by the cork are filled with the same values as the neighboring cells to avoid *None* values in areas, where the fluid is now present. Finally, the first auxiliary field is set by the same procedure as in *aux_init()*, but this time only in the region of the cork.

Note, that $aux(1,:,:)$, which stores the cork's velocity, is also updated before each time step and, unfortunately, must be assigned to every cell in the domain, although v_C does not vary in space. This unnecessary computation must be done because the *normal solver*, which is a one-dimensional solver, works with a reduced auxiliary array $aux(1,:) = aux(1,:,j)$, where j is the constant index of the dimension, which is currently not scanned through. Because j cannot be accessed by the solver and changes for every new scanning process, the whole domain must be filled with the cork's velocity and not only one cell, which could then be addressed.

In the following figures the effects of a moving cork interacting with the surrounding fluid are depicted as a density $\rho(r,z,t)$, pressure $p(r,z,t)$, Mach number $M(r,z,t)$ and temperature $T(r,z,t)$ plot at some instances of time.

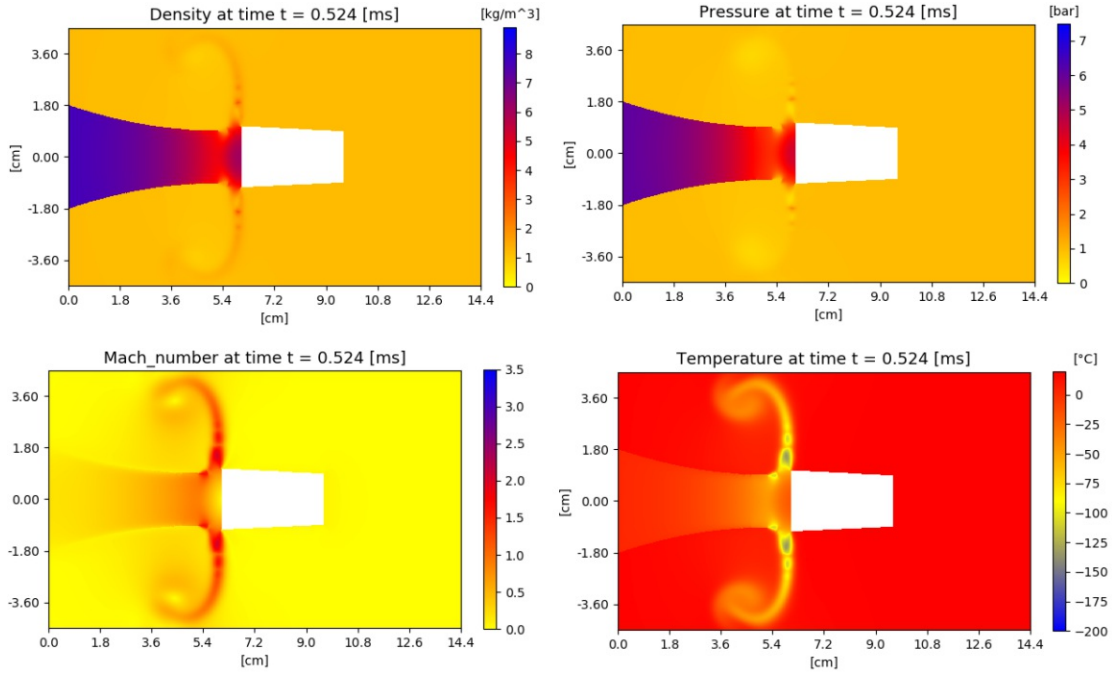

Figure 20: Density $\rho(r, z, t)$, pressure $p(r, z, t)$, Mach number $M(r, z, t)$ and temperature $T(r, z, t)$ at $t = 0.524\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

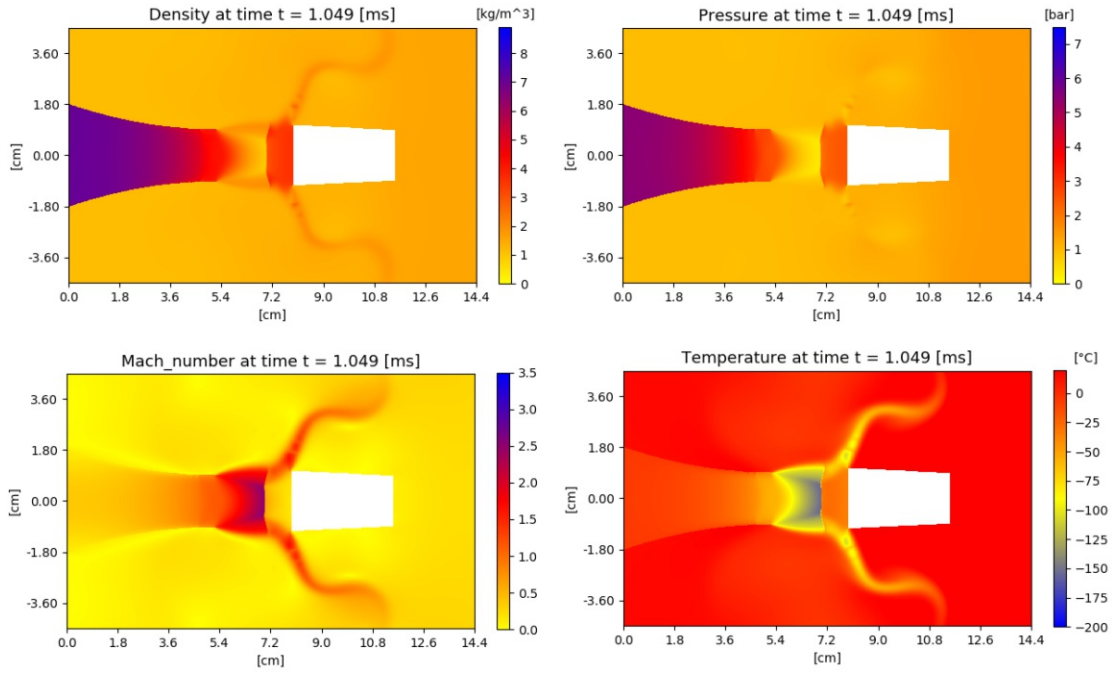

Figure 21: Density $\rho(r, z, t)$, pressure $p(r, z, t)$, Mach number $M(r, z, t)$ and temperature $T(r, z, t)$ at $t = 1.049\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

A number of different effects can be observed in figures 20 and 21. The main difference between the density and the pressure is that while the latter is only considerably higher in regions near the cork, besides the bottle's inside, high density also occurs along the main stream with the highest velocities. Along these lines Bernoulli's principle, [26], is sufficiently accurate to describe the behavior of decreasing pressure at least for some points. This decrease can also be observed, when depicting the temperature of the main stream. However, in this case there additionally exist regions of temperature minima, which can be nearly 100 °C lower than other points on these lines (see figure 22(b)).

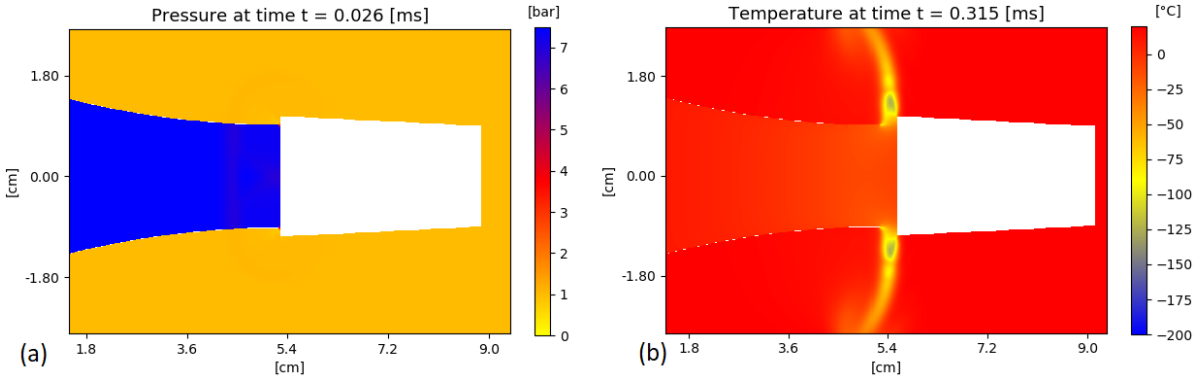

Figure 22: (a) An internally reflected pressure wave $p(r, z, t)$ at $t = 0.026\,\text{ms}$
(b) The temperature $T(r, z, t)$ at $t = 0.315\,\text{ms}$
for a total cell number of 800 and $T_r = 20\,°\text{C}$.

As mentioned before, the Mach number and the temperature plots are also very similar. Therefore it is no surprise that the region behind the cork and after the shock wave visualizes the rapid decrease in velocity due to the given boundary conditions, as well as the rapid increase in temperature.

Another effect can be found by observing the curvature of the main stream. In figure 20 the speed of the cork is still very small compared to the fluid's velocity. Therefore most of the gas is reflected by the cork's left wall and the stream is curved backwards. However, in figure 21 the increased speed and distance between the bottle and the cork allow the majority of the fluid to flow around the object, hence a curvature in the other direction occurs. The previously mentioned reflection can already be observed in the first few microseconds after the cork popping as a pressure wave (see figure 22(a)).

A more expressive analysis of the cork's movement is performed by depicting its acceleration, velocity and the position of the left wall as a function of time.

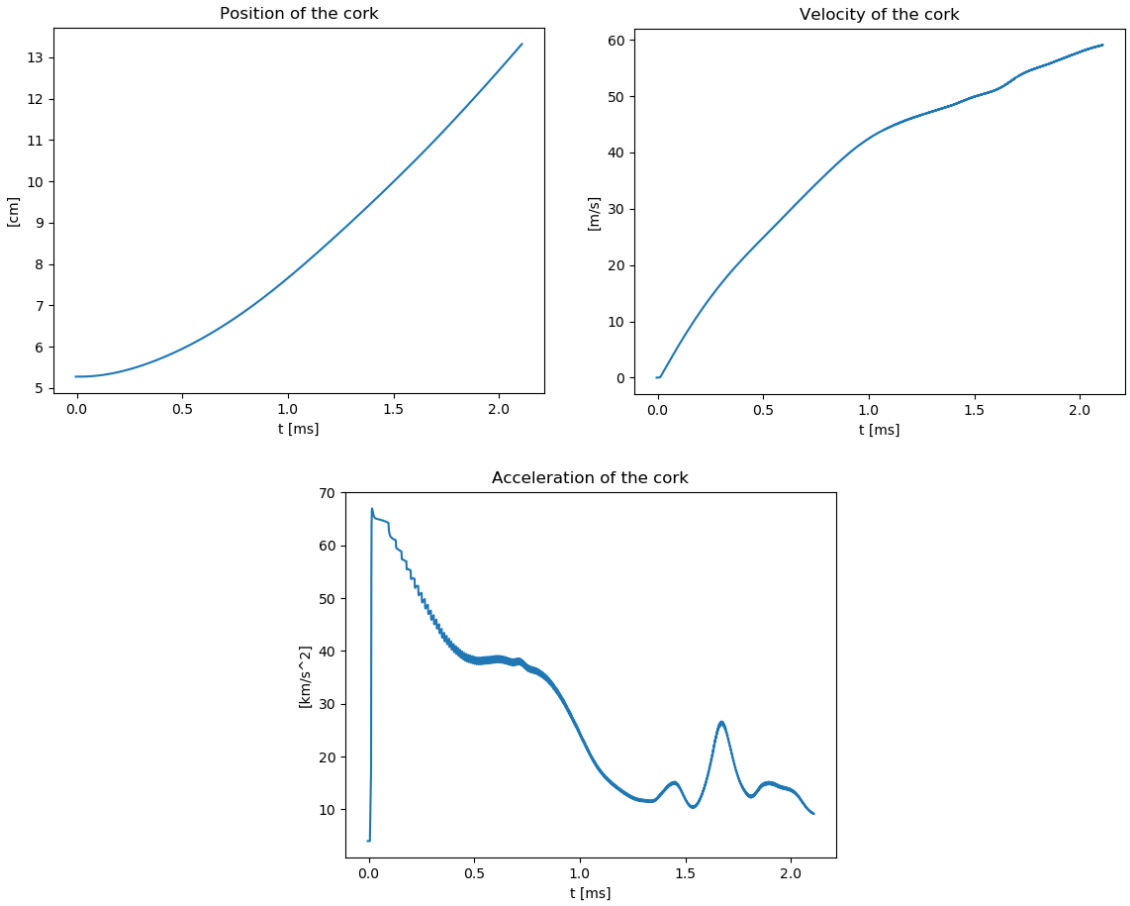

Figure 23: The cork's position of the left wall s_C, velocity v_C and acceleration a_C as a function of time for a total cell number of 800 and $T_r = 20\,°\mathrm{C}$, displaying the first 2 ms.

While s_C and v_C are continuous due to the behavior of the Taylor series expansion, a_C still possesses the discrete nature of its derivation in (147). This fact is most prominent at the start of the simulation, where the cork's indices remain unchanged for a longer period of time, due to the cork's small speed, than at later points in time.

5.2.1 Distance between the Shock Wave and the Cork

To further understand the behavior of the emerging shock wave, the distances between the bottle opening and the shock discontinuity and between the cork and the shock are evaluated as a function of time and compared with a series of contour plots, considering the pressure and Mach number at specific points in time. The position of the shock wave is determined by calculating the Mach number along $r = 0$ and then finding the global maximum between the bottle and cork. Whenever this maximum is reached, the corresponding cell's index is used to calculate the desired position.

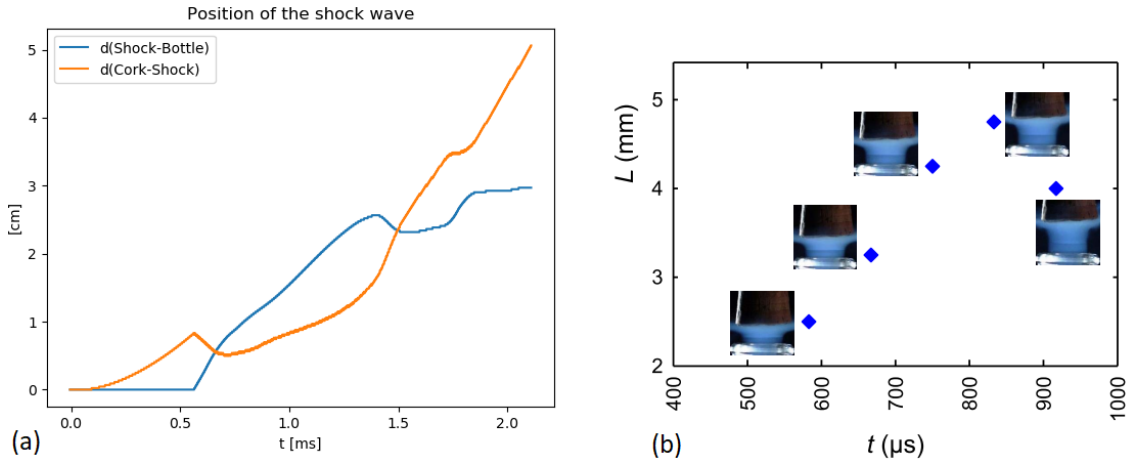

Figure 24: (a) The distance between the bottle opening and the shock (blue) and between the cork and the discontinuity (orange) for a total cell number of 800 and $T_r = 20\,°\text{C}$, displaying the first 2 ms.
(b) The corresponding experiment from [1] for the blue graph.

While the cork is already moving, the shock wave, as [1] suggests, only starts to appear after half a millisecond. Without a discontinuity being present, the shock numerically appears to be at the same position as the bottle opening and therefore the orange graph in figure 24 keeps increasing, while the blue one stays zero until 0.56 ms. The shock wave's appearance is visualized in figure 25.

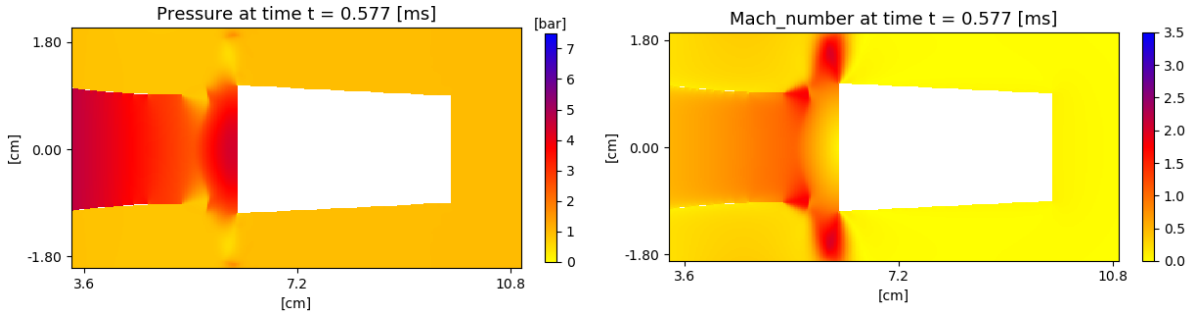

Figure 25: Close-up view of the pressure and Mach number at $t = 0.577\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

A region of increased pressure and decreased velocity is then formed, which temporarily becomes sharper and thinner, hence the distance between the shock and the cork decreases.

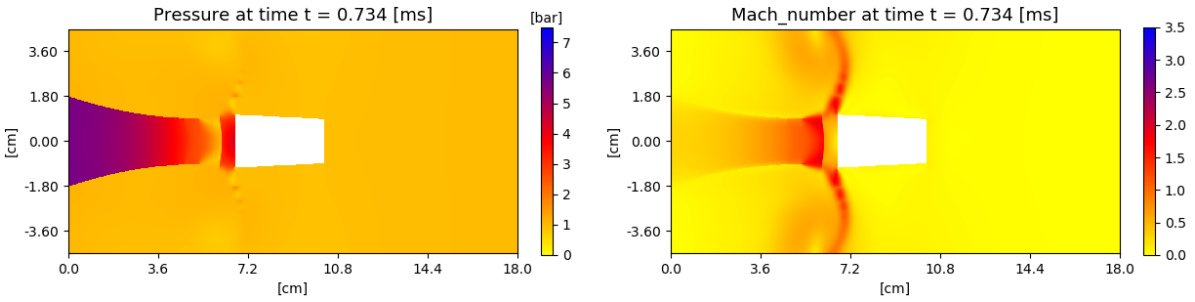

Figure 26: Contour plot of the pressure and Mach number at $t = 0.734\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

This behavior changes after the one millisecond mark. This is the time, when the Mach disk disappears in [1], leading to a widening of the pressure wave and a gradual increase in both distances.

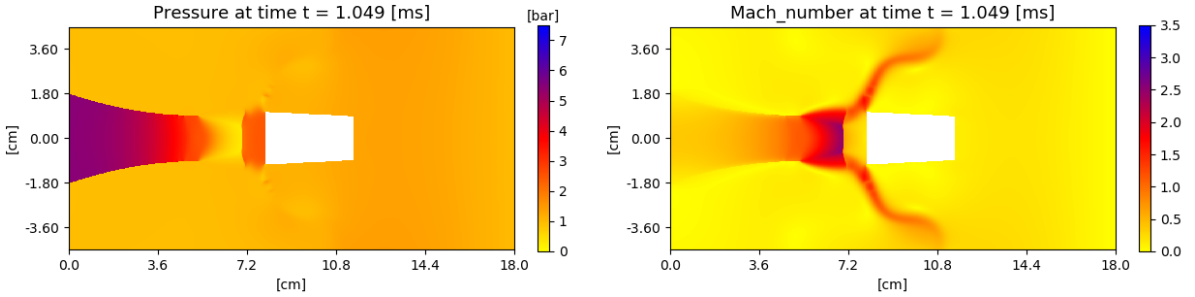

Figure 27: Contour plot of the pressure and Mach number at $t = 1.049\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

At approximately $t = 1.35\,\text{ms}$ the region behind the cork starts to disintegrate into two high pressure areas, where the first one seems to remain stationary for a short period of time and the second layer continues to move with the cork. Because the sudden decrease in the Mach number always occurs behind the first pressure wave, the stationary behavior of the first area explains the local maximum at $t = 1.4\,\text{ms}$ for the blue graph in figure 24.

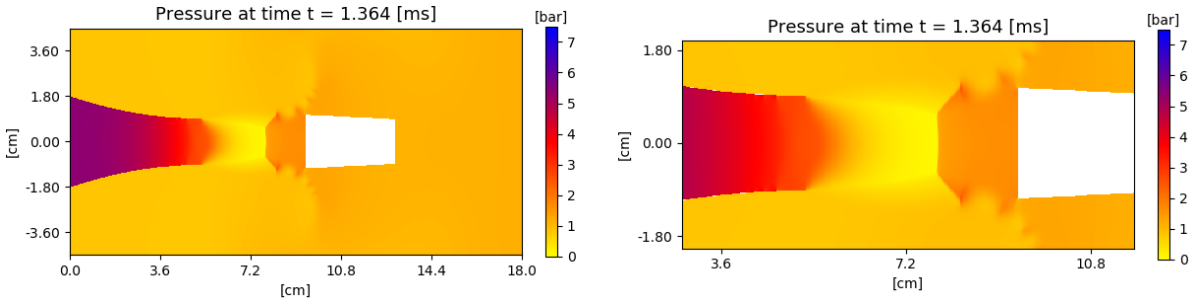

Figure 28: Contour plot and close-up view of the pressure at $t = 1.364\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

After the local maximum, the first layer even starts to retract, forming a Gaussian-shaped area. Around $t = 1.5\,\text{ms}$ the distance between the cork and the shock surpasses the distance between the wave and the bottle, reaching a local minimum for the blue graph shortly after. This retraction behavior also occurs in [1], but at an earlier time.

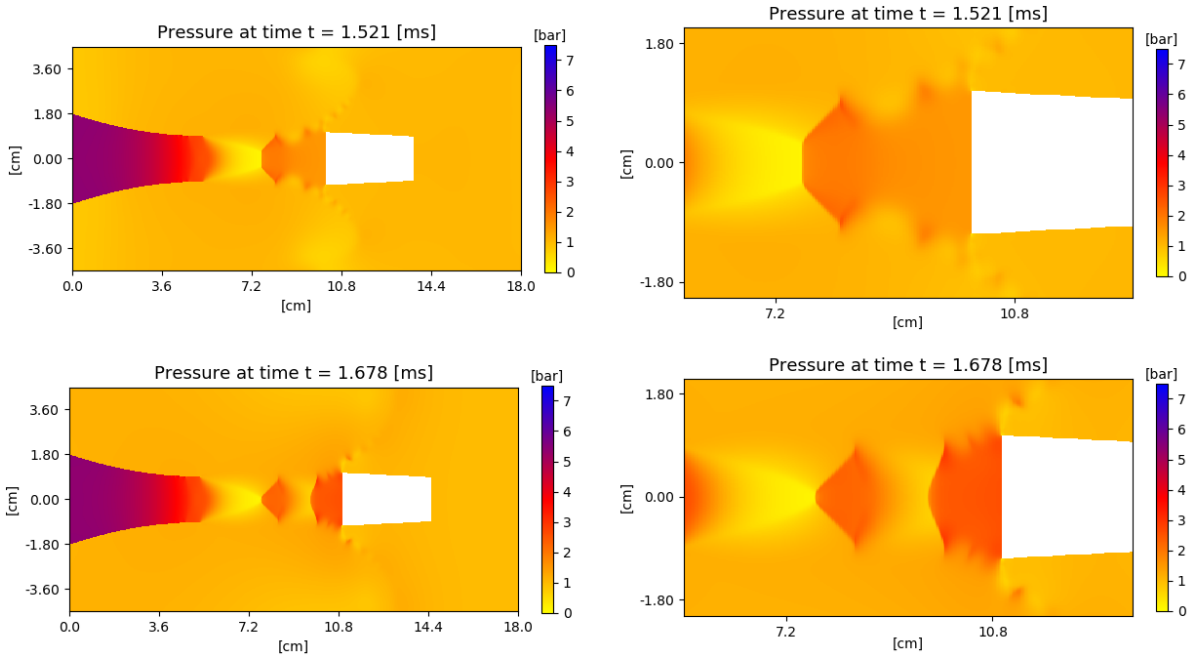

Figure 29: Contour plot and close-up view of the pressure at $t = 1.521\,\text{ms}$ and $t = 1.678\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

The mentioned Gaussian shape starts to expand in the r-direction and rapidly increases its velocity in the z-direction, which explains the blue graph's jump between $t = 1.7\,\text{ms}$ and $t = 1.85\,\text{ms}$. Moreover, the second layer widens one more time at $t = 2\,\text{ms}$ up to the point, where it disintegrates again into a third region. This process appears to repeat for every millisecond, though a longer time interval would be needed to prove this claim.

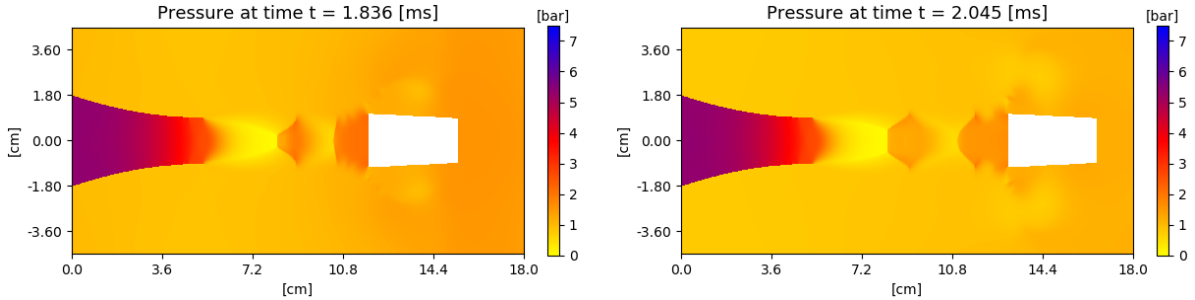

Figure 30: Contour plot of the pressure at $t = 1.836\,\text{ms}$ and $t = 2.045\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

A similar behavior can be observed for the Mach number, in this case indicating regions of low velocities.

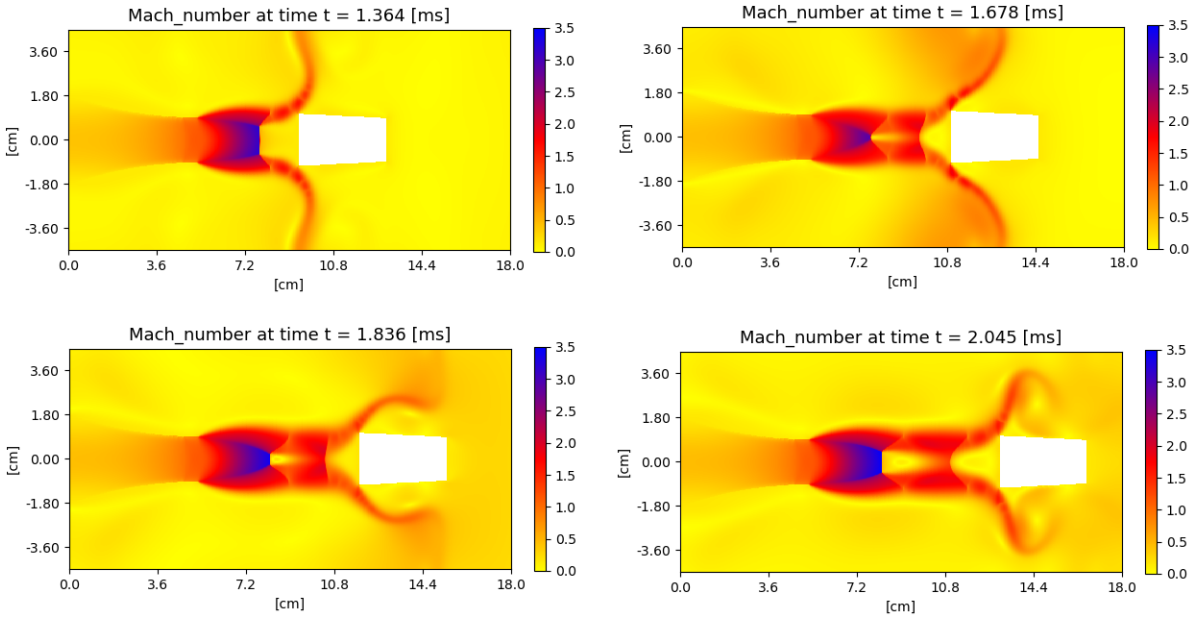

Figure 31: Contour plot of the Mach number at $t = 1.364\,\text{ms}$, $t = 1.678\,\text{ms}$, $t = 1.836\,\text{ms}$ and $t = 2.045\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

Finally, a visual comparison of the experiment in [1] and the results of the simulation can be achieved, as depicted in figures 32 and 33.

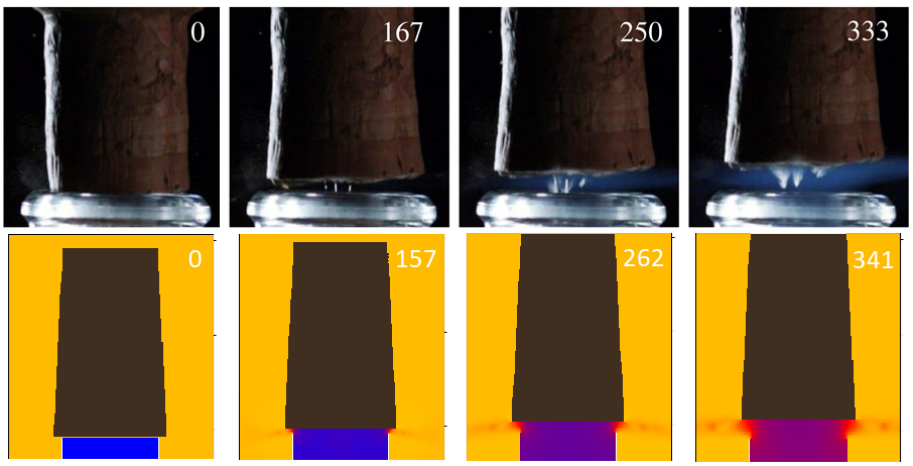

Figure 32: Comparison of the experiment performed in [1] and the present simulation for the density between $t = 0\,\text{ms}$ and $t = 0.350\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

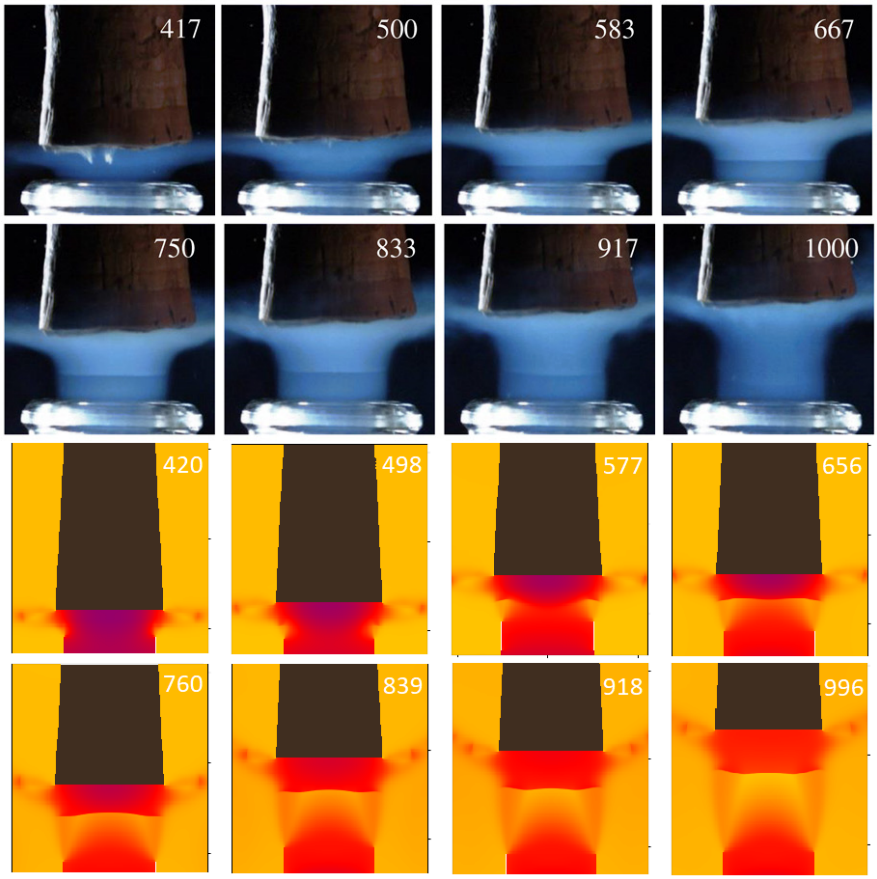

Figure 33: Comparison of the experiment performed in [1] and the present simulation for the density between $t = 0.4\,\text{ms}$ and $t = 1.0\,\text{ms}$ for a total cell number of 800 and $T_r = 20\,°\text{C}$.

5.3 Unresolved Issues and Further Improvements

Although the present thesis provides a complete numerical simulation regarding the proposed system in [1], there are numerous further improvements to be made and unresolved issues worth discussing.

The biggest adaptation would be changing the used solver type. Instead of the *Roe solver*, a more advanced *n-wave-solver* could be used to include all possible waves for a system of n equations. This would allow to generalize the given Euler equations to be able to describe a number of different gases and liquids, which could all interact with each other, exchanging certain properties and quantities of the corresponding fluid. However, this system would dramatically increase the complexity of the necessary equations and therefore the computational power for simulating a certain number of time steps.

To counteract these requirements, the overall grid could be modified to be *non-uniform*, so that the size of the grid cells is bigger in areas, where the fluid's quantities are similar, compared with their cell neighbors, and smaller in regions, where a lot of movement is happening or a detailed depiction is desired. This would generally decrease the required computational effort, with the expense of a more complex numerical algorithm. A simpler way would be to simulate just the upper half of the bottle and simply mirror the solution for the bottom half. This would be a logical step due to the given axial symmetry. For a pure two-dimensional Cartesian system this procedure would not cause any issues at all, but for the given system the present source term demands its own boundary conditions for $r = 0$, if this value represents the bottom edge of the domain. This requirement is not trivial to accomplish and therefore the longer duration of computation is tolerated instead.

In this thesis the cork starts at rest with one cell of fluid between itself and the bottle opening. This is necessary to fulfill the numerical implementation of the boundary conditions, where the ghost cells of the bottle opening must each have at least one fluid cell on their right side and for the cork's cells on their left side. A more physical initial placement of the cork would be to start inside the bottle. This in turn would require a slim film of fluid between both objects and would inevitably cause some of the fluid to exit the bottle before the cork has even started to move. Additionally, the film would cause friction, which should also be considered.

While convergence, stability and accuracy were at least partly determined in chapter 5 and briefly discussed regarding Roe's proposed solver, the latter can be considered by high-resolution methods. Unfortunately, it was not possible to increase the system's accuracy by applying a standard flux-limiter. By modifying the core instances of the CLAWPACK library, which are responsible for this implementation, a limiter could achieve an anti-diffusive flux that could sharpen up Godunov's method, [4]. Stability also depends on the CFL-condition, where different values for *cfl_desired* lead to distinguishable behaviors of the fluid, especially for fine details.

Finally, there are still unresolved issues, which should be addressed in a following work or paper concerning similar systems. While the acceleration of the cork in figure 23 behaves as expected for the first millisecond, after 1.5 ms a_C starts to oscillate. The reason for this behavior could be that after the separation of the two high pressure areas, the first one moves towards the bottle, hence possessing a negative velocity. Due to the conservation of momentum, the pressure wave could cause a push onto the cork in the positive z-direction and therefore increases its acceleration temporarily. Another observation, which was not explained in the previous section, is the retraction happening at a later point in time in the simulation than [1] could determine. While the cork at the start of the simulation is at rest, in the experiment the cork already has an initial velocity when exiting the bottle opening. This fact could explain the earlier retraction. However, both claims must be studied more thoroughly to determine their validity.

6 Conclusion

The simulation of a sudden gas discharge through a bottle opening is performed by assuming a compressible Euler fluid in cylindrical coordinates undergoing an adiabatic expansion. The numerical solution for each time step is found via Godunov's method, receiving the waves and corresponding fluctuations from a modified HLLE-solver. Due to axial symmetry, the system's domain can be reduced to a two-dimensional cross-section, where a source term denotes the mixed-dimensional terms and which is incorporated inside the particular equation and solved by a second order Runge-Kutta method. While the boundary conditions at the computational domains' edges consider an undisturbed outflow, the bottle's and cork's walls cause a reflection of the quantities' values in accordance with the no-penetration condition. The latter effect is numerically implemented inside the normal solver and executed with the help of auxiliary fields.

Analyzing the unobstructed flowing fluid in section 5 validates the assumption of an isentropic change of state at the bottle opening and disproves that the flow could be considered steady. Furthermore, the Mach disk can be seen as a discontinuity inside the distribution of the local Mach number and temperature, when illustrating the quantities' values along the z-axis for a given point in time. Finally, the interaction between the fluid and the incorporated cork is regarded as a propelled motion of the solid object due to the pressure difference between its front and back end wall. While the cork's acceleration consists of discrete jumps, especially at the beginning due to the finite number of grid cells, its velocity and position of the left wall are calculated with the truncated Taylor series expansion with varying time steps and are therefore regarded as being continuous. Finally, the distances between the bottle and the shock and between the cork and the discontinuity are investigated. Hereby the appearance of the shock wave after half a millisecond, its disintegration and the retraction behavior of the newly created second wave can all be observed, when viewing the pressure and Mach number distributions at certain points in time, as well as the time evolution of the corresponding distances.

The vision of this thesis is that a following paper or thesis could use the acquired knowledge and the provided Python and Fortran programs to further investigate the behavior of the Mach disk and especially the phase transition of the CO_2 gas after its emission. First of all, it should be validated if the transition really only occurs after the bottle opening and not inside of it for every bottle geometry and initial condition. If this is not the case, dry ice could damage the inner walls of the bottle, which would be devastating considering a similar system such as a thrust nozzle. The second question would then be, how to design the bottle opening's geometry to minimize the interaction between the wall and the dry ice particles for a given initial condition. Therefore the applied equations must certainly consider inner friction and a multi-phase mixture.

7 Appendix

This section includes the table of all symbols and quantities used in this thesis, and three code listings which are necessary for the numerical computation. *Constants.py* stores all constants either in their dimensional or non-dimensional form. *Custom Euler Solver.f90* includes the Roe solver and the algorithm to fulfill the internal no-penetration condition. *Main.py* is the main file including all necessary functions and PyClaw's classes to run the simulation, as well as the instances which are responsible for plotting the resulting data. Note, that a few quantities possess different names, such as κ_0 being called *gamma_0*, $w \to u$ and $u \to v$, so that u describes the velocity in the z-direction and v the velocity in the r-direction. Because u is always considered first, the entries 2 and 3 of the quantities in (44) are swapped and of all the other vectors and matrices in later equations too, which are derived from the first ones. This fact explains the different entries of the source term.

Constants.py

```
import numpy as np

#Gas constant
R = 8.314

#Inital values of the environment
p0 = 1.013
rho_0 = 1.204
gamma_0 = 7./5.
T0 = 293.15
mu_0 = 1.827
lambda_0 = 0.0262

#Diameter of the bottle opening
d0 = 1.8

#Derived quantities
c0 = np.sqrt(10**5*gamma_0*p0/rho_0)
E0 = 10**5*p0/(gamma_0-1.)
tau_0 = d0/c0 * 0.01 #in seconds

#Gas volume inside the bottle
Vr = 25.

#Inital values inside the bottle
pr = 7.5 #10.2# for the second bottle
Tr = 293.15 #303.15# for the second bottle

#Density of the cork
rho_Cr = 240.

#Dimensionless quantities
p = pr/p0
T = Tr/T0
rho = p/T
E = p

#Dimensionless length of the bottle
L = 15./(7.*np.pi) * Vr/d0**3

#Dimensionless density and volume of the cork
rho_C = rho_Cr/rho_0
VC = 91.*np.pi/150.

#Heat capacity for constant pressure
cp_0 = gamma_0*p0*10**5/((gamma_0-1.)*rho_0*T0)
```

Symbol	Description
Chapter 2:	
$d_0 = 1.8$ cm, [1]	diameter of the bottle opening
$L = 5.26$ cm	length of the gaseous region inside the bottle
$p_0 = 1.013$ bar, [20]	ambient pressure
$T_0 = 20$ °C	ambient temperature
$\rho_0 = 1.204$ kg/m^3, [21]	ambient mass density of air
$\kappa_0 = 7/5$, [22]	heat capacity ratio of air
$T_r = 20, 30$ °C, [1]	initial temperature of the internal gas
$V_r = 25$ ml, [1]	initial volume of the internal gas
$p_r = 7.5$, 10.2 bar, [1]	initial pressure of the internal gas
$\rho_C = 240$ kg/m^3, [24]	mass density of the cork
Chapter 3:	
v^i	velocity
E	energy density
$\rho\, e$	internal energy density
$\mathcal{M}$	molar mass
$\rho\, h$	internal enthalpy density
f	degrees of freedom
$\mathcal{V}$	specific volume
s	specific entropy
$\rho\, H$	total enthalpy density
x_i	spatial dimensions
r^m	position vector
t	time
q^i	vector of unknown quantities
σ^{ij}	stress tensor
Q^i	heat flux
D^{ij}	strain rate tensor
∇^i	Nabla operator
δ^{ij}	Kronecker delta
Φ	dissipation function
f^{ij}, f_l^i	flux matrix and vector
Ψ^i	source term
A_l^{ij}	speed matrix
$\mathcal{B}$	vector basis
r, ϕ, z	cylindrical coordinates
$\hat{e}^i$	unit vector
u, v, w	velocity components
v_w^i	velocity of a solid wall
$\hat{n}^i, \hat{t}^i$	normal and tangent unit vectors of a body's surface

Symbol	Description
$R = 8.314$ J/(K mol), [27]	gas constant
$c_p, c_\mathcal{V}$	heat capacities for constant p and $\mathcal{V}$
$c_{p_0} = 1004.53$ J/(kg K)	ambient heat capacity
$c_0 = 343.21$ m/s	ambient speed of sound
$\tau_0 = 52.45\,\mu$s	reference-time
$\bar{\mu}, \mu$	volume and dynamic viscosity
$\mu_0 = 1.827 \cdot 10^{-5}$ Pa s, [28]	ambient dynamic viscosity
$g_0 = 9.81$ m/s^2, [29]	gravitational acceleration on earth's surface
$\lambda_0 = 0.0262$ W/(m K), [30]	ambient thermal conductivity
$Sr = 1$	Strouhal-number
$Eu = 5/7$	Euler-number
$Fr = 817$	Froude-number
$Re = 4.07 \cdot 10^5$	Reynolds-number
$Ec = 2/5$	Eckert-number
$Pr = 0.700$	Prandtl-number
Chapter 4:	
s	wave speed
$\hat{n}^m$	wave direction
λ_p	eigenvalues of A_l^{ij}
r_p^i	right eigenvectors of A_l^{ij}
w_p	characteristic variables
T^{ij}	transformation matrix of A_l^{ij}
Λ^{ij}	diagonal matrix of A_l^{ij}
l_p^i	left eigenvectors of A_l^{ij}
α_p	jump constant
W_p^i	p-th wave
q_m^i	middle state between two discontinuities
m	cell index
ΔV	volume of a grid cell
$r_m^i, r_{m-1/2}^i$	center and left interface of the m-th cell
f_{in}^i, f_{out}^i	total fluxes into and out of a cell
Δt	time step
F^i	approximation to the average flux over Δt
$\mathcal{F}^i$	numerical flux vector
$\tilde{q}^i$	piecewise polynomial function
$\mathcal{W}_p^i$	numerical p-th wave

Symbol	Description
Δx_m	length of the m-th cell
$A^{ij}_- \Delta Q^j, A^{ij}_+ \Delta Q^j$	left and right-traveling wave fluctuations
$\hat{A}^{ij}$	approximation of A^{ij} over two adjacent cells
$\hat{Q}^i$	some average of Q^i
z^i	Roe's parameter vector
Z^i	average value of z^i inside a cell
$\hat{A}^{ij}, \hat{B}^{ij}, \hat{C}^{ij}$	linearization matrices
$\bar{Z}^i$	discretized mean vector of z^i
$\hat{u}, \hat{w}, \hat{H}$	Roe averages
L	non-dimensional length of the bottle
$f_B(x), f_C(x)$	curvature function of the bottle's or cork's surface
x_{st}, s_{end}	start and end point of an object in the x-direction
V_C	volume of the cork
N	total number of cell interfaces
i_x	index of x_{st}
j_{bot}, j_{top}	arrays storing the indices of the object's surface
Q^i_h	homogeneous solution of Q^i
$b_j, a_{j\,l}$	coefficients from the Butcher table
M	last internal index of an object
N_G	number of ghost cells
$\mathcal{D}$	domain of dependence
r	fixed step size ratio
ν	Courant number
$\tilde{F}^i$	fluxes based on the waves from the Riemann solution
$\mathcal{F}^i_L, \mathcal{F}^i_H$	low and high order flux formula
Chapter 5:	
M	local Mach number
$\dot{m}$	mass flow of the fluid
A	cross section of the bottle
F	force acting on one side of the cork
N_C	biggest cell index still occupied by the cork
m_C	mass of the cork
N_l, N_r	N_C on the left and right surface of the cork
$\Delta r, r_i$	grid distance and position of the cell center in the y-direction
s_C, v_C, a_C	position of the left wall, velocity and acceleration of the cork
s, v, a	numerical arrays of s_C, v_C, a_C
j	constant index of the dimension which is not scanned through

Custom Euler Solver.f90

```
! ================================================================
subroutine rpn2(ixy,maxm,meqn,mwaves,maux,mbc,mx,ql,qr,auxl,auxr,wave,s,amdq,apdq)
! ================================================================

! HLLE solver for the Euler equations.
! This solver takes into account internal reflecting boundaries within the domain.
! The aux array(1) contains the geometry: where it is equal to zero, a solid object exists.
! The fluid domain is the region, where aux is equal to one.

! waves: 2
! equations: 4

! Conserved quantities:
!       1 density
!       2 x-momentum
!       3 y-momentum
!       4 energy

    implicit none

    integer, intent(in) :: ixy, maxm, meqn, mwaves, maux, mbc, mx
    double precision, dimension(meqn,1-mbc:maxm+mbc), intent(inout) :: ql, qr
    double precision, dimension(maux,1-mbc:maxm+mbc), intent(in) :: auxl, auxr
    double precision, dimension(meqn, mwaves, 1-mbc:maxm+mbc), intent(out) :: wave
    double precision, dimension(mwaves, 1-mbc:maxm+mbc), intent(out) :: s
    double precision, dimension(meqn, 1-mbc:maxm+mbc), intent(out) :: amdq, apdq

    double precision :: rho_l,rho_r,u_l,u_r,v_l,v_r,E_l,E_r
    double precision :: p_l,p_r,H_l,H_r,c_l,c_r
    double precision :: sqrho_l,sqrho_r,u_hat,v_hat,H_hat,c_hat
    double precision :: s1, s2,rho_m,rhou_m,rhov_m,E_m
    integer :: rho, mu, mv, E
    integer :: i, j, m, mw

    double precision :: gamma
    common /cparam/  gamma


!     # Set mu to point to the component of the system that corresponds
!     # to the momentum in the direction of this slice, mv to the orthogonal
!     # momentum:

    rho = 1
    E = 4
    if (ixy.eq.1) then
        mu = 2
        mv = 3
    else
        mu = 3
        mv = 2
    endif

    do i=2-mbc,mx+mbc

        ! First handle boundaries
        if( (auxl(1,i-1) .eq. 0.d0) .and. (auxl(1,i) .eq. 1.d0) ) then
            ! Reflecting boundary to the left
            do j=1,mbc
                qr(1,i-j) = ql(1,i+j-1)
                qr(mu,i-j) = -ql(mu,i+j-1)
                qr(mv,i-j) = ql(mv,i+j-1)
                qr(4,i-j) = ql(4,i+j-1)

                !Called if the cork's velocity is not zero
                if((auxr(2,2) .ne. 0d0) .and. (ixy.eq.1)) then
                    qr(mu,i-j) = qr(mu,i-j) + 2d0*auxl(2,2)*qr(1,i+j-1)
                end if
            end do

        else if( (auxl(1,i) .eq. 0.d0) .and. (auxl(1,i-1) .eq. 1.d0) ) then
            ! Reflecting boundary to the right
            do j=1,mbc
                ql(1,i+j-1) = qr(1,i-j)
                ql(mu,i+j-1) = -qr(mu,i-j)
                ql(mv,i+j-1) = qr(mv,i-j)
                ql(4,i+j-1) = qr(4,i-j)

                !Called if the cork's velocity is not zero
                if((auxr(2,2) .ne. 0d0) .and. (ixy.eq.1)) then
                    ql(mu,i+j-1) = ql(mu,i+j-1) + 2d0*auxl(2,2)*ql(1,i-j)
                end if
            end do
```

```
            end if


            !Density
            rho_l = qr(rho,i-1)
            rho_r = ql(rho,i  )
            !Velocity
            u_l = qr(mu,i-1) / rho_l
            u_r = ql(mu,i  ) / rho_r
            v_l = qr(mv,i-1) / rho_l
            v_r = ql(mv,i  ) / rho_r
            !Energy
            E_l = qr(E,i-1)
            E_r = ql(E,i  )
            !Pressure
            p_l = E_l - gamma*(gamma-1d0)*rho_l*0.5d0*(u_l**2+v_l**2)
            p_r = E_r - gamma*(gamma-1d0)*rho_r*0.5d0*(u_r**2+v_r**2)

            !Enthalpy
            H_l = E_l/rho_l - (gamma-1d0)**2 *0.5d0* (u_l**2+v_l**2)
            H_r = E_r/rho_r - (gamma-1d0)**2 *0.5d0* (u_r**2+v_r**2)
            !Sound speed
            c_l = dsqrt(p_l/rho_l)
            c_r = dsqrt(p_r/rho_r)

            sqrho_l = dsqrt(rho_l)
            sqrho_r = dsqrt(rho_r)

            !Roe averages
            u_hat = (sqrho_l*u_l+sqrho_r*u_r) / (sqrho_l+sqrho_r)
            v_hat = (sqrho_l*v_l+sqrho_r*v_r) / (sqrho_l+sqrho_r)
            H_hat = (sqrho_l*H_l+sqrho_r*H_r) / (sqrho_l+sqrho_r)
            c_hat = dsqrt(H_hat-(gamma-1d0)*0.5d0*(u_hat**2+v_hat**2))

            ! Speeds of non-shear waves
            s1 = min(u_l - c_l, u_hat - c_hat)
            s2 = max(u_r + c_r, u_hat + c_hat)

            ! "Middle" state
            rho_m = (ql(mu,i) - qr(mu,i-1) - s2*ql(rho,i) + s1*qr(rho,i-1))/(s1-s2)
            rhou_m = (ql(rho,i)*u_r**2 - qr(rho,i-1)*u_l**2 + (p_r - p_l)/gamma - s2*ql(mu,i) + s1*qr(mu,i-1))/(s1-s2)
            rhov_m = (ql(mv,i)*u_r - qr(mv,i-1)*u_l - s2*ql(mv,i) + s1*qr(mv,i-1))/(s1-s2)
            E_m = ( u_r*(E_r+(gamma-1d0)*p_r) - u_l*(E_l+(gamma-1d0)*p_l) -s2*E_r + s1*E_l)/(s1-s2)

            !The first wave
            wave(rho,1,i) = rho_m - qr(rho,i-1)
            wave(mu,1,i) = rhou_m - qr(mu,i-1)
            wave(mv,1,i) = rhov_m - qr(mv,i-1)
            wave(E,1,i) = E_m - qr(E,i-1)
            !The speed of the first wave
            s(1,i) = s1

            !The second wave
            wave(rho,2,i) = ql(rho,i) - rho_m
            wave(mu,2,i) = ql(mu,i) - rhou_m
            wave(mv,2,i) = ql(mv,i) - rhov_m
            wave(E,2,i) = ql(E,i) - E_m
            !The speed of the second wave
            s(2,i) = s2
        end do


        do m=1,meqn
            do i=2-mbc, mx+mbc
                amdq(m,i) = 0.d0
                apdq(m,i) = 0.d0
                do mw=1,mwaves
                    if (s(mw,i) .lt. 0.d0) then
                        !The left fluctuations
                        amdq(m,i) = amdq(m,i) + s(mw,i)*wave(m,mw,i)
                    else
                        !The right fluctuations
                        apdq(m,i) = apdq(m,i) + s(mw,i)*wave(m,mw,i)
                    endif
                end do
            end do
        end do


    end subroutine rpn2
```

Main.py

```
import matplotlib.pyplot as plt
from clawpack import pyclaw
import numpy as np
from clawpack import riemann
import Constants as c

#Function of the Bottle
def f_B(x):
    return x**2/(2.*c.L**2) - x/c.L + 1.


#Function of the Cork
def f_C(x):
    return 0.6 - x/20.


#Evaluates the closest index for any point along the x-axis
def Ind_x(x_st,x_end):

    # ERROR Management
    if x_end < x_st + dx:
        raise ValueError("x_end must be equal or greater than x_st+dx")
    if x_end > x.upper:
        raise ValueError("x_end exceeded the domain")
    if x_st != x_st or x_end != x_end:
        raise ValueError("x_st = " + str(x_st) + " and x_end = " + str(x_end) + " both must be a number")

    #Returnes the corresponding index if the absolute distance is at a minimum
    def min(i_st,xp):
        l = 1000.
        for i in range(i_st,M):
            absl = abs(X[i,0]-xp)
            if absl >= l:
                return i - 1
            else:
                l = absl

    index_left = min(0,x_st)
    index_right = min(index_left,x_end)

    return index_left, index_right


#Evaluates the y-index above the function's value
#for all x-positions between xl and xr
def Ind_y(xl,xr,func):

    f = []

    #Filling in the function's values at all interfaces
    # between xl and xr, excluding the last one
    for i in range(xl,xr):
        f.append(func(X[i,0]-X[xl,0]))

    index_top = []

    #Stores the index if the y-position of the cell interface
    #is bigger than the function's value at a given x-value
    for i in range(len(f)):
        for j in range(int(N/2),N):
            if Y[0,j] >= f[i]:
                index_top.append(j)
                break

    index_top = np.array(index_top)
    index_bot = N-index_top-1

    return index_bot,index_top


#Determines the inital values of q
def init_values(value):

    #filling the domain with ones
    array = np.ones((Mm,Nm))

    #ix==0: for the bottle
    #else: for the cork
    def init(ix,j_bot, j_top):
        for i in range(len(j_top)):
            if ix == 0.:
                array[ix+i,j_bot[i]:j_top[i]] = value
```

```
                array[ix+i,j_bot[i]-tkn:j_bot[i]] = None
                array[ix+i,j_top[i]:j_top[i]+tkn] = None
            else:
                array[ix+i,j_bot[i]:j_top[i]] = None

    init(b_st,b_bot,b_top)

    #The cork is only considered for the
    #second set of plots
    if plot == 2:
        init(c_st,c_bot,c_top)

    return array


#Determines the initial values of the aux-field
def aux_init():

    # filling the aux-field with ones
    array = np.ones((Mm,Nm))

    #Same procedure as for the previous None values
    def init(ix,j_bot, j_top):
        for i in range(len(j_top)):
            if ix == 0.:
                array[ix+i,j_bot[i]-tkn:j_bot[i]] = 0
                array[ix+i,j_top[i]:j_top[i]+tkn] = 0
            else:
                array[ix+i,j_bot[i]:j_top[i]] = 0

    init(b_st,b_bot,b_top)

    #The cork is only considered for the
    #second set of plots
    if plot == 2:
        init(c_st,c_bot,c_top)

    return array


#Is responsible for the 2D heat-maps
def setplot(plotdata):

    from clawpack.visclaw import colormaps

    # Clear any old figures,axes,items data
    plotdata.clearfigures()

    def default_plot_settings(plotfigure, max):

        # Set up for axes in this figure:
        plotaxes = plotfigure.new_plotaxes()

        #Configures settings to which VisClaw has no direct access
        def afteraxes(current_data):
            from matplotlib import pyplot as plt

            #Creates the title
            name = plotfigure.name.split(" ")
            t = current_data.t*c.tau_0*1000
            plt.title(name[0]+" at time t = "+"{:.3f}".format(t)+" [ms]", fontsize=13)

            plt.xlabel("[cm]")
            plt.ylabel("[cm]")

            #Size of the colorbar and its label
            #0.032, 0.0235 and 0.029 were used for the fraction
            cb = plt.colorbar(fraction=0.0235)
            cb.set_label(label=name[1], labelpad=-18,y=1.07,rotation=0.)

            #Location of the axis' values
            #x-axis:
            locs, labels = plt.xticks()
            if float("{:.6f}".format(locs[-1])) != x.upper:
                locs = locs[:-1]
            A = []
            for i in range(len(locs)):
                A.append("{:.1f}".format(locs[i] * c.d0))
            plt.xticks(locs, A)

            #y-axis:
            locs, labels = plt.yticks()
            if float("{:.6f}".format(locs[-1])) != y.upper:
                locs = locs[1:-1]
            A = []
```

```
            for i in range(len(locs)):
                A.append("{:.2f}".format(locs[i]*c.d0))
            plt.yticks(locs,A)

        figno = plotfigure.figno

        plotaxes.afteraxes = afteraxes
        plotaxes.scaled = True # so aspect ratio is 1

        # Set up for item on these axes:
        plotitem = plotaxes.new_plotitem(plot_type='2d_pcolor')

        plotitem.pcolor_cmap = colormaps.yellow_red_blue
        plotitem.pcolor_cmin = 0.
        plotitem.pcolor_cmax = max

        #Is needed because the colorbar was already
        #implemented inside afteraxes
        plotitem.add_colorbar = False

        #Density plot
        if figno == 0:
            def Dens(current_data):
                rho = current_data.q[0, :, :]
                return rho * c.rho_0
            plotitem.plot_var = Dens

        #Pressure plot
        elif figno == 1:
            def Press(current_data):
                u = current_data.q[1, :, :] / current_data.q[0, :, :]
                v = current_data.q[2, :, :] / current_data.q[0, :, :]
                p = current_data.q[3,:,:]-c.gamma_0*(c.gamma_0-1.)*current_data.q[0,:,:]*0.5*(u**2+v**2)
                return p * c.p0
            plotitem.plot_var = Press

        #Mach number plot
        elif figno == 2:
            def abs_velocity(current_data):
                u = current_data.q[1, :, :] / current_data.q[0, :, :]
                v = current_data.q[2, :, :] / current_data.q[0, :, :]
                T = current_data.q[3,:,:]/current_data.q[0,:,:] - c.gamma_0*(c.gamma_0-1.)*0.5*(u**2+v**2)
                return np.sqrt((u ** 2 + v ** 2)/T)
            plotitem.plot_var = abs_velocity

        #Temperature plot
        else:
            def Temp(current_data):
                u = current_data.q[1, :, :] / current_data.q[0, :, :]
                v = current_data.q[2, :, :] / current_data.q[0, :, :]
                T = current_data.q[3,:,:]/current_data.q[0,:,:] - c.gamma_0*(c.gamma_0-1.)*0.5*(u**2+v**2)
                return T * c.T0 - 273.15
            plotitem.plot_var = Temp
            plotitem.pcolor_cmin = - 200.

            #Only plot which has another color gradient
            plotitem.pcolor_cmap = colormaps.blue_yellow_red


    plotfigure0 = plotdata.new_plotfigure(name='Density [kg/m^3]', figno=0)
    default_plot_settings(plotfigure0, max=c.rho_0*c.rho)

    plotfigure1 = plotdata.new_plotfigure(name='Pressure [bar]', figno=1)
    default_plot_settings(plotfigure1, max = c.pr)

    plotfigure2 = plotdata.new_plotfigure(name='Mach_number  ', figno=2)
    default_plot_settings(plotfigure2, max = 3.5)

    plotfigure3 = plotdata.new_plotfigure(name='Temperature [C]', figno=3)
    default_plot_settings(plotfigure3, max = c.Tr-273.15)

    return plotdata


#Handles the source term with the
#general two-order Runge-Kutta-method
def source_term(solver,state,dt):

    q = state.q
    rad = Yc
    dt = solver.dt

    #The source term with arr[1,:,:]
    #and arr[2,:,:] being swapped
```

```
    def Psi(a):

        rho = a[0,:,:]
        u = a[1,:,:]/rho
        v = a[2,:,:]/rho
        E = a[3,:,:]
        p = E - gamma*(gamma-1.)*rho*0.5* (u**2 + v**2)

        arr = np.empty(a.shape)

        arr[0,:,:] = -rho*v/rad
        arr[1,:,:] = -rho*u*v/rad
        arr[2,:,:] = -rho*v**2/rad
        arr[3,:,:] = -v*(E+p*(gamma-1.))/rad

        return arr

    k1 = Psi(q)
    k2 = Psi(q+dt*0.5*k1)
    q = q + dt*k2

    state.q = q


#Calculates the quantities for the
#first set of plots
def Flow_calc(solver,state):

    q = state.q

    #Quantities for the bottle opening at r=0
    rho = q[0,xb,Nh]
    u = q[1,xb,Nh]/rho
    v = q[2,xb,Nh]/rho
    p = q[3,xb,Nh] - c.gamma_0*(c.gamma_0-1.)*rho*0.5*(u**2+v**2)
    T = p/rho

    Press.append(p)
    Temp.append(T)

    #Necessary so that their sizes are the same as for t
    if len(t) == 1:
        Press.append(p)
        Temp.append(T)

    dt = solver.dt
    t.append(t[-1]+dt)

    #If the desired point in time is reached
    if t[-1] >= time[0] and time[1]:
        time[0] = t[-1]*c.tau_0*1000.

        #Important so that the if statement
        #is only called once
        time[1] = False

        #Quantities along the x-axis for r=0
        u = q[1,:,Nh]/q[0,:,Nh]
        v = q[2,:,Nh]/q[0,:,Nh]
        p = q[3,:,Nh] - c.gamma_0 * (c.gamma_0-1.) * q[0,:,Nh] * 0.5 * (u**2+v**2)
        Temp2[:] = p/q[0,:,Nh]

        #Calulation of both M plots
        Mach[:] = np.sqrt((u**2+v**2)/Temp2)
        for i in range(num_cells_x):
            #If statement to avoid 1/0^2
            if u[i] == 0.:
                M_v[i] = 0.
            else:
                M_v[i] = np.sqrt(1./(1./u[i]**2 - 0.5*(c.gamma_0-1.)))


#Updates the aux-field before each time step
def update_aux(solver,state):

    q = state.q

    #The acting force on a wall
    def Force(ix,iy):

        #Quantities at a wall for r > 0
        u = q[1,ix,Nh:iy]/q[0,ix,Nh:iy]
        v = q[2,ix,Nh:iy]/q[0,ix,Nh:iy]
        p = q[3,ix,Nh:iy] - c.gamma_0*(c.gamma_0-1.)*q[0,ix,Nh:iy]*0.5*(u**2+v**2)
        sum = 0.
```

```
            for i in range(Nh,iy):
                sum += Yc[0,i]*p[i-Nh]
            return sum

        #Total force difference
        F = 2.*np.pi*dy*(Force(ic[0],c_top[0])-Force(ic[1],c_top[Lc]))

        acc = F/(c.gamma_0*c.rho_C*c.VC)
        a.append(acc)    #a[i]

        # Necessary so that its size is the same as for t
        if len(t) == 2:
            a.append(acc)

        dt = solver.dt
        t.append(t[-1]+dt)  #t[i+1]

        dt = [t[-2]-t[-3],t[-1]-t[-2]]  #dt_n, dt_{n-1}
        r = dt[1]/dt[0]
        vel = v[-2]*r**2 + v[-1]*(1.-r**2) + a[-1]*dt[1]*(1.+r)
        v.append(vel)    #v[i+1]
        dis = -s[-2]*r + s[-1]*(1.+r) + 0.5*a[-1]*dt[1]*(dt[1]+dt[0])
        s.append(dis)    #s[i+1]

        #aux[1,:,:] stores the cork's velocity
        state.aux[1,:,:] = vel

        #Second argument is not needed, therefore "dummy"
        ind_c,dummy = Ind_x(s[-1],s[-1]+dx)
        #-1 to address the cell center and not the interface
        ind_c -= 1
        if ind_c != ic[0]:
            if ind_c-ic[0] > 1:
                raise ValueError("The Cork moved too fast")

            #the wall's indices move by one
            ic[0] += 1
            ic[1] += 1

            #Fill the cells, now occupied by the fluid, with values of their left neighbors
            state.q[:,ic[0],c_bot[0]:c_top[0]] = state.q[:,ic[0]-1,c_bot[0]:c_top[0]]

            #Update the aux-values
            state.aux[0,ic[0]:ic[1]-1,:] = 1.
            for i in range(Lc+1):
                state.aux[0,ic[0]+i+1,c_bot[i]:c_top[i]] = 0.

        #Mach Disk Movement, quantities along x-axis
        vx = q[1,b_end:ic[0]+1,Nh]/q[0,b_end:ic[0]+1,Nh]
        vy = q[2,b_end:ic[0]+1,Nh]/q[0,b_end:ic[0]+1,Nh]
        pr = q[3,b_end:ic[0]+1,Nh] -c.gamma_0*(c.gamma_0-1.)*q[0,b_end:ic[0]+1,Nh]*0.5*(vx**2+vy**2)
        Te = pr/q[0,b_end:ic[0]+1,Nh]
        Ma = np.sqrt((vx**2+vy**2)/Te)

        #Returns the corresponding index if M reaches its maximum
        def d_ind(arr):
            max = -1.
            for i in range(b_end,ic[0]+1):
                if arr[i-b_end] > max:
                    max = arr[i-b_end]
                    ind = i
            return ind

        Dist.append(Xc[d_ind(Ma),0]-Xc[b_end,0])
        Dist2.append(Xc[ic[0],0]-Xc[d_ind(Ma),0])


    #Creates the first set of plots
    def plot_flow(t,Temp,Temp2,Press):

        #Transforms the quantities into
        #their dimensional form
        t = c.tau_0 * np.array(t)*1000.
        Press = c.p0*np.array(Press)
        Temp = c.T0 * np.array(Temp) - 273.15
        Temp2 = c.T0 * Temp2 - 273.15

        #T(q) vs T(p) plot as a function of time
        #validating the adiabatic expansion
        plt.figure()
        plt.xlabel("t [ms]")
        plt.ylabel("T [C]")
        plt.title("Temperature at the bottle opening")
        plt.plot(t,Temp,label="T(q)")
        exp = (c.gamma_0-1.)/c.gamma_0
```

```
        T_isent = c.Tr * (Press/c.pr)**exp - 273.15
        plt.plot(t,T_isent,label="T(p)")
        plt.legend()

        #Measure the average error between the two temperatures
        err = 0.
        for i in range(len(Temp)):
                err += abs(Temp[i]-T_isent[i])
        err /= len(Temp)
        print("N = ",num_cells_x, "  err = ", err, "  dt = ", delta_t)

        #Temperature plot for r=0 at a given point in time
        #showing the jump at the position of the Mach disk
        plt.figure()
        plt.xlabel("[cm]")
        plt.ylabel("T [C]")
        # ylim=-200 or -150 are used as the starting point
        plt.ylim(-200.,50)
        plt.title("Temperature for r=0 at t= "+"{:.3f}".format(time[0])+" [ms]")
        plt.plot(Xc[:,0]*c.d0,Temp2)
        plt.axhline(y=0.,color="black",linestyle="dotted")
        plt.axhline(y=20., color="red", linestyle="dotted", label="T_0 = 20 C")
        plt.axvline(x=c.L*c.d0,color="green",linestyle="dotted",label="L = 5.26 cm")
        plt.legend(loc="lower right")

        #M(q) vs M(v) plot for r=0 at a given point in time
        #disproving that the fluid is stationary
        plt.figure()
        plt.xlabel("[cm]")
        plt.ylabel("M")
        plt.title("Mach for r=0 at t= " + "{:.3f}".format(time[0]) + " [ms]")
        plt.plot(Xc[:,0]*c.d0,Mach,label="M(q)")
        plt.plot(Xc[:,0]*c.d0,M_v,label="M(v)")
        plt.axhline(y=1., color="black", linestyle="dotted")
        plt.axvline(x=c.L * c.d0, color="green", linestyle="dotted",label="L = 5.26 cm")
        plt.legend(loc="upper right")

        #Show the plots simultaneously
        plt.show()


#Creates the second set of plots
def plot_mov(t,s,v,a,Dist,Dist2):

        #Transforms the quantities into
        #their dimensional form
        t = c.tau_0*np.array(t)*1000.
        s = c.d0*np.array(s)
        v = c.c0*np.array(v)
        # Necessary so that its size is the same as for t
        a.append(a[-1])
        a = c.c0**2/c.d0*np.array(a)*0.1
        Dist = c.d0*np.array(Dist)
        Dist2 = c.d0 * np.array(Dist2)

        #sc
        plt.figure()
        plt.xlabel("t [ms]")
        plt.ylabel("[cm]")
        plt.title("Position of the cork")
        plt.plot(t,s)

        #vc
        plt.figure()
        plt.xlabel("t [ms]")
        plt.ylabel("[m/s]")
        plt.title("Velocity of the cork")
        plt.plot(t,v)

        #ac
        plt.figure()
        plt.xlabel("t [ms]")
        plt.ylabel("[km/s^2]")
        plt.title("Acceleration of the cork")
        plt.plot(t,a)

        #distances Shock-Bottle vs Cork-Shock
        plt.figure()
        plt.xlabel("t [ms]")
        plt.ylabel("[cm]")
        plt.title("Position of the shock wave")
        plt.plot(t,Dist,label="d(Shock-Bottle)")
        plt.plot(t,Dist2,label="d(Cork-Shock)")
        plt.legend()
```

```
    #Show the plots simultaneously
    plt.show()

#########################
###START MAIN PROGRAM###

#Solver using the custom euler solver
solver = pyclaw.ClawSolver2D(riemann.custom_euler_solver)

#Total cell number, also 200 was used,
#must be even so that the source term functions properly
num_cells = 400

ymax = 2.5
num_cells_y = num_cells
y = pyclaw.Dimension(-ymax,ymax,num_cells_y,"y")

#xmax=8 and 10 are used
xmax = 8.#10.#
#Same total cell number as for the y-axis
num_cells_x = num_cells
x = pyclaw.Dimension(0.,xmax,num_cells_x,"x")

domain = pyclaw.Domain([x,y])
#4 euler equations
solver.num_eqn = 4
#2 waves for the Roe solver
solver.num_waves = 2
#1 ghost cell needed for Godunov's method
solver.num_ghost = 1

#The order of the solver defines
#the number of ghost cells
solver.order = solver.num_ghost

#First aux field for boundary conditions
#Second one for the cork's velocity
num_aux = 2

state = pyclaw.State(domain,solver.num_eqn,num_aux)

#Position of the cell interfaces
X,Y = state.grid.p_nodes
#Position of the cell centers
Xc,Yc = state.grid.p_centers
#Total number of cell interfaces
M,N = X.shape
#Index for r=0
Nh = int(N/2)
#Total cell number
Mm,Nm = M-1,N-1
#Grid distances
dx,dy = state.grid.delta

#x-indices of the bottle's walls
b_st,b_end = Ind_x(0,c.L)
#y-indices of the bottle's walls
b_bot,b_top = Ind_y(b_st,b_end,f_B)
#Position of the cork's left wall
xc_st = c.L + dx
#x-indices of the cork's walls
c_st,c_end = Ind_x(xc_st,xc_st+2.)
#y-indices of the cork's walls
c_bot,c_top = Ind_y(c_st,c_end,f_C)
#Number of indices along the x-axis
#occupied by the cork
Lc = len(c_top)-1

#Gamma for air
gamma = c.gamma_0
state.problem_data["gamma"] = gamma

#Determines the set of plots
#plot=1: shows the quantities without the cork
#plot=2: with the cork
plot = 1

#Thickness of the wall
tkn = 2

#Inital values for q
state.q[0,:,:] = init_values(c.rho)
state.q[1,:,:] = 0.
state.q[2,:,:] = 0.
state.q[3,:,:] = init_values(c.E)
```

```
#Auxiliary fields
state.aux[0,:,:] = aux_init()
state.aux[1,:,:] = 0.

#Boundary conditions for the domain's edges
solver.bc_lower[0] = pyclaw.BC.extrap
solver.bc_upper[0] = pyclaw.BC.extrap
solver.bc_lower[1] = pyclaw.BC.extrap
solver.bc_upper[1] = pyclaw.BC.extrap

#Only for Clawpack. Does not serve any computational purpose.
solver.aux_bc_lower[0] = pyclaw.BC.extrap
solver.aux_bc_upper[0] = pyclaw.BC.extrap
solver.aux_bc_lower[1] = pyclaw.BC.extrap
solver.aux_bc_upper[1] = pyclaw.BC.extrap

#Implements the source term
solver.step_source = source_term

if plot == 1:
    #xb is the last x-index occupied by
    #the fluid inside the bottle
    xb = b_end - 1
    #time[0]: certain point in time: t1=2., t2=6.
    #time[1]: decides if a statement is called or not
    time = [6., True]
    #Usual Python lists, without a defined size
    #so that entries can just be appended
    t = [0.]
    Temp = []
    Press = []
    #Numpy arrays with a fixed size
    Temp2 = np.empty(num_cells_x)
    Mach = np.empty(num_cells_x)
    M_v = np.empty(num_cells_x)

    #Calls the corresponding function before each time step.
    solver.before_step = Flow_calc

elif plot == 2:
    #Must start with 2 values for the Taylor-series
    t = [-solver.dt,0.]
    ic = [c_st-1,c_end]
    s = [Xc[ic[0],0],Xc[ic[0],0]]
    v = [0.,0.]
    #Only a does not have this requirement
    a =[]
    Dist = [0.,0.]
    Dist2 = [0.,0.]

    # Calls the corresponding function before each time step.
    solver.before_step = update_aux

#Initializing the controller
claw = pyclaw.Controller()
claw.solver = solver
claw.solution = pyclaw.Solution(state,domain)
claw.setplot = setplot
claw.keep_copy = True

#The number of created plots for each quantity
claw.num_output_times = 40
#The final time, where the simulation stops
#tfinal= 40, 20, 10 are used
claw.tfinal = 10.

#Runs the simulation and measures the running time
import time as tt
start = tt.time()
claw.run()
end = tt.time()
delta_t = end-start

#Plots the first set without the cork
if plot == 1:
    plot_flow(t,Temp,Temp2,Press)

#Plots the second set with the cork
elif plot == 2:
    plot_mov(t,s,v,a,Dist,Dist2)

#After the one_dimensional plots
#finally shows the 2D heat-maps
claw.plot()
```

8 References

Note, that all websites were most recently visited in October 2021.

[1] Liger-Belair, G., Cordier, D., Georges, R.: *Under-expanded supersonic CO2 freezing jets during champagne cork popping*, Science Advances, **5**, 1-9, 2019.

[2] Henry's law: https://en.wikipedia.org/wiki/Henry%27s_law

[3] Braun, S.: *Strömungslehre für TPh*, lecture notes, TU Wien, 2016.

[4] Leveque, R.J.: *Finite-Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2004.

[5] CLAWPACK homepage: https://www.clawpack.org/

[6] CLAWPACK download: https://github.com/clawpack/clawpack/releases/tag/v5.8.0

[7] PyCharm download: https://www.jetbrains.com/pycharm/download/

[8] PyCharm WSL: https://www.jetbrains.com/help/pycharm/using-wsl-as-a-remote-interpreter.html

[9] Xming download: https://sourceforge.net/projects/xming/

[10] Custom Riemann solver: https://www.clawpack.org/pyclaw/problem.html

[11] Geometry: https://www.clawpack.org/pyclaw/geometry.html

[12] Riemann solver: https://www.clawpack.org/riemann.html

[13] State: https://www.clawpack.org/pyclaw/state.html

[14] Solution: https://www.clawpack.org/pyclaw/solution.html

[15] Controller: http://www.clawpack.org/pyclaw/controller.html

[16] Roe, P.L.: *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*, Journal of Computational Physics, **43**, 357-372, 1981.

[17] Einfeldt, B., Munz, C.D., Roe, P.L., Sjögreen, B.: *On Godunov type methods near low densities*, Journal of Computational Physics, **92**, 213-295, 1991.

[18] Einfeldt, B.: *On Godunov-type methods for gas dynamics*, SIAM Journal on Numerical Analysis, **2**, 294-317, 1988.

[19] Runge-Kutta-method: https://en.wikipedia.org/wiki/Runge-Kutta_methods

[20] Ambient gas pressure: https://en.wikipedia.org/wiki/Atmospheric_pressure

[21] Ambient air density: https://en.wikipedia.org/wiki/Density_of_air

[22] Heat capacity ratio: https://en.wikipedia.org/wiki/Heat_capacity_ratio

[23] Freezing point of dry ice: https://en.wikipedia.org/wiki/Dry_ice

[24] Density of the cork: https://www.aqua-calc.com/page/density-table/substance/cork-coma-and-blank-solid

[25] Sundqvist, H., Veronis, G.: *A simple finite-difference grid with non-constant intervals*, Tellus, 22:**1**, 26-31, 1970.

[26] Bernoulli's principle: https://en.wikipedia.org/wiki/Bernoulli%27s_principle

[27] Gas constant: https://en.wikipedia.org/wiki/Gas_constant

[28] Sutherland model: https://de.wikipedia.org/wiki/Sutherland-Modell

[29] Gravitational acceleration: https://en.wikipedia.org/wiki/Gravitational_acceleration

[30] Ambient thermal conductivity: https://en.wikipedia.org/wiki/List_of_thermal_ conductivities