ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Fluid Dynamics

Simulation and validation of compressible flow in nozzle geometries and validation of OpenFOAM for this application

Benjamin Wüthrich

Computational Science and Engineering MSc

Master Thesis SS 07

Institute of Fluid Dynamics
ETH Zurich

Written at
ABB Corporate Research
Baden-Dättwil

Supervisors: Dr. H. Nordborg
Dr. Y.-J. Lee
Professor: Prof. Dr. L. Kleiser

Abstract

In this thesis, the open source CFD software framework OpenFOAM is evaluated with regard to its suitability for the simulation of supersonic compressible flows as they occur during arc extinguishing in high-voltage circuit breakers. After a general introduction to circuit breakers and the switching case of interest, switching of capacitive currents, a short overview of the functionality of OpenFOAM is given. A selection of compressible flow solvers is then tested in two verification cases (shock tube and supersonic wedge flow) and two validation cases (backward facing step and transonic diffuser flow). Results include the elimination of some solvers from further analysis, high demands on grid refinement for accurate simulation of recirculation and satisfactory performance for a normal shock/flow separation scenario. Taking the lessons learned from these cases into account, a series of cold gas flow simulations for ABB circuit breaker geometries is run and the results are compared to experimentally obtained values, again with a satisfactory outcome. The largest deviations from measured values have their roots always in a false estimation of shock locations. The flow phenomena encountered in this thesis comprise normal and oblique shocks, expansion waves, flow separation and reattachment as well as recirculation. The main result of the present work is a recommendation to use OpenFOAM as the basis for more complete simulations of circuit switching and arc extinguishing.

Contents

List of Figures

Notation

Roman symbols

Symbol	Description	Units
A_s	Sutherland coefficient	$[\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s}\cdot\mathrm{K}^{1/2})]$
a	Local speed of sound	[m/s]
c_p	Specific heat capacity at constant pressure	$[\mathrm{J}/(\mathrm{kg}\cdot\mathrm{K})]$
C_μ	Turbulent-viscosity constant in the k–ε turbulence model	—
c_v	Specific heat capacity at constant volume	$[\mathrm{J}/(\mathrm{kg}\cdot\mathrm{K})]$
Co	Courant number	—
e	Specific internal energy	[J/kg]
H_f	Heat of fusion	[kJ/kg]
h	Specific enthalpy	[J/kg]
h	Step height for the backward facing step case, channel height for the diffuser case	[m]
h_*	Throat height of the transonic diffuser	[m]
k	Turbulent kinetic energy	$[\mathrm{m}^2/\mathrm{s}^2]$
L_i	Distance from inlet to step	[m]
L_o	Distance from step to outlet	[m]
L_u	Distance from step to upper boundary	[m]
$\mathcal{M}$	Molecular weight	[u]
Ma	Mach number	—
Ma_n	Normal component of the Mach number	—
Ma_s	Moving shock Mach number	—
p	Pressure	[Pa]
p_0	Total pressure	[Pa]
Pr	Prandtl number	—
R	Specific gas constant	$[\mathrm{J}/(\mathrm{kg}\cdot\mathrm{K})]$
R	Exit static to inflow total pressure ratio	—
T	Temperature	[K]
T_s	Sutherland temperature	[K]
T_0	Total temperature	[K]
t	Time	[s]
$\mathbf{u}$	Velocity field	[m/s]
u	x-component of velocity/component parallel to shock	[m/s]
u_p	Velocity of gas behind the normal shock wave	[m/s]
v	Specific volume	$[\mathrm{m}^3/\mathrm{kg}]$
v	y-component of velocity	[m/s]
W	Wave velocity	[m/s]
w	z-component of velocity/component perpendicular to shock	[m/s]

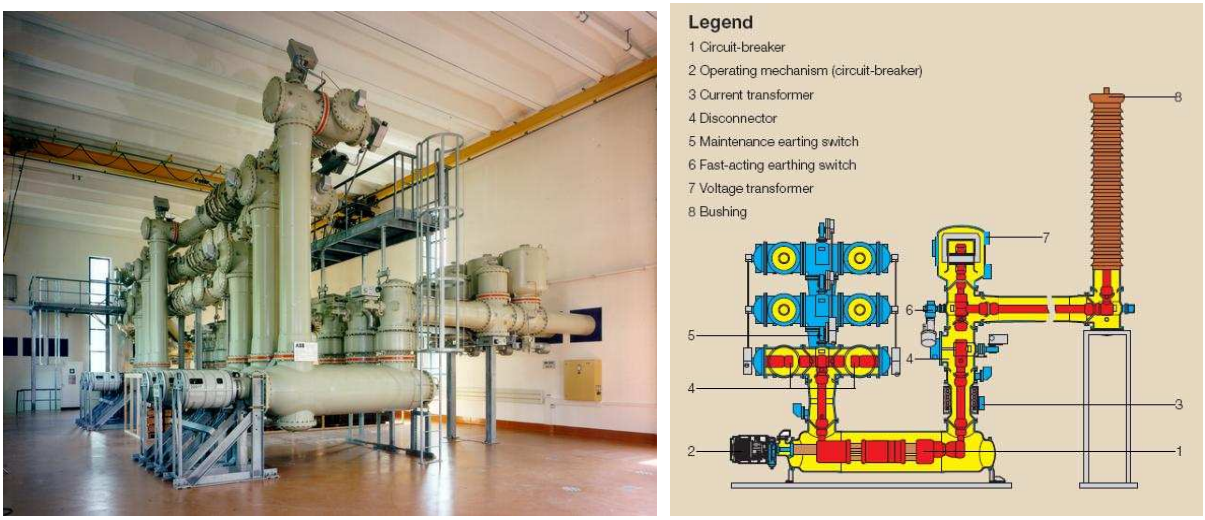
continued on next page

continued

Symbol	Description	Units
x	Position along the x-axis	[m]
y	Position along the y-axis	[m]
z	Position along the z-axis	[m]

Greek symbols

Symbol	Description	Units
β	Shock wave angle	[rad]
γ	Heat capacity ratio c_p/c_v	—
ε	Turbulent dissipation rate	$[\mathrm{m}^2/\mathrm{s}^3]$
$\varepsilon_{\mathrm{rms}}$	Root mean square error	—
θ	Deflection angle	[rad]
κ	Thermal conductivity	$[\mathrm{W}/(\mathrm{m}\cdot\mathrm{K})]$
μ	Dynamic viscosity	$[\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s})]$
ρ	Density	$[\mathrm{kg}/\mathrm{m}^3]$



(a) (b)

Figure 1.1: Gas insulated switchgear, type ABB ELK-3 (voltage up to 550 kV): (a) photography; (b) schematic drawing. (Source: http://www.abb.com/)

1 Introduction

Switch components are a key element in the journey of electrical energy from the generator to the consumer; the reliability of our energy supply depends heavily on their proper functioning. Switchgear enables the actual distribution of electricity and the combination of load as necessary: almost every major branch is connected to a hub by means of a switching device.

Circuit breakers are an integral part of switchgear: they take care of protecting their respective circuit from faults such as overload or short circuit by actually—as their name suggests—breaking the circuit. This is effected by mechanically opening a contact, much like pulling the plug of a household appliance; the difference is that circuit breakers do so automatically at the right time and that for higher voltages, some issues arise when "pulling the plug".

A switching device for high voltage (up to 500 kV) is shown in Fig. 1.1; our principal interest lies in the part labelled "1", the circuit breaker. The device seen here is so called gas insulated switchgear (GIS): the contacts are insulated by pressurised SF_6, a gas with excellent insulating properties.

Switching high voltages gives rise to the phenomenon of the *light arc*: the gas between the contacts is ionised and becomes a conductor. To definitely break the circuit, this arc has to be quenched. One of the methods to do so is the motivation for this thesis.

In Section 1.1, a type of circuit breakers that use the energy of the arc to have it extinguish itself is presented. Section 1.2 outlines the difficulties in simulating this process; in Section 1.3, the ultimate goal of this thesis is described. Section 1.4 finally explains the typesetting conventions of this document and gives an overview of the content of the other chapters.

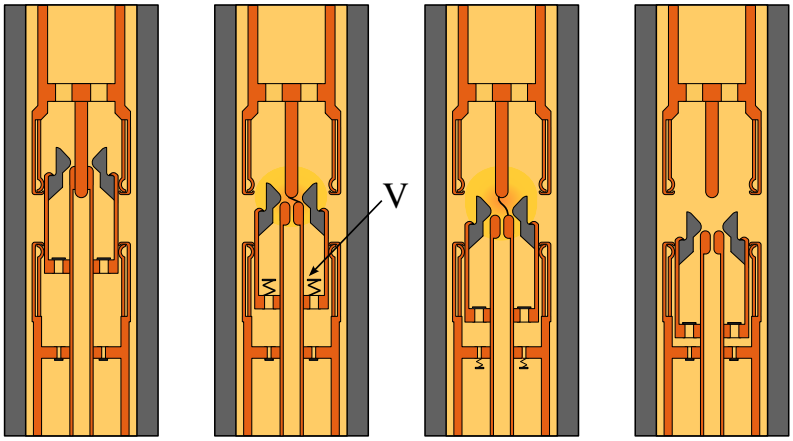


Figure 1.2: Functioning principle of a self-blast circuit breaker: the heat of the arc is used to build up pressure, which induces a flow back to the arc, finally extinguishing it (source: http://en.wikipedia.org).

1.1 Self-blast circuit breakers

One of the measures to quench the arc is to cool it sufficiently. In simple terms, this is effected by a gas flow along the arc to withdraw heat from it; it is "blown out". SF_6 is especially well suited for this task because of its insulating properties. The gas flow can be caused by, for example, a piston; breakers where the relative movement between the gas and the arc is provided by the arc itself are called *self-blast breakers*. Figure 1.2 illustrates the idea: after the contact is opened, the arc starts to burn. In alternating current high-voltage circuits (the case we are interested in), it may extinguish at every current zero, but reignites immediately after passing zero crossing. The heat being generated by the arc is now used to build up pressure in a special chamber, from which gas flows back to the arc, now burning in a nozzle/diffuser geometry. The goal is to avoid reignition after the next zero crossing.

Of the various switching cases (terminal fault, closing of inductive currents etc.), the switching of *capacitive currents* is of special interest for our purposes: if restriking of the arc occurs, the voltage load on the breaker can reach a multiple of the peak value of the driving voltage.

The goal of the gas flow is on the one hand to provide for cooling, but on the other hand, low pressure and the associated low density caused by high speed flow must be avoided, because it lowers the dielectric strength of the insulating gas and facilitates restriking of the arc.

More details about switchgear, physical foundations and the various switching cases can be found in Lindmayer (1987) and Gremmel & Kopatsch (2007); Fröhlich (2006) is a little more general and less in-depth.

1.2 Challenges in simulating the switching procedure

If one was able to simulate the complete breaker procedure, including the plug movement, arc ignition, electromagnetism, fluid dynamics (supersonic and turbulent flow with

shocks), fluid-structure interaction and the physics of the transient burning arc, circuit breaker development would profit immensely. Understandably, this has not been achieved yet but is a declared long-term goal. However, already coupling computational electromagnetics (CEM) and computational fluid dynamics (CFD) to computational magnetohydrodynamics (CMHD) for compressible fluids is a very challenging task.

Various attempts at solving or simulating at least partial aspects of the above problem have been undertaken:

- Kim *et al.* (2003) describes how to optimise the shape design of gas circuit breakers by means of an evolutionary algorithm. For evaluation of the objective function, an Euler finite volume solver is used.
- Wolter (1997) examines CFD codes with respect to their suitability for gas flow simulations in high voltage circuit breakers, quite similar to the topic of the present thesis.
- Mantilla Florez (2007) creates a robust reference experiment after which simulations could be assessed and compares CFD-ACE+ predictions to the measured values.

1.3 Objective of this thesis

To get closer to the goal of simulating the complete circuit breaking procedure, there are basically two alternatives: software could be developed from scratch, or existing software could be extended. Closed-source commercial codes such as Fluent, ANSYS CFX or CFD-ACE+ are naturally not suitable for extensions, thus open source software must serve as a basis.

OpenFOAM is such a software (see Chapter 2 for details about OpenFOAM): open source and especially designed with extensibility in mind. The goal of this thesis is thus:

> **Evaluation of OpenFOAM with respect to its suitability for flows as they occur in circuit breaker nozzle/diffuser geometries and laying the groundwork for making an informed decision as to whether OpenFOAM could be used as the basis for circuit breaker simulation specific extensions**

To this end, various test cases are to be performed; see the following section for details.

1.4 Conventions and structure of this report

1.4.1 Typesetting conventions

We follow largely the conventions used in OpenCFD (2007b):

- The names of utilities and solvers of OpenFOAM are typeset in sans-serif: Mach, sonicTurbFoam
- Directories and dictionaries are typeset in slanted sans-serif: *thermophysicalProperties*, *polyMesh*

- Command line examples, source code and specific settings/keywords in dictionaries are set in typewriter font: `Mach . myNozzle -latestTime`

1.4.2 Structure of this report

Chapter 2 gives an overview of OpenFOAM, the software used in this thesis.

In Chapter 3, two *verification* cases are examined: the shock tube problem and the supersonic wedge problem. The studies include mesh refinement analysis, comparisons among different OpenFOAM solvers and also commercial tools. The subject of Chapter 4 are two *validation* cases: the backward facing step and the Sajben transonic diffuser. The results are compared to experimentally obtained values and the predictions of commercial software.

Chapter 5 is a real-world application: cold gas flow in an ABB circuit breaker geometry and comparison to the experiment in Mantilla Florez (2007). Every case contains a "lessons learned" section; Chapter 6 summarises the learnings and insights of the whole thesis and outlines possible directions for future research activity.

The appendix contains a listing of the contents of the accompanying CD, source code of MATLAB scripts and functions as well as additional plots not included in the main chapters.

1.4.3 The difference between verification and validation

As it is used in this thesis and according to Gerritsma (2002), verification could be defined as "solving the equations right" and validation as "solving the right equations".

In *verification*, we are interested in learning whether our numerical method actually solves (or approximates) the (partial) differential equation describing our problem or if it converges to an erroneous solution. We do this by comparing the numerical solution to a known exact (analytical) solution.

Because theory is just an approximation of the physical reality, we also want to know if the equations we solve actually describe the real world; this is *validation*. We compare our result to experimental outcomes, so validation is actually a test of how well theory describes the physical reality—provided that we know our solution converges to the theoretical one, as established by verification.

2 OpenFOAM: A first glance

OpenFOAM stands for "Open Source Field Operation and Manipulation"; it is the software being evaluated in the course of this thesis. This chapter is intended to give the reader who is unfamiliar with it an idea of OpenFOAM; it is by no means an attempt at a complete documentation. More complete information can be obtained from OpenCFD (2007a) and OpenCFD (2007b); even though partially outdated and not complete either, these documents are probably the best starting points for OpenFOAM beginners.

In Subsection 2.1, a short outline of the history of OpenFOAM is given; Subsection 2.2 describes its features, and Subsection 2.3 gives an idea of what has to be done to run a case like the ones carried out in the later chapters.

2.1 History

OpenFOAM started as FOAM around 1993 at Imperial College, London, as a collaboration of Henry Weller and Hrvoje Jasak, who started working on his PhD thesis, Jasak (1996), at that time. The motivation to develop CFD software from scratch was mainly dissatisfaction with legacy codes in Fortran and the goal to create something reusable by others.

For a few years, FOAM was developed as a closed-source commercial software, before becoming open source in December 2004 with the announcement of OpenFOAM 1.0. Since then, four major releases were launched; the latest version is OpenFOAM 1.4.1, released in August 2007. OpenFOAM is, according to their website[1], used by R&D teams in large companies such as Audi, Bayer, Mitsubishi, Shell and Volkswagen, as well as by more than 200 academic institutions, among them Imperial College London, Chalmers University and the Tokyo Institute of Technology.

Main sources for information about OpenFOAM are, apart from the website and the users and programmer's guide mentioned above, the OpenFOAM message board[2] and the OpenFOAM Wiki[3].

2.2 Features

OpenFOAM is on the one hand a C++ *library*, on the other hand a collection of *applications* (created using these libraries). The applications can be divided into two different categories: *solvers* and *utilities*, of which the former perform the actual calculations and the latter provide a range of functionalities for pre- and post-processing.

2.2.1 Solvers

OpenFOAM covers an impressive range of applications with solvers ranging from a simple potential flow solver (potentialFoam) over incompressible steady-state (simpleFoam), transient laminar (icoFoam) turbulent (turbFoam) or dynamic mesh (icoDyMFoam) solvers,

[1] http://www.opencfd.co.uk/

[2] http://openfoam.cfd-online.com/cgi-bin/forum/discus.cgi

[3] http://openfoamwiki.net/index.php/Main_Page

compressible steady-state (rhoSimpleFoam) or trans- and supersonic turbulent (sonicTurbFoam) solvers to multiphase flow solvers (e. g., interFoam), LES solvers (oodles), combustion codes (dieselEngineFoam), electromagnetics (mhdFoam), solid stress analysis (solidDisplacementFoam) and even finance (financialFoam) solvers.

Naturally, the focus in this thesis lies on the compressible flow transient solvers; OpenFOAM comes with five of them. More on the solver selection process can be found in Chapter 3.

2.2.2 Utilities

The utilities can basically be divided into supporting pre- and post-processing tasks. There is also a tool called FoamX, which is actually just a GUI to effect changes in the different dictionary files and execute other utilities, instead of calling them directly from the command line. It works only with solvers for which a FoamX configuration file exists, and even its creators recommend switching to editing the files directly as soon as possible.

Utilities are usually called using

```
<utility> <root> <case> [-optionalParameters]
```

where `<utility>` is the name of the utility (e. g., blockMesh), `<root>` is the path to the root directory, and `<case>` is the path of the actual case, relative to the root directory. More about the directory structure can be found in Section 2.3. As an example, to calculate the Mach number for the latest time step in a case called *ABBnozzle*, one has to issue

```
Mach . ABBnozzle -latestTime
```

where the working directory has to be the root directory.

Pre-processing utilities Of the many pre-processing utilities that come with OpenFOAM[4], the following are used most often in the course of the test cases executed for this thesis:

- mapFields maps volume fields from one mesh to another; this is useful for mesh refinement studies to map results from a coarse mesh to a finer one without starting all over.
- blockMesh is the small included mesh generator. It is quite powerful in principle, but for more complicated meshes, it is recommended to use mesh conversion tools.
- checkMesh checks the mesh for validity, skewness and the like and gives information about its size.
- setFields is used to set initial conditions for the different volume fields, especially for cases where they are not uniform.
- fluentMeshToFoam converts Fluent meshes to OpenFOAM format. This is used for the ABB nozzle meshes in Chapter 5.

[4] A complete list of all utilities can be found in OpenCFD (2007b, Section 3.6).

Post-processing utilities The following post-processing utilities are the ones that offer functionality required for this thesis:

- Mach calculates the local Mach number and writes it at each time in a database.
- sample allows to sample arbitrary quantities at specified locations. This is used to compare to experiments or analytical solutions.
- foamLog extracts data such as initial residuals, iterations and Courant number from a log file for plotting and observing trends over longer periods of time.

To view and post-process simulations graphically, OpenFOAM comes with paraFoam, a reader module for the open source visualisation application ParaView[5]. To allow post-processing with 3rd party applications, data could be converted to other formats such as Fluent, EnSight or OpenDX, but ParaView is sufficient for our needs here.

2.2.3 Extensibility

One of the key advantages of OpenFOAM is its extensibility: the source code is accessible, and the architecture of OpenFOAM should make it easy to write for example a new solver or adapt existing ones. The creators take pride in the high level of abstraction, which is supposed to make the source code of a solver its own documentation. An equation such as

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot \varphi \mathbf{u} - \nabla \cdot \mu \nabla \mathbf{u} = -\nabla p \tag{2.1}$$

is represented by the code

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc:grad(p)
);
```

In this way, the understanding of the actual algorithm, the implemented models and equations are supposed to be much more important than a deep knowledge of object orientation and C++ programming. This author disagrees with this statement, see the critique in Subsection 3.1.6.

2.3 Running a case

Every OpenFOAM case has a similar structure with slight differences stemming only from the particular choice of solver. The basic file structure corresponds to the following list:

- *system*
  - *controlDict*
  - *fvSchemes*

[5] http://www.paraview.org/

  - *fvSolution*
  - Dictionaries for utilities like setFields or sample

- *constant*
  - *...Properties*
  - *polyMesh*
    - *points*
    - *cells*
    - *faces*
    - *boundary*

- Time directories (*0*, ...)

This section outlines the various steps to be undertaken when setting up a simulation in OpenFOAM: boundary conditions have to be set (Subsection 2.3.1), fluid properties selected (Subsection 2.3.2), numerical schemes and algorithms for the solution of systems of equations must be chosen (Subsection 2.3.3), and finally general simulations settings must be fixed (Subsection 2.3.4).

2.3.1 Mesh issues

OpenFOAM operates with unstructured polyhedron meshes where volumes can have any number of faces. After having obtained a mesh, either using blockmesh or one of the import utilities, the appropriate boundary types and conditions have to be set. To this end, the boundary types in the *boundary* dictionary have to be set to the right values. An excerpt from that dictionary might look like

```
inlet
{
    type patch;
    nFaces 50;
    startFace 10325;
}

bottom
{
    type symmetryPlane;
    nFaces 25;
    startFace 10375;
}

obstacle
{
    type patch;
    physicalType adiabaticWall;
    nFaces 110;
    startFace 10400;
}

defaultFaces
{
    type empty;
    nFaces 10500;
    startFace 10510;
}
```

where our focus lies on the lines with `type` in them. `patch` is a generic type, while `symmetryPlane` is for symmetry boundary conditions, and `empty` is to reduce the dimensionality of a problem[6].

When the mesh is created using blockMesh, the boundary types in *boundary* are already set as they should. If the mesh is imported however, boundary conditions are usually reset and must be edited again.

Apart from the *boundary* dictionary, every volume field dictionary has to be edited to set the right boundary conditions. The volume fields are contained in the time directories, whose name is the corresponding time level. Suppose we want to start a simulation from time zero, the field files in *0* have to be edited accordingly.

The part in the pressure file *p* where a fixed value of $p = 15\,\mathrm{kPa}$ for the inlet is to be prescribed could look like

```
inlet
{
    type        fixedValue;
    value       uniform 15e3;
}
```

The fixed velocity inlet (150 m/s in x-direction) and no-slip wall conditions in the velocity file *U* could be

```
inlet
{
    type        fixedValue;
    value       uniform (150 0 0);
}

bottom
{
    type        fixedValue;
    value       uniform (0 0 0);
}
```

In this manner, every field has to be edited, until all the boundary conditions are set. Depending on the type of solver, there might be only *p*, *T* and *U* dictionaries; for turbulent solvers, there are also *k* and *epsilon*. Details about boundaries can be found in OpenCFD (2007b, Section 6.2).

2.3.2 Fluid properties

Up next, the dictionaries in the *constant* directory are being looked at. For all the solvers used in this thesis, there is a *thermophysicalProperties* dictionary: it determines fluid model settings for the equation of state, whether c_p is constant or not, what transport model should be used and so on. This information is all put into one long string, followed by the corresponding parameter values. An excerpt (with comments) for a pure mixture called "air" with constant c_p using the constant transport model might look like

```
hThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas>>>>>

mixture         // keyword
air 1 28.9      // name of specie, number of moles, molecular weight (kg/kmol)
1000 2.544e6    // thermodynamic coefficients: cp and heat of fusion
1.8e-5 0.7      // transport coefficients: mu and Prandtl number
```

[6] OpenFOAM handles only 3D meshes, so to work on a 2D case, a mesh with a depth of one volume has to be created and `empty` boundary conditions specified along the dimension one wants to drop.

If the solver is turbulent, there is also a *turbulenceProperties* dictionary in which the turbulence model can be chosen and even the coefficients for every single model can be edited.

Details about models and physical properties can be found in OpenCFD (2007b, Chapter 8).

2.3.3 Schemes and solution algorithms

The *fvSolution* dictionary in the *system* directory controls solvers, tolerances and algorithms for the systems of equations solved to obtain every variable. Part of this dictionary might look like

```
solvers
{
    p ICCG  1e-06 0.01;
    U BICCG 1e-05 0.1;
    T BICCG 1e-05 0.1;
}

PISO
{
    nCorrectors              2;
    nNonOrthogonalCorrectors 0;
}
```

This means: when solving for pressure, use the Incomplete-Cholesky preconditioned conjugate gradient solver `ICCG` with a tolerance of 10^{-6} and a relative tolerance of 0.01. For velocity and temperature, use the Incomplete-Cholesky preconditioned biconjugate gradient solver `BICCG` with a tolerance of 10^{-5} and a relative tolerance of 0.1.

The *PISO* subdictionary specifies settings for the pressure-implicit split-operator method used for transient solvers: the number of corrections `ncorrector`, the number of corrections to account for mesh non-orthogonality `nNonOrthogonalCorrectors`, and sometimes more. The possible settings for the *fvSolution* dictionary are listed in OpenCFD (2007b, Section 4.5).

The *fvSchemes* dictionary on the other hand determines the numerical schemes for terms appearing in the constituent equations: time schemes, divergence, Laplacian terms and more. For every type of term, the user can choose from a comprehensive list of schemes. The first few lines of *fvSchemes* with second order implicit time stepping and a second order unbounded scheme for the $\nabla \cdot (\rho \mathbf{uu})$ divergence terms would look like

```
ddtSchemes
{
    default         implicit;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
```

The *fvSchemes* settings are listed in OpenCFD (2007b, Section 4.4).

2.3.4 Simulation control

The last dictionary with essential settings for every simulation is *controlDict* in the *system* directory. In this dictionary, the starting time `startTime`, the end time `endTime` and the time step `deltaT` are set; furthermore, the timing of writing output, its format and compression are determined.

More details about *controlDict* can be found in OpenCFD (2007b, Section 4.3).

With all the settings effected, a solver can be started just like a utility: to run our *ABBnozzle* case using the sonicTurbFoam solver, we would enter

```
sonicTurbFoam . ABBnozzle
```

provided that the current working directory is the root directory of the *ABBnozzle* case directory.

3 Verification cases

This chapter comprises two verification cases, i. e., problems for which an exact analytical solution is known (see Subsection 1.4.3). The intention is to identify the OpenFOAM solvers best suited for the kind of problem in question and, once this goal is reached, try to understand *why* they outperform the other solvers.

The documentation of OpenFOAM for users is not very detailed; the description of a solver in the User Guide, OpenCFD (2007b), is just a single phrase per solver. When looking for the optimal solver, we keep the projected application in mind: a supersonic flow of a compressible gas, possibly turbulent. The approach to select that optimal solver is then as follows:

- Compilation of a shortlist based on the requirements and the short descriptions in OpenCFD (2007b)
- Evaluation of the solver quality, comparison to analytical solution
- Analysis of temporal and spatial convergence, CPU time requirements

Ideally, this leads to the selection of one laminar and one turbulent solver.

Of all the solvers in OpenFOAM, only the ones classified as suitable for "compressible flow" are candidates for the shortlist. The descriptions of the five solvers chosen read as shown in Tab. 3.1:

rhopSonicFoam	Pressure-density-based compressible flow solver
rhoSonicFoam	Density-based compressible flow solver
sonicFoam	Transient solver for trans-sonic/supersonic, laminar flow of a compressible gas
rhoTurbFoam	Transient solver for compressible, turbulent flow
sonicTurbFoam	Transient solver for trans-sonic/supersonic, turbulent flow of a compressible gas

Table 3.1: Solver descriptions from OpenCFD (2007b).

The two verification cases selected are the *shock tube problem* (Section 3.1) and the *supersonic wedge problem* (Section 3.2).

3.1 The shock tube problem

This section deals with the well-known *shock tube problem* (also: *Riemann problem*). Subsection 3.1.1 describes the problem in detail and explains its relevance as a CFD verification case. In Subsection 3.1.2, the analytical solution is derived and a few comments about its implementation in MATLAB are given. In Subsection 3.1.3, the numerical solutions of the chosen solvers are compared to that analytical solution; Subsections 3.1.4 and 3.1.5 deal with temporal and spatial convergence behaviour. Subsection 3.1.6 is a review of the source code, Subsection 3.1.7 compares the results with a commercial product and Subsection 3.1.8 lastly summarises the findings and draws conclusions from them.

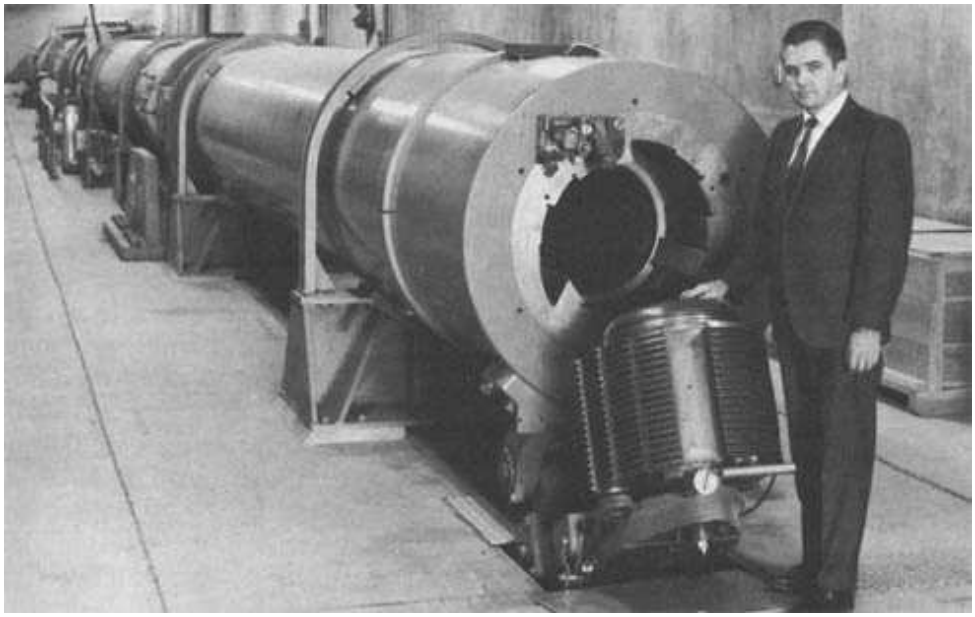


Figure 3.1: Example of a shock tube (source: http://history.nasa.gov/).

3.1.1 Description and relevance

The shock tube problem is a classical verification[1] case for CFD codes. Real-world shock tubes (see Fig. 3.1) are devices used for the study of travelling shock waves and the high-temperature, high-pressure gases created by them.

Basically, a shock tube is a very long pipe in which a strong shock wave is generated. In the initial configuration, the tube is divided by a diaphragm into two compartments: the *driver section* containing the gas with the higher pressure p_4 and the *driven section* with the gas at the lower pressure p_1. The gases can be of different species, i. e., have different molecular weights $\mathcal{M}$ and heat capacity ratios γ; furthermore, they might differ in temperature T and, as a consequence of all that, in speed of sound a. The initial conditions are depicted in Fig. 3.2.

After the diaphragm is broken, the following happens:

- A normal shock wave propagates into the driven section with velocity W
- The contact surface between the driver and driven gases moves at the velocity u_p of the gas behind the shock
- An expansion wave propagates into the driver section

This state is illustrated in Fig. 3.3.

The aspect of the problem making it interesting as a verification case for CFD is mainly the occurrence of discontinuities. Solvers could vary with respect to how well the shock and other discontinuities are captured and whether the numerical solution is "smeared" and/or spurious oscillations appear. Other than that, the flow field is likely to be very unspectacular since the problem is just one-dimensional.

[1] See the definition of "verification" (as opposed to "validation") in Subsection 1.4.3.

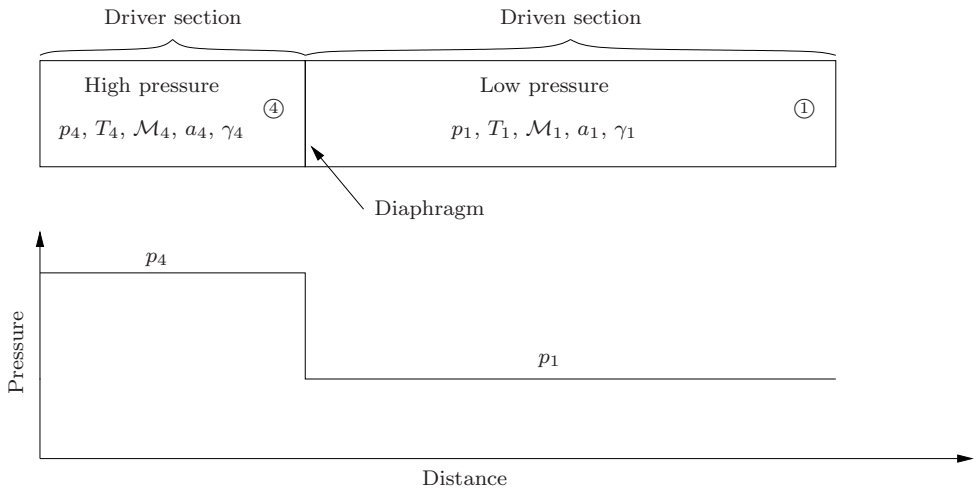


Figure 3.2: Shock tube initial conditions, pressure along the tube.

Setup used for testing For the actual testing, we utilise the configuration described in Slater (2005). The shock tube has a length of $3.048 \cdot 10^{-1}$ m and a radius of $3.048 \cdot 10^{-2}$ m. The diaphragm is placed in the exact middle at $x = 1.524 \cdot 10^{-2}$ m.

Both compartments are filled with air, so $\gamma = 1.4$, but at different pressures and temperatures (see Table 3.2)[2].

Compartment	Pressure (in Pa)	Temperature (in K)
Left (Driver)	$p_4 = 6.897 \cdot 10^4$	$T_4 = 288.89$
Right (Driven)	$p_1 = 6.897 \cdot 10^3$	$T_1 = 231.11$

Table 3.2: Initial values for the shock tube problem.

The tube is closed at both ends, but reflected waves are of no particular interest for our purposes, so solutions at a time after the first wave reaches a wall are disregarded. The tube walls are assumed to be slip walls and air is treated as a calorically perfect gas. The situation at time zero is depicted in Fig. 3.4.

3.1.2 Analytical solution

The derivation of an exact solution to the shock tube problem can be found in any textbook on compressible flows. Here, we follow Anderson (2003, Chapter 7), omitting the parts about reflected shock and expansion waves as well as some theory considered too general: on the following page, the right-running normal shock is treated, on page 18 the expansion wave, and on page 20, the solution to the complete problem is given; on page 21, the implementation in MATLAB is described.

[2]This seemingly odd choice of lengths is due to the fact that Slater (2005) uses U. S. customary units, so the tube dimensions correspond to 10 ft length and 1 ft radius. Consequentially, also pressure (from pound per square inch absolute) and temperature (from Rankine) are converted from "nice" numbers to somewhat odd ones.

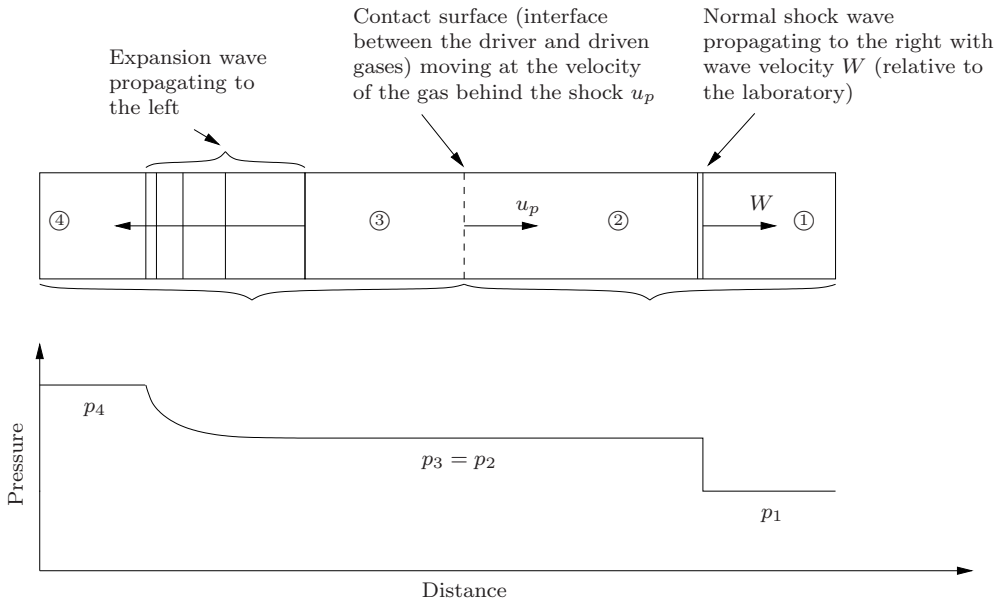


Figure 3.3: Shock tube after the diaphragm is broken.

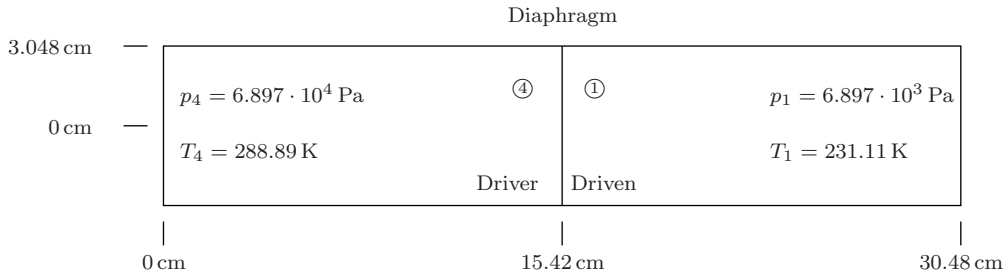


Figure 3.4: Setup for OpenFOAM solver evaluations at time $t = 0$.

The moving shock wave We start from the continuity, momentum and energy equations for a stationary normal shock wave:

$$\rho_1 u_1 = \rho_2 u_2 \tag{3.1}$$

$$p_1 + \rho_1 u_1^2 = p_2 + \rho_2 u_2^2 \tag{3.2}$$

$$h_1 + \frac{u_1^2}{2} = h_2 + \frac{h_2^2}{2} \tag{3.3}$$

where the index 1 refers to gas upstream of the wave, index 2 to gas downstream of the wave. The important point is that u_1 and u_2 are interpreted as velocities relative to the wave; because it is stationary, they are in this case also relative to the laboratory. For the moving wave as seen in Fig. 3.3, the velocities *relative to the wave* are W (for the gas ahead) and $W - u_p$ (for the gas behind). After replacing u_1 and u_2, Eqs. (3.1) to (3.3)

become

$$\rho_1 W = \rho_2 (W - u_p) \tag{3.4}$$

$$p_1 + \rho_1 W^2 = p_2 + \rho_2 (W - u_p)^2 \tag{3.5}$$

$$h_1 + \frac{W^2}{2} = h_2 + \frac{(W - u_p)^2}{2} \tag{3.6}$$

Equations (3.4) to (3.6) are the normal-shock equations for a shock moving with velocity W into a stagnant gas. They can be rearranged and substituted, using $h = e + p/\rho$, to obtain

$$e_2 - e_1 = \frac{p_1 + p_2}{2}(v_1 - v_2) \tag{3.7}$$

Equation (3.7) is the *Hugoniot equation*, which is also valid for a stationary shock. Because we assume air to be calorically perfect, $e = c_v T$ and $v = RT/p$; Eq. (3.7) becomes

$$\boxed{\frac{T_2}{T_1} = \frac{p_2}{p_1}\left(\frac{\frac{\gamma+1}{\gamma-1} + \frac{p_2}{p_1}}{1 + \frac{\gamma+1}{\gamma-1}\frac{p_2}{p_1}}\right)} \tag{3.8}$$

or

$$\boxed{\frac{\rho_2}{\rho_1} = \frac{1 + \frac{\gamma+1}{\gamma-1}\frac{p_2}{p_1}}{\frac{\gamma+1}{\gamma-1} + \frac{p_2}{p_1}}} \tag{3.9}$$

Equations (3.8) and (3.9) give the temperature and density ratios across the shock wave as functions of the pressure ratio.

We define the moving shock Mach number as

$$\mathrm{Ma_s} = \frac{W}{a_1}$$

By taking the calorically perfect gas relations and Eqs. (3.4) to (3.6) into account, we can derive

$$\frac{p_2}{p_1} = 1 + \frac{2\gamma}{\gamma + 1}(\mathrm{Ma_s^2} - 1)$$

or, solved for $\mathrm{Ma_s}$,

$$\mathrm{Ma_s} = \sqrt{\frac{\gamma + 1}{2\gamma}\left(\frac{p_2}{p_1} - 1\right) + 1} \tag{3.10}$$

Since $\mathrm{Ma_s} = W/a_1$, Eq. (3.10) leads to

$$\boxed{W = a_1\sqrt{\frac{\gamma + 1}{2\gamma}\left(\frac{p_2}{p_1} - 1\right) + 1}} \tag{3.11}$$

Equation (3.11) relates the velocity of the moving shock wave to the pressure ratio across the wave and the local speed of sound of the gas ahead of the shock wave.

The last quantity we are interested in is the velocity u_p of the mass motion induced by the shock wave. Equation (3.4) can be rewritten

$$u_p = W\left(1 - \frac{\rho_1}{\rho_2}\right) \tag{3.12}$$

After substitution of Eqs. (3.9) and (3.11) into Eq. (3.12), we obtain

$$\boxed{u_p = \frac{a_1}{\gamma}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{\frac{2\gamma}{\gamma+1}}{\frac{p_2}{p_1} + \frac{\gamma-1}{\gamma+1}}\right)^{1/2}} \tag{3.13}$$

With what we have derived so far, we can obtain for a given pressure ratio p_2/p_1 and speed of sound a_1 the corresponding values of ρ_2/ρ_1, T_2/T_1, W and u_p from Eqs. (3.8), (3.9), (3.11) and (3.13).

The next paragraph deals with the counterpart of the moving shock wave, namely the expansion wave.

The incident expansion wave While the last paragraph has been dealing with the relations between the regions 1 and 2 of Fig. 3.3, this paragraph examines the regions 3 and 4, i. e., the expansion wave between them. The formulas relate to a left-running expansions wave; they would be similar for a right-running one, except some changes of the signs in the equations.

To examine the expansion wave problem on its own, we use the fact that the gas in region 4 feels as if a piston was pulled to the right with velocity u_3. In fact, u_3 is the mass-motion velocity of the gas behind the expansion wave.

It can be shown (see e. g. Anderson (2003, Section 7.6)) that any part of a left-running finite wave moves with local velocity $u - a$. In region 4, the gas is at rest ($u_4 = 0$), so the head of the expansion wave moves with a velocity $u_4 - a_4 = -a_4$. This means that in the xt plane, the path of the head is a straight line (see Fig. 3.5).

Inside the wave, a mass motion with velocity u is induced, directed toward right. Temperature T and subsequently the local speed of sound a are reduced; because of this, the head of the wave propagates faster into region 4 than other parts of the wave, so the wave is spreading out while running down the tube.

The tail of the wave moves at velocity $\mathrm{d}x/\mathrm{d}t = u_3 - a_3$. Note that if $u_3 > a_3$, i. e., u_3 is supersonic, the tail of the wave propagates to the right although the wave is left-running. It can be shown that the characteristics of the expansion wave are straight lines and thus the values of u, a (and consequently also p, ρ, T etc.) are constant along them. Such a wave is called a *simple wave*; a *nonsimple* waves where the characteristics are curved comes up for example when the expansion wave is reflected.

To obtain a closed analytical form for the expansion wave, we use that

$$u + \frac{2a}{\gamma - 1} = \text{const through the wave} \tag{3.14}$$

(without derivation). Evaluation of Eq. (3.14) in region 4 gives

$$u_4 + \frac{2a_4}{\gamma - 1} = 0 + \frac{2a_4}{\gamma - 1} = \text{const} \tag{3.15}$$

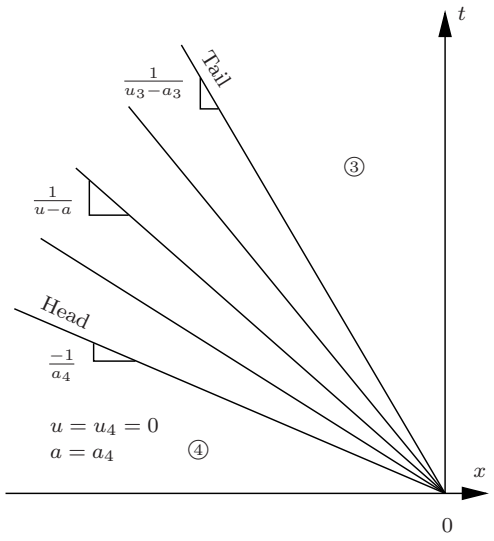


Figure 3.5: Characteristics for an expansion wave centred at 0.

and from combining Eqs. (3.14) and (3.15) we obtain

$$\boxed{\frac{a}{a_4} = 1 - \frac{\gamma - 1}{2}\left(\frac{u}{a_4}\right)} \tag{3.16}$$

which relates a and u within the expansion wave. Because $a = \sqrt{\gamma RT}$, Eq. (3.16) also gives

$$\boxed{\frac{T}{T_4} = \left(1 - \frac{\gamma - 1}{2}\left(\frac{u}{a_4}\right)\right)^2} \tag{3.17}$$

Because we assume the flow to be isentropic, $p/p_4 = (\rho/\rho_4)^\gamma = (T/T_4)^{2\gamma/(\gamma-1)}$, so from Eq. (3.17) we also find

$$\boxed{\frac{p}{p_4} = \left(1 - \frac{\gamma - 1}{2}\left(\frac{u}{a_4}\right)\right)^{2\gamma/(\gamma-1)}} \tag{3.18}$$

and

$$\boxed{\frac{\rho}{\rho_4} = \left(1 - \frac{\gamma - 1}{2}\left(\frac{u}{a_4}\right)\right)^{2/(\gamma-1)}} \tag{3.19}$$

Equations (3.16) to (3.19) give all the properties within the simple expansion wave as a function of the local gas velocity u.

To get them as a function of x and t, we look at any of the straight characteristics in Fig. 3.5:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = u - a$$

or, because the wave is centred, i. e., the characteristics are straight lines through the origin,

$$x = (u - a)t \tag{3.20}$$

Inserting Eq. (3.20) into Eq. (3.16) gives

$$x = \left(u - a_4 + \frac{\gamma - 1}{2}u\right) t$$

or, solved for u,

$$\boxed{u = \frac{2}{\gamma + 1}\left(a_4 + \frac{x}{t}\right)} \tag{3.21}$$

With this, the properties within the expansion wave, $-a_4 \leq x/t \leq u_3 - a_3$, are determined.

Shock tube relations Finally, we combine the findings from the last two paragraphs for a closed analytical solution to the shock tube problem. To this end, we recall that $u_3 = u_2 = u_p$ and $p_2 = p_3$ across the contact surface; u_p was obtained

$$u_p = u_2 = \frac{a_1}{\gamma_1}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{\frac{2\gamma_1}{\gamma_1+1}}{\frac{p_2}{p_1} + \frac{\gamma_1-1}{\gamma_1+1}}\right)^{1/2} \tag{3.13}$$

Appyling Eq. (3.18) to the tail of the expansion wave,

$$\frac{p_3}{p_4} = \left(1 - \frac{\gamma_4 - 1}{2}\left(\frac{u_3}{a_4}\right)\right)^{2\gamma_4/(\gamma_4-1)} \tag{3.22}$$

Solving Eq. (3.22) for u_3 gives

$$u_3 = \frac{2a_4}{\gamma_4 - 1}\left(1 - \left(\frac{p_3}{p_4}\right)^{(\gamma_4-1)/2\gamma_4}\right) \tag{3.23}$$

Because $p_3 = p_2$, Eq. (3.23) becomes

$$u_3 = \frac{2a_4}{\gamma_4 - 1}\left(1 - \left(\frac{p_2}{p_4}\right)^{(\gamma_4-1)/2\gamma_4}\right) \tag{3.24}$$

Finally, because $u_2 = u_3$, we can equate Eqs. (3.13) and (3.24) as

$$\frac{a_1}{\gamma_1}\left(\frac{p_2}{p_1} - 1\right)\left(\frac{\frac{2\gamma_1}{\gamma_1+1}}{\frac{p_2}{p_1} + \frac{\gamma_1-1}{\gamma_1+1}}\right)^{1/2} = \frac{2a_4}{\gamma_4 - 1}\left(1 - \left(\frac{p_2}{p_4}\right)^{(\gamma_4-1)/2\gamma_4}\right) \tag{3.25}$$

Equation (3.25) can be rearranged

$$\boxed{\frac{p_4}{p_1} = \frac{p_2}{p_1}\left(1 - \frac{(\gamma_4 - 1)(a_1/a_4)(p_2/p_1 - 1)}{\sqrt{2\gamma_1\left(2\gamma_1 + (\gamma_1 + 1)(p_2/p_1 - 1)\right)}}\right)^{-2\gamma_4/(\gamma_4-1)}} \tag{3.26}$$

Equation (3.26) gives the incident shock strength p_2/p_1 as an implicit function of the diaphragm pressure ratio p_4/p_1. We can now unfold a recipe for the solution of the shock tube problem, which consists of all the boxed equations:

1. Calculate p_2/p_1 from Eq. (3.26) to get the strength of the shock wave[3].
2. Calculate all other shock properties from Eqs. (3.8), (3.9), (3.11) and (3.13).
3. Calculate $p_3/p_4 = (p_3/p_1)/(p_4/p_1) = (p_2/p_1)/(p_4/p_1)$ to get the strength of the expansion wave.
4. All other properties behind the expansion wave can be found using

$$\frac{p_3}{p_4} = \left(\frac{\rho_3}{\rho_4}\right)^{\gamma} = \left(\frac{T_3}{T_4}\right)^{\gamma/(\gamma-1)}$$

5. The properties *inside* the expansion wave can be found from Eqs. (3.16) to (3.19) and (3.21).

The following subsection describes the implementation of this in MATLAB.

Implementation in MATLAB The complete source code of the function described here can be found in Section B.1. The function offers an interface

```
function [x_mesh,u,a,rho,T,p] = shocktube(time,p1,p4,T1,T4)
```

where `x_mesh` is a vector containing the x-positions where the solution is evaluated; `u`, `a`, `rho`, `T` and `p` are vectors of the size of `x_mesh` containing the local velocity, speed of sound, density, temperature and pressure at the corresponding positions.

`time` specifies at what time the solution is to be evaluated; `p1`, `p4`, `T1` and `T4` are the initial settings for pressure and temperature in the two compartments. The indices 1 and 4 refer to the scheme depicted in Fig. 3.3. If only `time` is given, the other variables are set to the values defined in Tab. 3.2.

After parsing the input and optionally setting the default values, some constants are specified[4]: the heat capacity ratio `gamma` (to 1.4), the specific gas constant of air `R` (to 287.05) and the position of the diaphragm `L1` (to 0.1524). Instead of centring the tube at zero, it was chosen to have its left end at zero (see Fig. 3.4).

Next, the local speeds of sound and the densities for compartments 1 and 4 are computed from the quantities known so far. To get the wave velocity `W` of the moving shock, we have to solve Eq. (3.26) for p_2/p_1, i. e., we have to find a p_2/p_1 satisfying

$$\frac{p_2}{p_1}\left(1 - \frac{(\gamma_4-1)(a_1/a_4)(p_2/p_1-1)}{\sqrt{2\gamma_1\left(2\gamma_1+(\gamma_1+1)(p_2/p_1-1)\right)}}\right)^{-2\gamma_4/(\gamma_4-1)} - \frac{p_4}{p_1} = 0 \tag{3.27}$$

Equation (3.27) is solved using the MATLAB function `fzero`, which employs a combination of bisection, secant, and inverse quadratic interpolation methods to find a zero. As an initial guess, $(p_4/p_1)/2$ is taken. From this, the pressure in region 2, `p2`, is obtained. With the pressure ratio p_2/p_1 at hand, all other shock properties can be calculated: temperature

[3] As this is an implicit equation, it has to be dealt with accordingly. Details can be found in every calculus textbook, e. g., Blatter (1996).

[4] All variables use SI units; the dimensions are given in the commentary of the source code.

T2, density rho2, local speed of sound a2, wave velocity W and the velocity of the mass motion behind the wave, u_p.

Since pressure and velocity are constant across the contact surface between regions 2 and 3, we know p3, u2 and u3; using these, the other quantities in region 3 can be determined.

Now, the mesh x_mesh is initialised, usually to a size of 1×100, but the number of points could also be changed to some other value if desired. Before iterating through all the solution vectors, the boundaries of the regions are determined using the knowledge about the velocities of head and tail of the expansion wave, the induced mass motion behind the shock wave and the velocity of the shock itself:

```
% Calculate boundaries of different zones.
% Boundary between leftmost driver gas and expansion wave.
x4_exp = L1 - time*a4;
% Boundary between expansion wave and lower pressure driver gas.
exp_x3 = L1 + time*(u3 - a3);
% Boundary between driver gas and driven gas.
x3_x2 = L1 + time*u_p;
% Location of the shock wave.
x2_x1 = L1 + time*W;
```

Now, the function iterates over all the points, checks what region the point belongs to and sets the corresponding value in the solution vectors u, a, rho, T and p. For points that are *in* the expansion wave, the nested function

```
function [u_exp,a_exp,rho_exp,T_exp,p_exp] = expansion_wave(x)
```

is called to calculate the properties using Eqs. (3.16) to (3.19) and (3.21).

Three MATLAB scripts on the accompanying CD generate movies, which visualise the pressure, density and Mach number progression calculated using shocktube.m; Tab. 3.3 lists their names and the name of the generated video file. See Appendix A for details on the CD.

Script	Video file	Quantity
shocktube_animation_p.m	shocktube_p.avi	Pressure p
shocktube_animation_rho.m	shocktube_rho.avi	Density ρ
shocktube_animation_Ma.m	shocktube_Ma.avi	Mach number Ma

Table 3.3: Scripts for video files visualising the analytical solution.

3.1.3 Solver quality evaluation

The first step in evaluating the solvers listed on page ?? consists in comparing the solutions obtained to the exact solution derived in Subsection 3.1.2. Details about the different meshes used are to be found below, the results are shown and commented on page 24.

The different meshes used Calculations are basically performed using four different meshes: one-dimensional, two-dimensional, axi-symmetric and three-dimensional. All meshes are structured—consist of hexahedral cells—and have a resolution of 100 cells in x-direction, so $\Delta x = 3.048 \cdot 10^{-4}\,\mathrm{m}$. Grading is uniform along all axes.

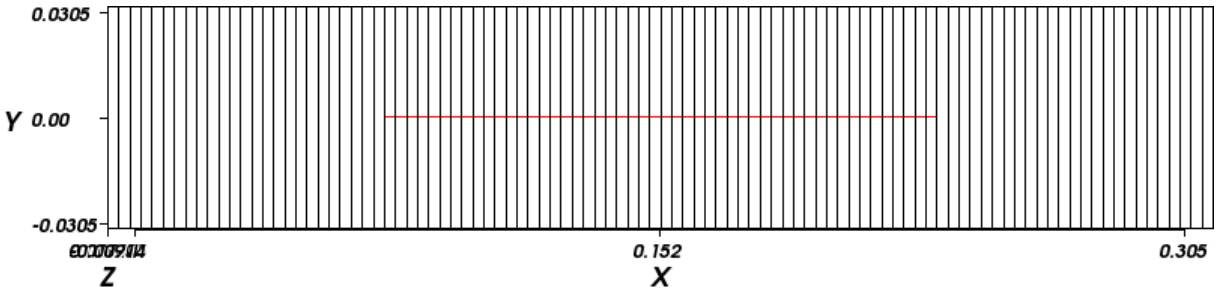


Figure 3.6: 1D mesh for the shock tube problem. The x-axis is from left to right, the y-axis is perpendicular to it in the paper plane, and the z-axis is perpendicular to the paper plane.

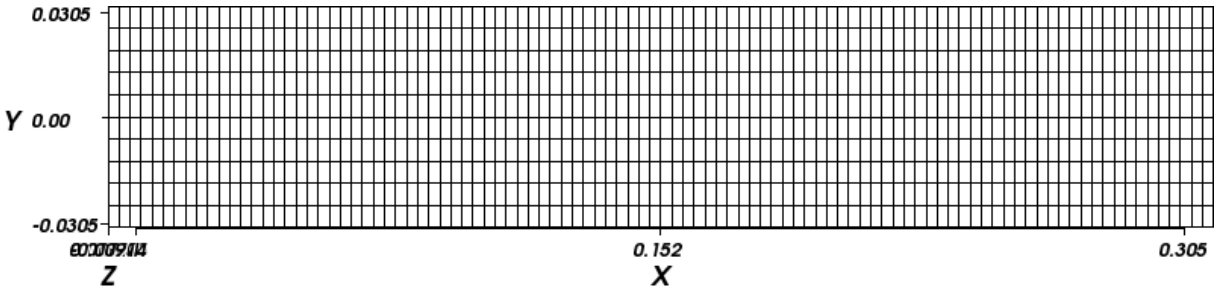


Figure 3.7: 2D mesh for the shock tube problem. The axes are oriented as in the 1D mesh (Fig. 3.6).

1D mesh Because OpenFOAM handles only 3D meshes, meshes for problems with a lower dimensionality still have to be at least one cell deep in every dimension. By specifying empty type patches at the corresponding boundaries, OpenFOAM is instructed to treat this dimension of the mesh as infinite, effectively reducing the number of dimensions. For the 1D mesh, this means having 100 cells in total with empty boundaries in y- and z-directions. The result is shown in Fig. 3.6.

2D mesh The 2D mesh has empty boundary conditions only on patches perpendicular to the y-axis. It has the same dimensions as the 1D mesh but has 10 cells in y-direction, leading to a cell count of 1'000. The mesh is shown in Fig. 3.7.

Axi-symmetric mesh For axi-symmetric problems, blockMesh allows the generation of wedge geometries. For use with the cylindric problem at hand, we follow the recommendations in OpenCFD (2007b) and create a wedge shaped block of one cell thickness and a 5° angle at the centre. The axisymmetric planes are specified as wedge type patches; the block has 5 cells in z-direction, leading to a cell count of 500. The outer boundary is actually curved, but this has no influence on the solution. The curved boundary patch is set to slip type. The resulting mesh is shown in Fig. 3.8.

3D mesh The full 3D mesh finally is created using five blocks, resulting in an O-grid type mesh. The yz cross section in Fig. 3.9a shows the central block having 10×10 cells and the other blocks having 5 cells in radial and 10 cells in azimuthal direction; the total

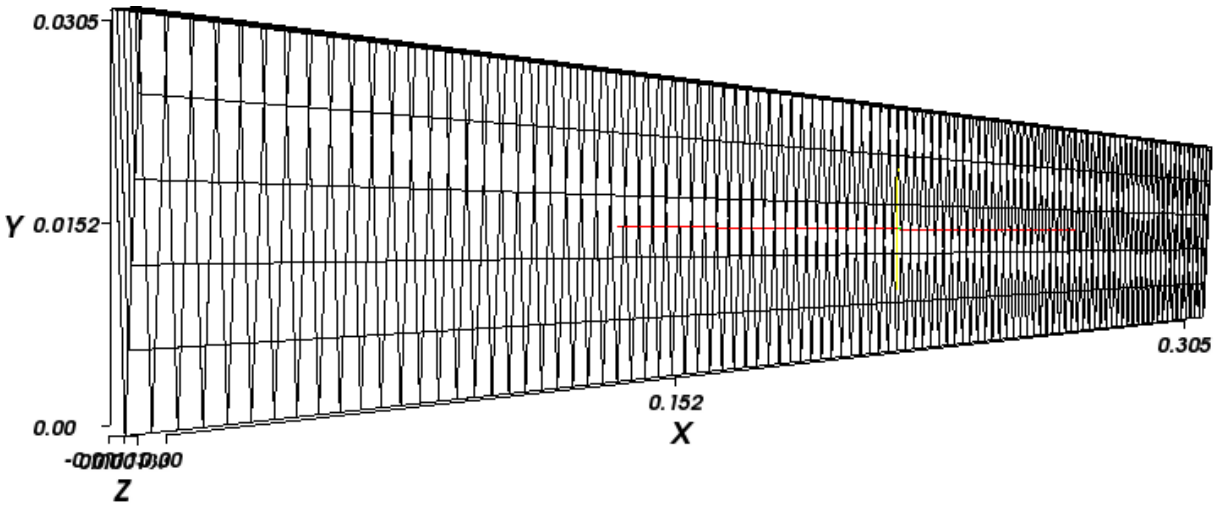


Figure 3.8: Axi-symmetric mesh for the shock tube problem. The wedge is aligned along the xy-plane.

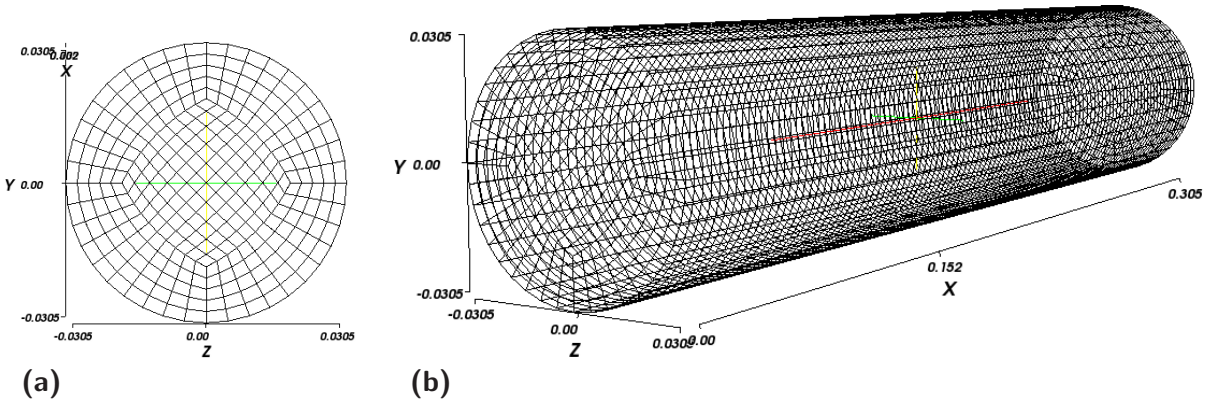


Figure 3.9: 3D mesh for the shock tube problem: (a) yz cross section; (b) perspective view.

cell count is 30'000. The mesh is shown in Fig. 3.9b.

Quality evaluation results The shock tube is included in OpenFOAM as an example for the solvers sonicFoam, rhoSonicFoam and rhopSonicFoam, namely the 1D case. We copy and modify it for our purposes.

The end time is set to $3 \cdot 10^{-4}$ s, a time where according to the exact solution the shock wave has not reached the end of the tube yet, so we can avoid dealing with reflected waves. Apart from `deltaT` (time step) and `writeInterval`, all settings[5] have been left in the quasi-default given by the example cases for the respective solvers. The three laminar solvers (rhopSonicFoam, rhoSonicFoam and sonicFoam) have the time step set to $\Delta t = 10^{-6}$ s; for the turbulent solvers (rhoTurbFoam, sonicTurbFoam) it is $\Delta t = 0.5 \cdot 10^{-7}$ s.

The time scheme is `Euler` (first order) for all solvers.

To compare the solvers, the pressure distribution obtained after $2.5 \cdot 10^{-4}$ s is com-

[5] I. e., all settings in the dictionaries *controlDict*, *fvSchemes* and *fvSolution*.

pared to the analytical solution. Following Slater (2005), also density and Mach number distributions are compared to their exact counterparts, but because pressure alone seems to be sufficient to judge the quality of the solvers, these plots are not shown here, but only in Appendix C.1. They basically show the same behaviour as the pressure plots, thus reinforcing the findings based on these.

Setting the initial conditions according to Fig. 3.4 is done by means of the setFields tool. The velocity field $\mathbf{u}$ is set to zero everywhere; the pressure p and the temperature T are set to the values given in Table 3.2. For the turbulent solvers, which are both using the k–ε turbulence model[6], the turbulent kinetic energy k and the turbulent dissipation rate ε have to be initialised as well.

k is given by

$$k = \frac{1}{2}(u'^2 + v'^2 + w'^2) \tag{3.28}$$

where u'^2, v'^2 and w'^2 are the fluctuating components of velocity in x, y and z direction. Following OpenCFD (2007b), we assume initially isotropic turbulence, i. e., $u'^2 = v'^2 = w'^2$, and that the fluctuations are equal to 5% of the expected velocity u_p:

$$u' = v' = w' = \frac{5}{100} u_p \approx \frac{5}{100} \cdot 267\,\mathrm{m/s} \tag{3.29}$$

Substituting Eq. (3.29) into Eq. (3.28) results in

$$k = \frac{3}{2}\left(\frac{5}{100} \cdot 267\right)^2 \frac{\mathrm{m}^2}{\mathrm{s}^2} \approx 267.3\,\frac{\mathrm{m}^2}{\mathrm{s}^2}$$

which is too large by far. As a remedy, the fluctuations are scaled by a factor l, which corresponds to the tube diameter times 0.07:

$$k = \frac{3}{2}\left(l \cdot \frac{5}{100} \cdot 267\right)^2 \frac{\mathrm{m}^2}{\mathrm{s}^2} \approx 7.789 \cdot 10^{-4}\,\frac{\mathrm{m}^2}{\mathrm{s}^2} \tag{3.30}$$

Equation (3.30) gives a result in the expected order of magnitude. For the dissipation rate ε, we have

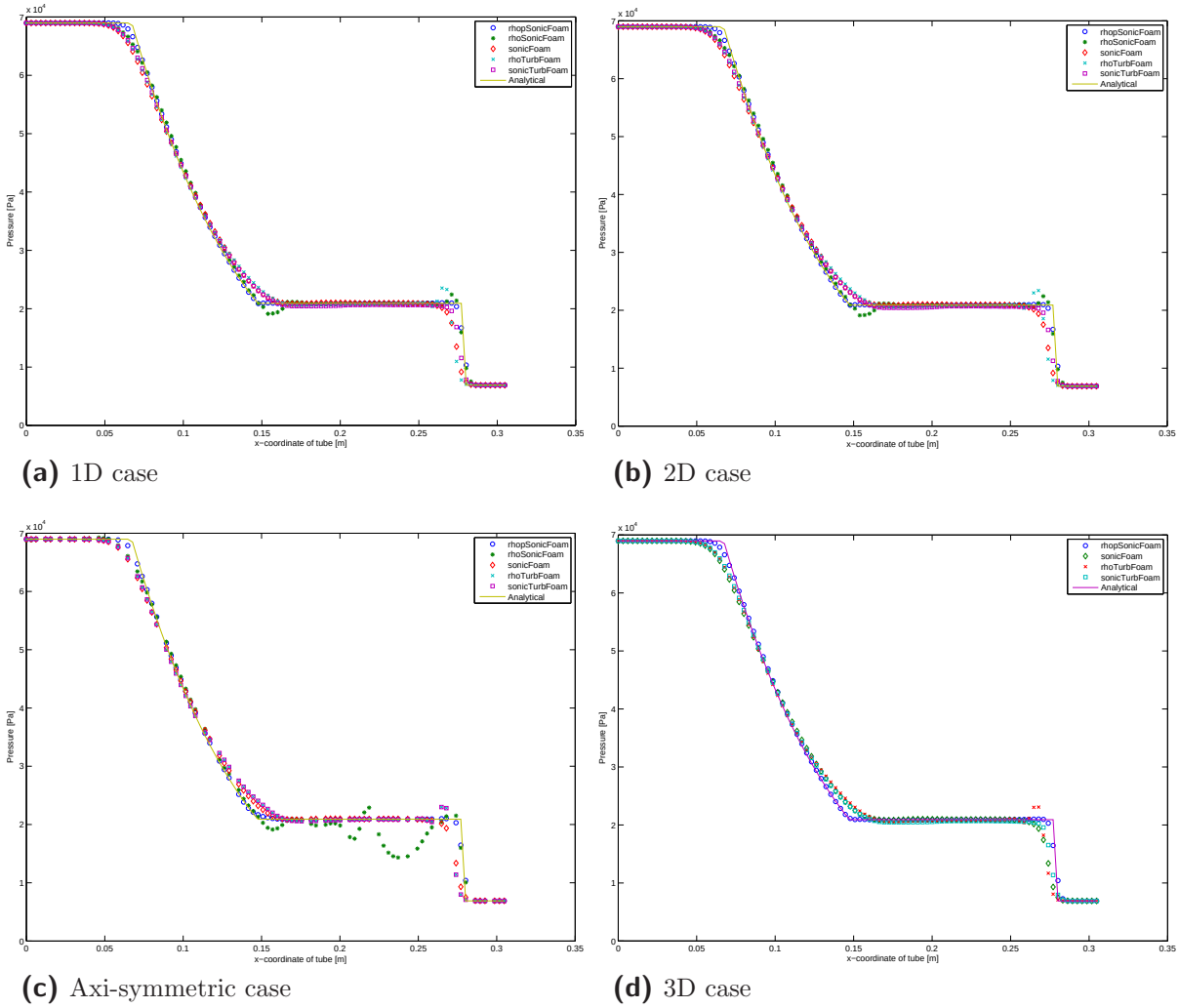
$$\varepsilon = \frac{\mathrm{C}_\mu^{0.75} k^{1.5}}{l} \tag{3.31}$$

where C_μ is a constant of the k–ε model[7] , $\mathrm{C}_\mu = 0.09$. Substituting this and Eq. (3.30) into Eq. (3.31) gives

$$\varepsilon = \frac{0.09^{0.75} \cdot (7.789 \cdot 10^{-4})^{1.5}}{0.07 \cdot (2 \cdot 3.048 \cdot 10^{-2})}\,\frac{\mathrm{m}^2}{\mathrm{s}^3} \approx 8.37 \cdot 10^{-4}\,\frac{\mathrm{m}^2}{\mathrm{s}^3} \tag{3.32}$$

[6]The question might arise, what the sense of testing a turbulent solver on a laminar case is. The reason is that like this, the quality of solving hyperbolic transport equations of the compressible flow can be assessed.

[7]The value of $\mathrm{C}_\mu = 0.09 = (0.3)^2$ stems from the empirical observation $|\langle uv \rangle|/k \approx 0.3$ in regions where $\mathcal{P}/\varepsilon$ is close to unity, $\mathcal{P}$ being the rate of production of turbulent kinetic energy. For details, see Pope (2000, Section 10.4).



(a) 1D case **(b)** 2D case

(c) Axi-symmetric case **(d)** 3D case

Figure 3.10: Pressure distribution at $t = 2.5 \cdot 10^{-4}$ s: comparison of OpenFOAM solvers.

For rhoTurbFoam and sonicTurbFoam, k and ε are set to the values obtained in Eqs. (3.30) and (3.32).

With all the necessary values set, we can start evaluating the solver quality. Fig. 3.10 shows the obtained pressure distributions for all four meshes. 100 samples are taken along the x-axis for $y = z = 0$ using the sample utility. The interpolation scheme is set to `cellPointFace`, i. e., the cells are decomposed into tetrahedra, one of the tetrahedra vertices coinciding with a face centre, which inherits field values by conventional interpolation schemes using values at the centres of cells that the face intersects.

1D case (Fig. 3.10a) The first observation is that rhopSonicFoam performs best of all solvers: it does not exhibit any overshoots and only little numerical dissipation[8].

[8]Numerical *dissipation* (or artificial viscosity) is the diffusive behaviour of a numerical solution that is purely numerical in origin; it is usually caused by even-order derivatives ($\partial^2 u/\partial x^2$, $\partial^4 u/\partial x^4$, etc.) as the leading term of the truncation error and could be described as "smearing out" the solution.

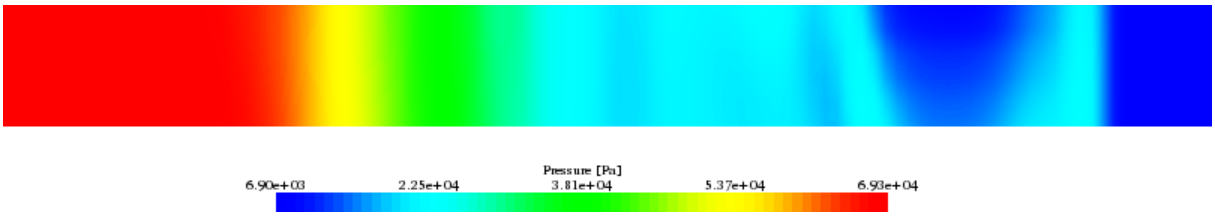


Figure 3.11: Pressure distribution at $t = 2.5 \cdot 10^{-4}$ s for the rhoSonicFoam solver. Instead of the expected constant pressure in y-direction (vertical axis in the picture), seemingly random variations occur.

sonicFoam has no overshoots either, but clearly the most numerical dissipation of the three laminar solvers. rhoSonicFoam lies in between in that respect but seems to be a victim of numerical dispersion[9], as the wiggles in the neighbourhood of large gradients indicate.

Of the turbulent solvers, rhoTurbFoam has the largest deviations in the expansion wave on the one hand and the largest overshoot at the shock wave on the other hand. rhoSonicFoam finally exhibits some numerical dissipation, but less so than other solvers.

2D case (Fig. 3.10b) Because the solution should not show variations of any variable in y-direction, no big differences to the 1D case are to be expected. In fact, upon closer inspection, the behaviour is qualitatively the same for all solvers, not yielding any new insights.

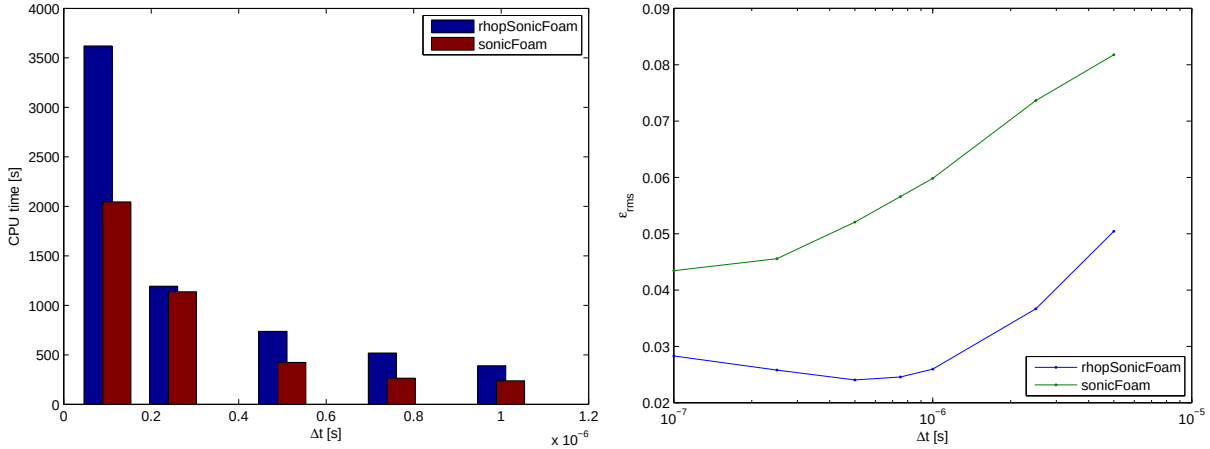
Axi-symmetric case (Fig. 3.10c) Also here, rhopSonicFoam performs best and sonicFoam shows some numerical dissipation. The two turbulent solvers perform exactly identically, smearing out the solution in the expansion wave and showing a small overshoot at the shock wave. rhoSonicFoam however deviates significantly from the analytical solution, exposing very large oscillations between expansion and shock wave. A look at the pressure distribution over the whole wedge in Fig. 3.11 shows that pressure varies in radial direction, which should not be the case.

It is decided at this point to exclude rhoSonicFoam from further investigations and focus on the better performing rhopSonicFoam and sonicFoam, as far as laminar solvers are concerned.

3D case (Fig. 3.10d) The final test on the 3D mesh shown in Fig. 3.9 confirms the findings so far. rhopSonicFoam is still the best in class, and only rhoTurbFoam shows some numerical dispersion. Artificial viscosity is more or less pronounced, but very similar for the three solvers other than rhopSonicFoam.

As a last remark, it can be stated that sonicTurbFoam calculates the pressure between expansions and shock wave to a value that is too little, but not by much; also density and Mach number plots in Section C.1 show these inaccurracies.

[9]Numerical *dispersion* on the other hand is caused by odd-order leading terms ($\partial^3 u/\partial x^3$, etc.); distortion of the propagation of different phases of a wave shows up as "wiggles" in front of and behind the wave. For details, see Anderson (1995, Section 6.6).



(a) CPU time required **(b)** Error behaviour

Figure 3.12: Temporal convergence and CPU time requirements for the laminar solvers.

3.1.4 Temporal convergence

In this subsection, we investigate the temporal convergence behaviour of the four solvers left, i. e., we want to observe how the accuracy is improved when the time step is reduced. "Error" is defined as follows: we take the same 100 pressure samples $p_1, p_2, \ldots, p_{100}$ with the sample utility as described above. These values are then compared to the exact solutions $p_{\mathrm{ex},i}$ and the difference is set relative to a reference pressure for the problem, $p_{\mathrm{ref}} = 3 \cdot 10^4\,\mathrm{Pa}$. Of these relative errors

$$\{\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_{100}\} = \{(p_1 - p_{\mathrm{ex},1})/p_{\mathrm{ref}}, (p_2 - p_{\mathrm{ex},2})/p_{\mathrm{ref}}, \ldots, (p_{100} - p_{\mathrm{ex},100})/p_{\mathrm{ref}}\}$$

the root mean square (RMS) error $\varepsilon_{\mathrm{rms}}$ is calculated. It is defined

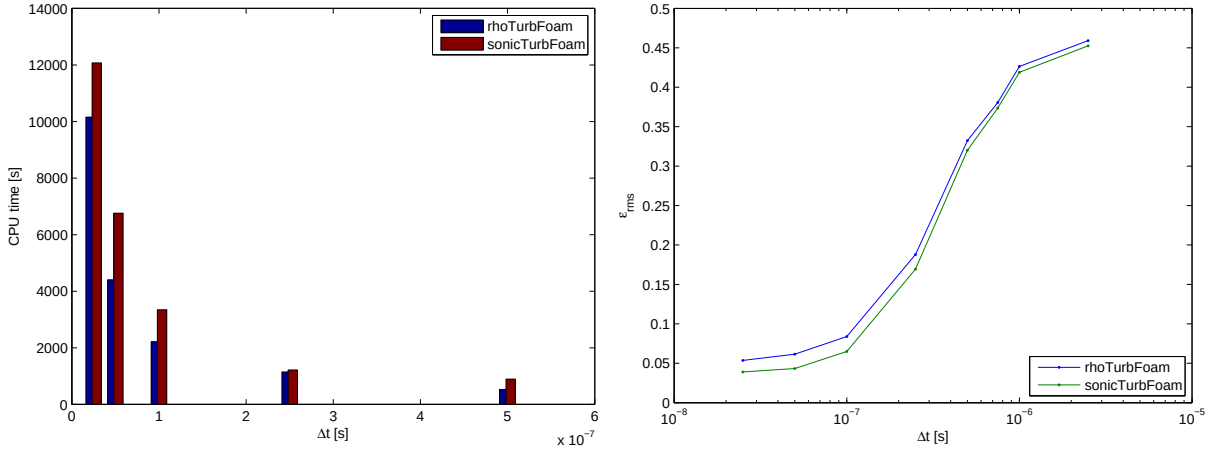
$$\varepsilon_{\mathrm{rms}} := \sqrt{\frac{1}{100}\sum_{i=1}^{100} \bar{p}_i^2} = \sqrt{\frac{\bar{p}_1^2 + \bar{p}_2^2 + \cdots + \bar{p}_{100}^2}{100}}$$

In addition, the CPU times required are recorded and compared in consideration of a possible trade-off between accuracy and CPU time required.

Because with some of the time steps used, no intermediate result is available for $t = 2.5{\cdot}10^{-4}\,\mathrm{s}$, the comparisons are made for $t = 3{\cdot}10^{-4}\,\mathrm{s}$. Also, we confine ourselves to the 3D mesh from now on, as it is the most interesting regarding more realistic applications.

Laminar solvers The results for the laminar solvers are shown in Fig. 3.12. First of all, it can be said that the good performance of rhopSonicFoam comes with a price tag: Fig. 3.12a shows that it constantly takes more time than sonicFoam, up to 80% more for the smallest time step of $\Delta t = 10^{-7}\,\mathrm{s}$. Also, the CPU time of nearly one hour for this relatively simple problem on a moderately large mesh seems quite long.

On the other hand, the relative error of rhopSonicFoam as seen in Fig. 3.12b is twice as small as the one of sonicFoam. An interesting observation is that for rhopSonicFoam,



(a) CPU time required **(b)** Error behaviour

Figure 3.13: Temporal convergence and CPU time requirements for the turbulent solvers.

the error has a minimum for $\Delta t = 0.5 \cdot 10^{-6}$ s; for smaller values of Δt, the error increases again. A possible explanation for this phenomenon could be that although it is imperative to have a Courant number equal to or less than unity for stability, $\mathrm{Co} \leq 1$, the accuracy can suffer if the time step is chosen very small, $\Delta t \ll \Delta t_{\mathrm{Co}=1}$, where $\Delta t_{\mathrm{Co}=1}$ is the time step for which exactly $\mathrm{Co} = 1$. Details can be found in Anderson (1995, Section 4.5).

Turbulent solvers Figure 3.13 shows the corresponding results for the turbulent solvers. Again, one of the solvers (sonicTurbFoam) requires clearly more CPU time than the other, but not as pronounced as with the laminar solvers. In general, the turbulent solvers require a much smaller time step to produce a meaningful solution at all, so the different Δt investigated lead to rather time-consuming runs: around three hours for $\Delta t = 0.25 \cdot 10^{-7}$ s, the smallest time step shown in Fig. 3.13a.

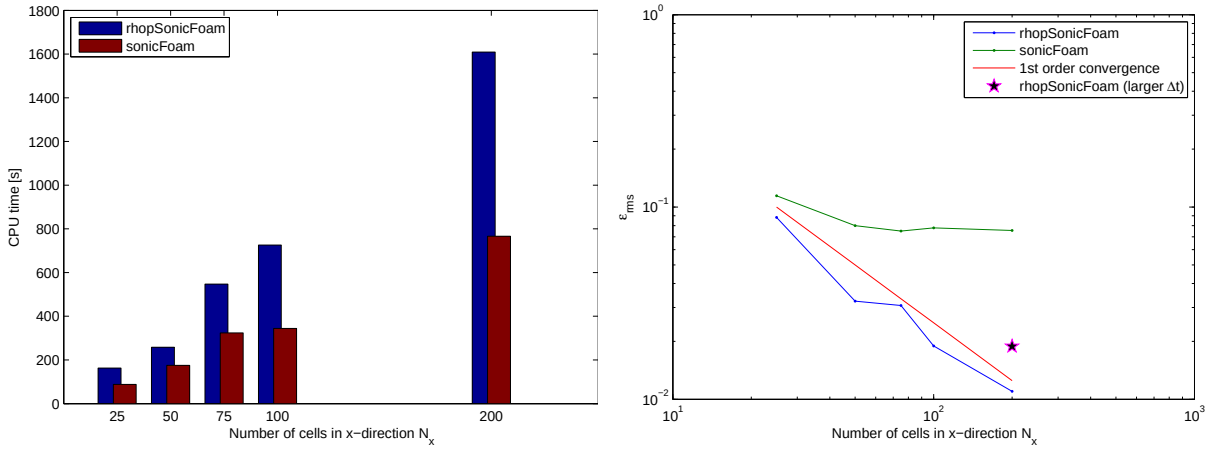
The necessity of setting Δt to a very small value is also reflected in the error plot, Fig. 3.13b: for runs where $\Delta > 10^{-7}$ s, the relative error is 15% or more. It is increased to around 45% for $\Delta t = 0.25 \cdot 10^{-5}$ s, while the laminar solvers have errors of only 7% and 3.5% for this time step size.

A second observation is that sonicTurbFoam performs constantly better, but only by very little. It is to expect that this will be more pronounced for cases where actual supersonic flows occur, see for example, Subsection 3.2.3.

3.1.5 Spatial convergence

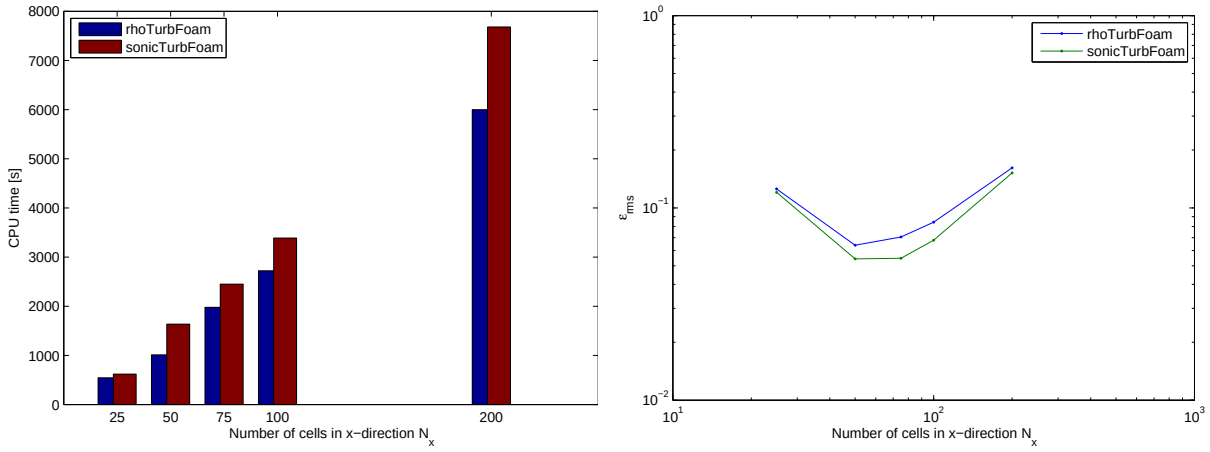
As a final stage of evaluating the solvers for the shock tube case, spatial convergence is examined. The number of cells in x-direction—always 100 so far—is varied between 25 and 200, always using the same (small) time step. The error is measured at $t = 2.5 \cdot 10^{-4}$ s.

Laminar solvers For the laminar solvers, this small time step is $\Delta t = 0.5 \cdot 10^{-6}$ s (the time step for which the error is minimal for rhopSonicFoam). Figure 3.14a confirms the



(a) CPU time required **(b)** Error behaviour

Figure 3.14: Spatial convergence and CPU time requirements for the laminar solvers.



(a) CPU time required **(b)** Error behaviour

Figure 3.15: Spatial convergence and CPU time requirements for the turbulent solvers.

findings of the previous subsection: rhopSonicFoam takes roughly twice as much time as sonicFoam and CPU time requirements grow approximately proportional to the mesh size. In turn, the mesh convergence rate of rhopSonicFoam as seen in Fig. 3.14b is much better (roughly first order convergence).

In addition to the mesh convergence tests with $\Delta t = 0.5 \cdot 10^{-6}\,\mathrm{s}$, we carry out an additional run with rhopSonicFoam, using 200 cells in x-direction, but a larger time step, namely $10^{-6}\,\mathrm{s}$. The result is indicated by the star in the plot. It shows that rhopSonicFoam, although using a time step twice as large as sonicFoam, still performs much better; and this with a CPU time of only 870 seconds, thus roughly the same as sonicFoam required for the finest grid.

Turbulent solvers For the turbulent solvers, Δt is set to 10^{-7} s. As before, computations for the turbulent solvers require more time (see Fig. 3.15a). The accuracy decreases with an increasing number of cells from 50 cells on again such that the result for 200 cells is as bad as for 25 cells (Fig. 3.15b).

3.1.6 Algorithm analysis

The claim that algorithm implementations in OpenFOAM are so readable that they serve as their own best documentation is somewhat bold. The example mentioned everywhere (Eq. (2.1) in this thesis) paints a picture that is a little too promising; in the actual solvers used in this thesis, for example rhopSonicFoam, there is definitely no obvious one-to-one correspondence of every expression used in the code to a term in one of the constituting equations. The complete absence of comments is not helpful, either.

In the list of OpenFOAM solvers in the OpenFOAM Wiki, source code of some of the solvers is commented extensively, for example for icoFoam[10]. Not for our compressible solvers, though.

What *can* be said about the transient solvers is that they all use some flavour of PISO, which stands for Pressure Implicit with Splitting of Operators. It is a predictor–corrector[11] method much like the popular SIMPLE (Semi-Implicit Method for Pressure-Linked Equations); both PISO and SIMPLE can be used for steady-state and transient problems, whereas SIMPLE was originally designed for steady-state flows and PISO for transient flows.

Transient PISO performs, simply speaking, at every time step the steady-state PISO algorithm with some extra terms in the pressure and momentum equations. Steady-state PISO (for 2D) can be described as follows:

- Starting from an initial guess p^*, u^* and v^*, some steps of the SIMPLE algorithm are performed, namely
  - Solving the discretised momentum equations
  - Solving the so called pressure correction equation
  - Correcting pressure and velocities
- PISO now solves a *second* pressure correction equation
- Pressure and velocities are corrected
- Once the corrections are smaller than a certain threshold, the results are taken as the initial guess of the next time step. Otherwise, the procedure is repeated on the same time level.

[10] http://openfoamwiki.net/index.php/IcoFoam

[11] Starting from a guessed pressure field p^*, a correction p' is defined as the difference to the correct pressure field p; similarly, velocity corrections are defined. The correct velocity field satisfies the continuity equation, the guessed field does not. In an iterative procedure, pressure and velocity fields are updated, until the continuity equation is satisfied.

SIMPLE, PISO and variations of these algorithms suited for particular applications are for example described in Versteeg & Malalasekera (1995).

OpenFOAM allows to specify even more corrector steps and also corrections for non-orthogonality of the mesh, see Subsection 2.3.3.

Getting to *really* understand what exactly is implemented in the OpenFOAM solvers would be an effort so large that it is decided to shelve this plan and focus on the performance of OpenFOAM in the more challenging and time-consuming test cases.

3.1.7 Comparison to CFD-ACE+

To compare the performance of OpenFOAM to a commercially available product, the shock tube problem is solved using CFD-ACE+[12], a solver widely employed within ABB. The fluid dynamics module of CFD-ACE+ is based on a pressure based FV method formulation Navier–Stokes equation flow solver and, according to the manufacturer, suitable to model "almost any gas or liquid system". Like OpenFOAM, general unstructured meshes with arbitrary mesh interfaces are supported; differencing schemes include upwind (1st and 2nd order), central, 2nd order limiter, 3rd order and smart schemes.

Earlier work dealing with ABB nozzle flows such as Mantilla Florez (2007) relies on CFD-ACE+ for their simulations, so this makes comparison of the results described in Chapter 5 to earlier findings possible.

For the shock tube problem, two meshes are created: a 2D axi-symmetric one and a 3D one. Like the meshes used in Subsection 3.1.3, they have 100 cells in x-direction. For both meshes, the solver is run using first order spatial differencing first, then second order. The results are then examined at $t = 0.3\,\text{ms}$. Because the first order run on the 3D mesh diverges, a snapshot at $t = 0.12\,\text{ms}$ is taken and compared to the corresponding analytical solution.

The result is shown in Fig. 3.16. Apart from diverging later on, the first order/3D mesh result also exhibits large oscillations between around the centre of the expansion fan and the shock wave travelling to the right. In a similar way, the first order solution on the axi-symmetric mesh features behaviour where in that same region the result is quite simply wrong.

The second order solutions are far better. There is no visible difference between the two grids, some artificial viscosity effects at the beginning and the end of the expansion fan and weak oscillations next to the shock wave. Table 3.4 shows the RMS relative errors (obtained as described in Subsection 3.1.4) for 100 samples in the axi-symmetric case and 370 samples in the 3D case. Clearly, the first order spatial discretisation results are useless, while second order results are quite accurate. The best OpenFOAM solver however, rhopSonicFoam, performs better with half as many cells in x-direction already, and with the identical mesh settings (100 cells in x-direction), the RMS relative error is only little more than one third of the CFD-ACE+ value.

3.1.8 Insights gained

The main findings of this section can be summarised as follows:

[12] See http://www.cfdrc.com/serv_prod/cfd_multiphysics/software/ace/

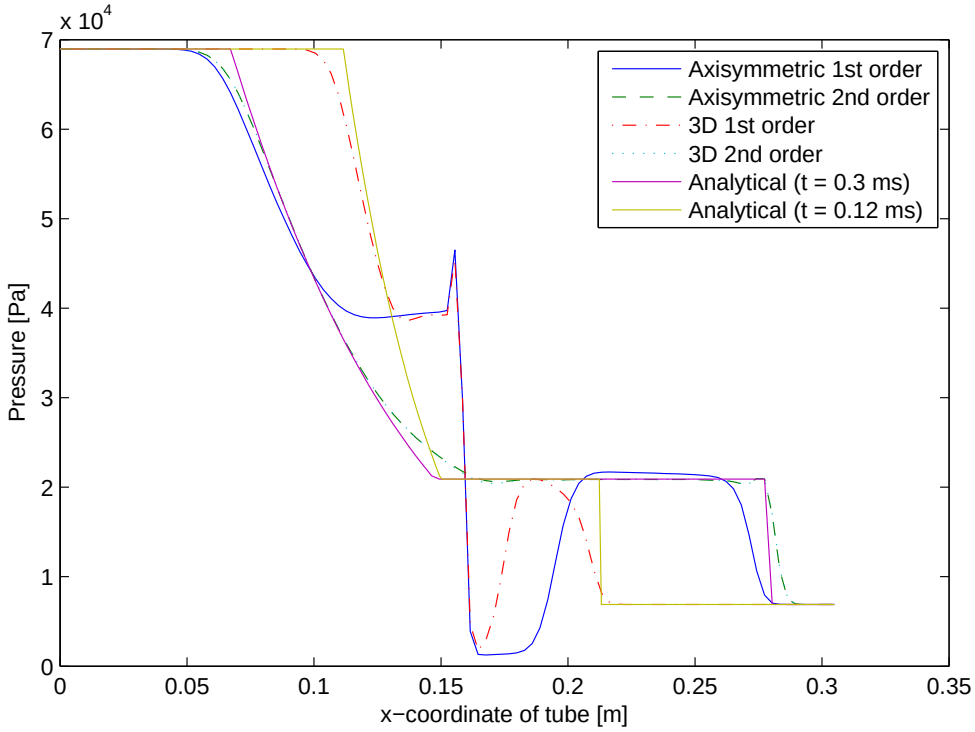


Figure 3.16: Results of CFD-ACE+ computations compared to analytical solution.

Solver	Grid/differencing	RMS relative error
CFD-ACE+	Axi-symmetric, 1st order	28.7%
CFD-ACE+	Axi-symmetric, 2nd order	4.5%
CFD-ACE+	3D, 1st order	19.6%
CFD-ACE+	3D, 2nd order	4.6%
rhopSonicFoam	3D (50 cells)	3.2%
rhopSonicFoam	3D (100 cells)	1.9%

Table 3.4: Quantitative comparison of CFD-ACE+ solver and rhopSonicFoam.

- All the solvers we looked at are capable of capturing the main features of the shock tube problem.
- rhopSonicFoam has the highest accuracy of all solvers for this problem.
- rhoSonicFoam struggles with the axi-symmetric problem and is excluded from further investigation.
- The turbulent solvers require significantly smaller time steps than the laminar solvers to get the error down to a reasonable value; the reason for this is probably that the initial flow configuration has extreme space gradients at the border.
- The better performing solver per category (rhopSonicFoam and sonicTurbFoam) requires more CPU time than the worse one under otherwise identical conditions.

- The differences between the laminar solvers are larger than between the turbulent solvers. However, when Δt of rhopSonicFoam is increased such that it is as fast as sonicFoam, its precision is still much better.
- OpenFOAM, or at least its best performing solver for this case, rhopSonicFoam, compares favourably to the commercial software CFD-ACE+, outperforming it by a factor of almost 3.

As the differences between the two turbulent solvers are only small so far, it is decided to carry them both along to see whether they differ more clearly when applied to other flows (e. g., supersonic ones).

3.2 The supersonic wedge problem

The second verification case is the *supersonic wedge problem*, which will be addressed in this section. After having treated a basically one-dimensional problem in Section 3.1, a two-dimensional problem is now being tackled.

We will proceed in a similar manner as in the previous section: in Subsection 3.2.1, the wedge problem is described as a case of an oblique shock wave and its relevance is pointed out. Subsection 3.2.2 describes how to obtain an analytical solution using the so called θ-β-Ma relation and how this solution is implemented as a MATLAB function. Subsection 3.2.3 evaluates the qualitative behaviour of the four relevant solvers, and in Subsection 3.2.4, the reduction of the error when refining the grid is looked at. In Subsection 3.2.5, the results are compared to the commercial package CFD-ACE+. Subsection 3.2.6 finally summarises the results and lessons learned.

3.2.1 Description and relevance

While the shock tube problem can be represented by one-dimensional Euler equations, the supersonic wedge problem involves two dimensions. The situation is as shown in Fig. 3.17a: a horizontal stream with $\mathrm{Ma} > 1$ encounters a wedge whose axis of symmetry is parallel to the flow direction. The supersonic flow is then "turned into itself", resulting in an oblique shock wave, after which the flow is again parallel and uniform.

Depending on the deflection angle θ, two different cases occur:

- $\theta < \theta_{\max}$: The shock is *attached* and a *straight line* (Fig. 3.17a).
- $\theta > \theta_{\max}$: The shock is *detached* and *bow-shaped* (Fig. 3.17b).

The value of $\theta_{\max}$ will be derived in Subsection 3.2.2. For the attached shock, there are again two different cases: the *strong shock solution* and the *weak shock solution*, where the shock wave angle β is larger for the strong shock than for the weak shock. Again, the derivation is postponed to Subsection 3.2.2.

Because the analytical treatment of the attached shock is much simpler than treatment of the detached shock, we focus on the first case; in particular, the weak shock solution, which usually occurs in nature, is analysed.

Oblique shock waves are prevalent in the study of super- and hypersonic flows, so it is crucial that a compressible flow solver is able to handle them well.

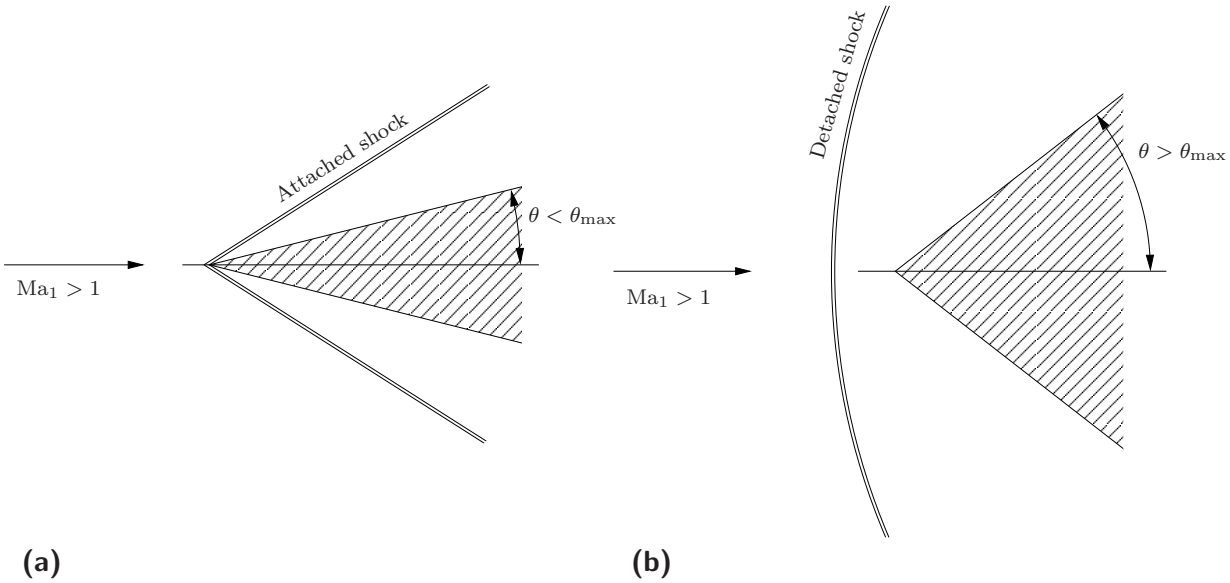


Figure 3.17: Supersonic wedge flows: (a) attached shock; (b) detached shock.

Setup used for testing The test cases are again modelled after Slater (2005); the setup is depicted in Fig. 3.18. Because of the symmetry of the problem, only the upper half of the wedge is being looked at.

The free stream Mach number is $\mathrm{Ma}_\infty = 2.5$, the temperature at the inlet is $T_\mathrm{i} = 288.9\,\mathrm{K}$ and the pressure at the inlet is $p_\mathrm{i} = 101.35\,\mathrm{kPa}$. The fluid is treated as a perfect gas with $\gamma = 1.4$ (dry air). Using the definition of the Mach number, $\mathrm{Ma} = u/a$, and because $a = \sqrt{\gamma RT}$ (with the specific gas constant for dry air, $R = 287.05\,\mathrm{J/(kg \cdot K)}$) we obtain the inlet velocity $u_\mathrm{i} = 851.84\,\mathrm{m/s}$. Again, these values look less random when converted to U. S. customary units, since $288.9\,\mathrm{K} = 520.0\,^\circ\mathrm{R}$ and $101.35\,\mathrm{kPa} = 14.7\,\mathrm{psi}$.

The deflection angle θ is set to $15\,^\circ$.

3.2.2 Analytical solution

The derivation of oblique shock relations can be found in many standard fluid dynamics textbooks, e. g., Kundu & Cohen (2004) or Anderson (2003). Here, we follow the latter. Below, we describe how quantities change across an oblique shock; on page 38, the interdependence of Mach number, deflection and shock wave angle is dealt with, and on page 39, it is shown how to deal with that result mathematically; on page 40, the outline of a MATLAB function implementing the analytical solution is given.

Oblique shock relations Let the index 1 refer to quantities upstream of the shock and index 2 to those behind the shock. $\mathbf{u}_1$ is the upstream (horizontal) velocity with its components u_1 (parallel to the shock) and w_1 (perpendicular to the shock); the corresponding Mach number is Ma_1. The oblique shock makes an angle β with respect to $\mathbf{u}_1$, and the flow is deflected by the angle θ. Behind the shock, the velocity is $\mathbf{u}_2$ with the parallel and perpendicular components u_2 and w_2; the Mach number is Ma_2.

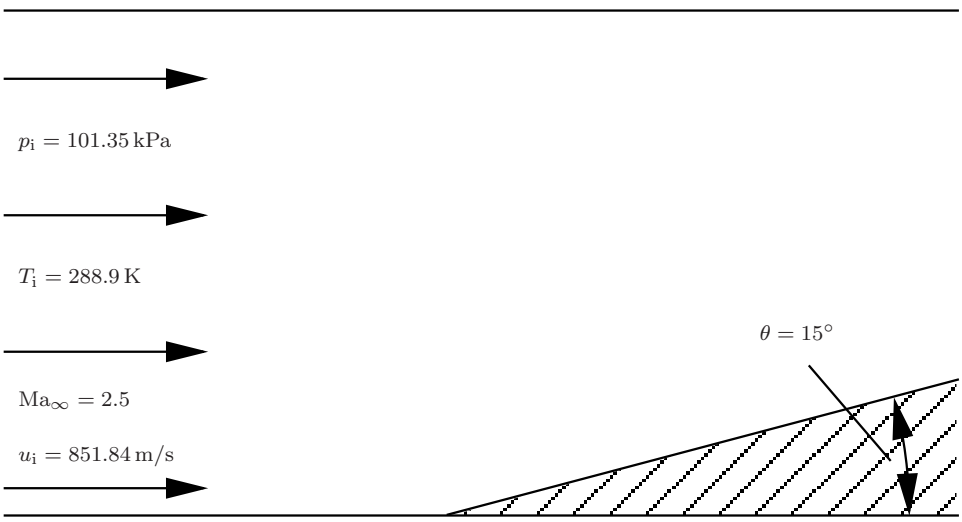


Figure 3.18: Setup for OpenFOAM evaluations of the supersonic wedge problem.

The normal and tangential Mach numbers corresponding to the parallel and perpendicular velocity components are called $\mathrm{Ma}_{\mathrm{n}_1}$ and $\mathrm{Ma}_{\mathrm{t}_1}$ ahead of the shock and $\mathrm{Ma}_{\mathrm{n}_2}$ and $\mathrm{Ma}_{\mathrm{t}_2}$ behind the shock. Figure 3.19 illustrates the nomenclature.

Applying the integral forms of the conservation equations to a control volume across the shock (see top of Fig. 3.19) yields, for the continuity equation, $-\rho_1 u_1 A_1 + \rho_2 u_2 A_2 = 0$ where $A_1 = A_2$ are the areas of the faces a and d, respectively. The other faces are parallel to the velocity and contribute nothing to the surface integral, so the oblique shock continuity equation is

$$\rho_1 u_1 = \rho_2 u_2 \tag{3.33}$$

The oblique shock momentum equation is split into tangential and normal components. For the tangential component, the faces a and d contribute nothing, and the components on b and f as well as c and e cancel each other out, so we get

$$(-\rho_1 u_1) w_1 + (\rho_2 u_2) w_2 = 0 \tag{3.34}$$

After division of Eq. (3.34) by (3.33), we have

$$w_1 = w_2$$

or, in words: *the tangential velocity component is preserved across an oblique shock wave.* For the normal component, we get

$$(-\rho_1 u_1) u_1 + (\rho_2 u_2) u_2 = -(-p_1 + p_2)$$

or

$$p_1 + \rho_1 u_1^2 = p_2 + \rho_2 u_2^2 \tag{3.35}$$

For the energy equation applied to the control volume in Fig. 3.19, we get

$$-(-p_1 u_1 + p_2 u_2) = -\rho_1 \left(e_1 + \frac{\mathbf{u}_1^2}{2} \right) u_1 + \rho_2 \left(e_2 + \frac{\mathbf{u}_2^2}{2} \right) u_2$$

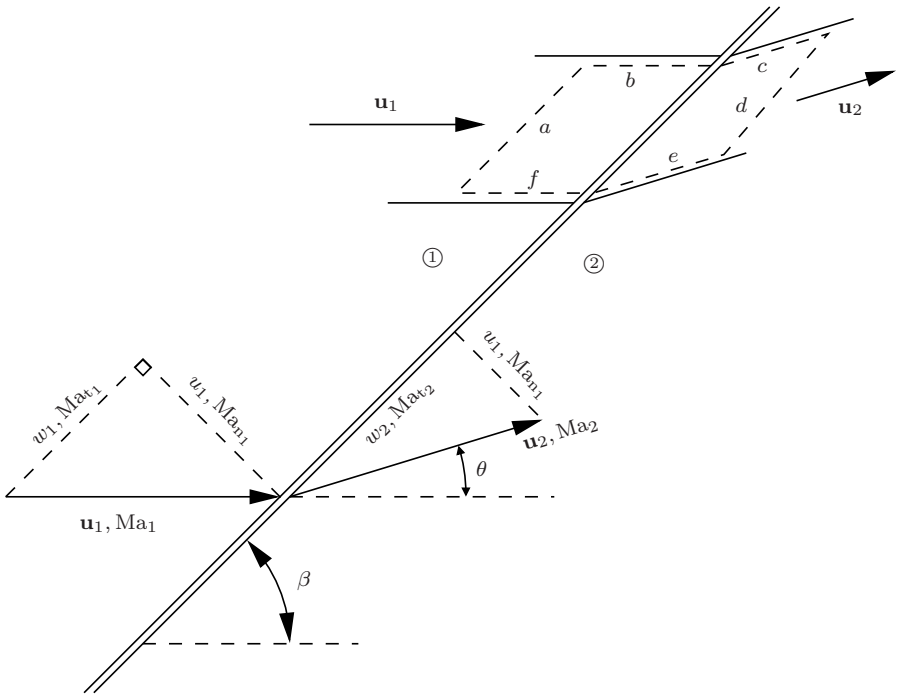


Figure 3.19: Oblique shock wave.

or

$$\left(h_1 + \frac{\mathbf{u}_1^2}{2}\right)\rho_1 u_1 = \left(h_2 + \frac{\mathbf{u}_2^2}{2}\right)\rho_2 u_2 \tag{3.36}$$

After division of Eq. (3.36) by (3.33), we get

$$h_1 + \frac{\mathbf{u}_1^2}{2} = h_2 + \frac{\mathbf{u}_2^2}{2} \tag{3.37}$$

Since $\mathbf{u}^2 = u^2 + w^2$ and $w_1 = w_2$, we have

$$\mathbf{u}_1^2 - \mathbf{u}_2^2 = (u_1^2 + w_1^2) - (u_2^2 + w_2^2) = u_1^2 - u_2^2$$

so Eq. (3.37) becomes

$$h_1 + \frac{u_1^2}{2} = h_2 + \frac{u_2^2}{2} \tag{3.38}$$

The Eqs. (3.33), (3.35) and (3.38) are exactly the normal shock wave relations, see Eqs. (3.1) to (3.3) on page 16. Consequently, the shock relations can be expressed as functions of the normal component of the upstream Mach number Ma_{n_1} where

$$\mathrm{Ma}_{n_1} = \mathrm{Ma}_1 \sin\beta \tag{3.39}$$

For a calorically perfect gas, this yields

$$\frac{\rho_2}{\rho_1} = \frac{(\gamma+1)\mathrm{Ma}_{n_1}^2}{(\gamma-1)\mathrm{Ma}_{n_1}^2+2} \tag{3.40}$$

$$\frac{p_2}{p_1} = 1 + \frac{2\gamma}{\gamma+1}(\mathrm{Ma}_{n_1}^2 - 1) \tag{3.41}$$

$$\mathrm{Ma}_{n_2}^2 = \frac{\mathrm{Ma}_{n_1}^2 + \big(2/(\gamma-1)\big)}{\big(2\gamma/(\gamma-1)\big)\mathrm{Ma}_{n_1}^2 - 1} \tag{3.42}$$

and

$$\frac{T_2}{T_1} = \frac{p_2}{p_1}\frac{\rho_1}{\rho_2} \tag{3.43}$$

The Mach number behind the shock, Ma_2, is

$$\mathrm{Ma}_2 = \frac{\mathrm{Ma}_{n_2}}{\sin(\beta-\theta)} \tag{3.44}$$

For $\beta = \pi/2$, the Eqs. (3.40) to (3.43) become the normal shock relations, which thus are just a special case of the oblique shock relations.

The θ-β-Ma relation All the oblique shock relations are functions of Ma_1 and the shock angle β. To find Ma_2 as given in Eq. (3.44) however, the flow deflection angle θ is required. θ is *also* a unique function of Ma_1 and β:

$$\tan\beta = \frac{u_1}{w_1} \tag{3.45}$$

and

$$\tan(\beta-\theta) = \frac{u_2}{w_2} \tag{3.46}$$

$$\tag{3.47}$$

Combining Eqs. (3.45) and (3.46) and using that $w_1 = w_2$, we get

$$\frac{\tan(\beta-\theta)}{\tan\beta} = \frac{u_2}{u_1} \tag{3.48}$$

Combining Eq. (3.48) with Eqs. (3.33), (3.39) and (3.40) yields

$$\frac{\tan(\beta-\theta)}{\tan\beta} = \frac{2+(\gamma-1)\mathrm{Ma}_1^2\sin^2\beta}{(\gamma+1)\mathrm{Ma}_1^2\sin^2\beta} \tag{3.49}$$

or, after some trigonometric manipulations,

$$\boxed{\tan\theta = 2\cot\beta\left(\frac{\mathrm{Ma}_1^2\sin^2\beta - 1}{\mathrm{Ma}_1^2(\gamma+\cos 2\beta)+2}\right)} \tag{3.50}$$

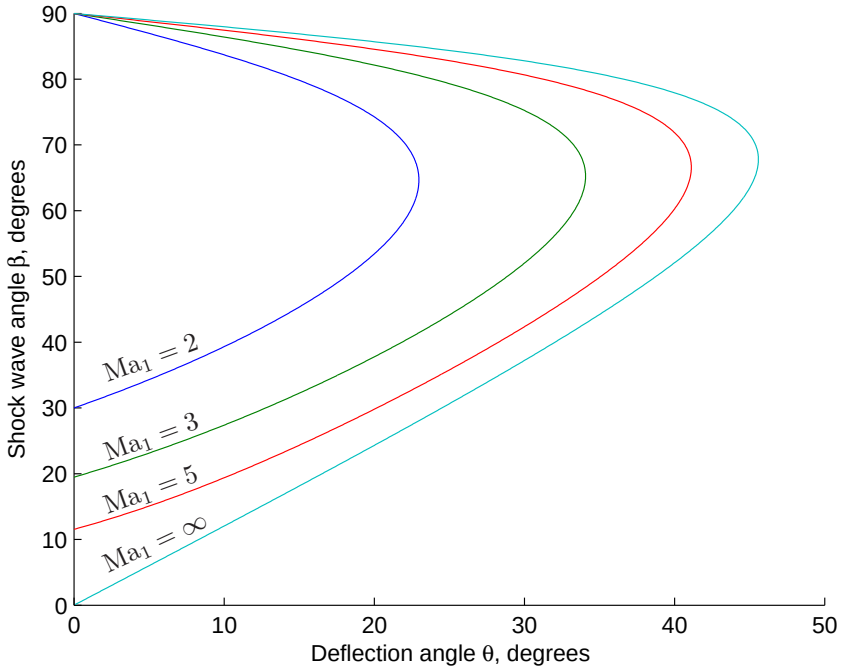


Figure 3.20: θ-β-Ma relation: deflection angle as a unique function of Mach number and shock wave angle.

Equation (3.50) is called the θ-β-*Ma relation*: it specifies θ uniquely as a function of Ma_1 and β. A plot showing some Mach isolines is given in Fig. 3.20.

The two possible cases now correspond to the weak/strong and detached shock cases mentioned in Subsection 3.2.1; $\theta_{\max}$ is, for a given Mach number, the value that corresponds to the rightmost point of the Mach isoline. For values of θ smaller than $\theta_{\max}$, two angles β are possible, the smaller being the weak shock case, the larger the strong shock case. If θ is larger than $\theta_{\max}$, the shock is detached.

An alternative form of the θ-β-Ma relation Often, one is interested in the weak shock angle β for a given deflection angle θ and upstream Mach number Ma_1. One method is to consult a large and precise diagram of the kind shown in Fig. 3.20, for example in Anderson (2003), which is of course very unhandy for automated lookup. Alternatively, Eq. (3.50) can be solved implicitly for β with the associated disadvantages of doing so.

The solution to the problem is to rewrite Eq. (3.50) as a cubic in $\tan\beta$:

$$\left(1+\frac{\gamma-1}{2}\mathrm{Ma}_1^2\right)\tan\theta\tan^3\beta-(\mathrm{Ma}_1^2-1)\tan^2\beta+\left(1+\frac{\gamma+1}{2}\mathrm{Ma}_1^2\right)\tan\theta\tan\beta+1=0 \tag{3.51}$$

Equation (3.51) has three different real roots for an attached shock with given Ma_1 and θ, of which one is negative. The positive roots correspond to the weak and strong shock

solutions and can be written

$$\boxed{\tan\beta = \frac{\mathrm{Ma}_1^2 - 1 + 2\lambda\cos\left((4\pi\delta + \cos^{-1}\chi)/3\right)}{3\left(1 + \frac{\gamma-1}{2}\mathrm{Ma}_1^2\right)\tan\theta}} \tag{3.52}$$

where for $\delta = 0$, we get the strong shock solution and for $\delta = 1$, we get the weak shock solution. λ and χ are

$$\lambda = \left((\mathrm{Ma}_1^2 - 1)^2 - 3\left(1 + \frac{\gamma-1}{2}\mathrm{Ma}_1^2\right)\left(1 + \frac{\gamma+1}{2}\mathrm{Ma}_1^2\right)\tan^2\theta\right)^{1/2} \tag{3.53}$$

and

$$\chi = \frac{(\mathrm{Ma}_1^2 - 1)^3 - 9\left(1 + \frac{\gamma-1}{2}\mathrm{Ma}_1^2\right)\left(1 + \frac{\gamma-1}{2}\mathrm{Ma}_1^2 + \frac{\gamma+1}{4}\mathrm{Ma}_1^4\right)\tan^2\theta}{\lambda^3} \tag{3.54}$$

With Eqs. (3.53) and (3.54) plugged into (3.52), we have an exact explicit formula for β when θ and Ma_1 are given.

Implementation in MATLAB The complete implementation of the analytical solution is given in Section B.2; here, we confine ourselves to a short description of the MATLAB function.

The function is called using

```
function [Ma,p,T,rho] = obliqueshock(x,y,Ma1,p1,T1,theta)
```

where `x` and `y` are the coordinates of the point in space for which Mach number `Ma`, pressure `p`, temperature `T` and density `rho` are to be computed. If none of the other input parameters are specified, they are set to the values given in Subsection 3.2.1. Alternatively, everything but the deflection angle `theta` can be specified, which sets `theta` to 15 degrees.

After checking the input, some constants are set and the density upstream of the shock is computed using the perfect gas relation, $\rho_1 = p_1/(R \cdot T_1)$. Also, the value of `theta` is made sure to be positive, because expansion waves cannot be treated by this function.

The core part of the function is the implementation of Eq. (3.52):

```
% Calculate and check shock wave angle using the beta-theta-Mach relation.
lambda = sqrt((Ma1^2 - 1)^2 - 3*(1 + (gamma-1)/2*Ma1^2)*...
    (1 + (gamma+1)/2*Ma1^2)*tan(theta)^2);
chi = ((Ma1^2 - 1)^3 - 9*(1 + (gamma-1)/2*Ma1^2)*...
    (1 + (gamma-1)/2*Ma1^2 + (gamma+1)/4*Ma1^4)*tan(theta)^2) / lambda^3;
beta = atan((Ma1^2 - 1 + 2*lambda*cos((4*pi + acos(chi))/3)) /...
    (3*(1 + (gamma-1)/2*Ma1^2)*tan(theta)));
if (beta < 0 || ~isreal(beta))
    error('The shock is detached, choose a smaller theta or a larger Ma1.')
end
```

By setting δ from Eq. (3.52) to $\delta = 1$, we ensure to obtain the weak shock solution, *unless β is negative or imaginary*, which would mean that the shock is detached. The `if` statement at the end of the above code block makes sure that this is not the case.

Now that the shock angle β is known, all that is left to be done is finding out where the coordinates in question are located: upstream of the shock, downstream of it, or in a position that makes no sense, i. e., *in* the wedge or in the lower half plane. The upstream case is trivial—the input values are returned as the result—, the downstream case returns the values obtained from Eqs. (3.39) to (3.44), and the last case throws an error.

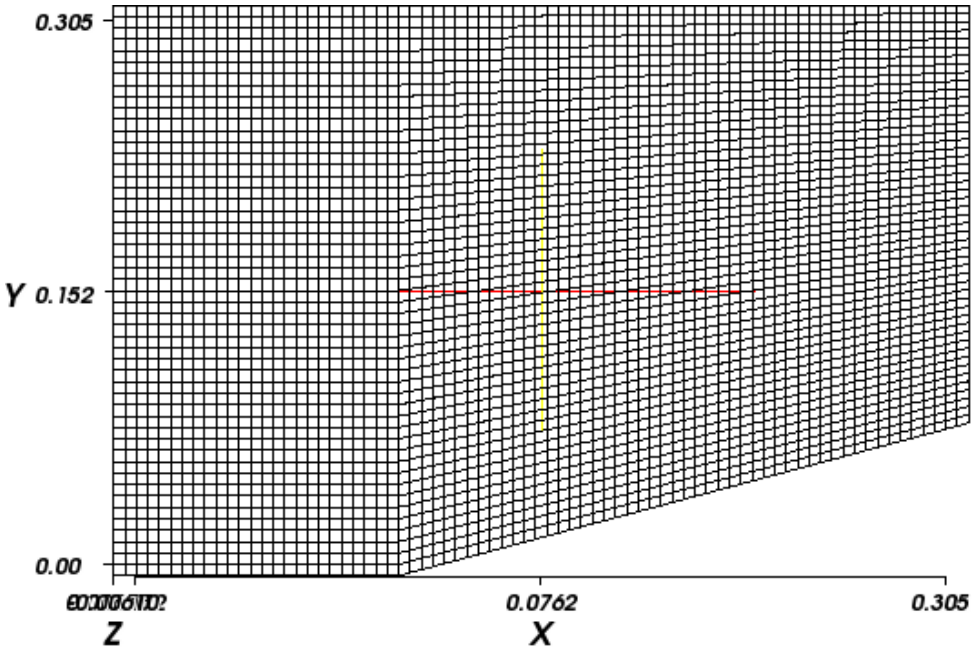


Figure 3.21: The 75×50 mesh used for solver quality evaluation with the supersonic wedge problem.

3.2.3 Solver quality evaluation

As in Subsection 3.1.3 for the shock tube, the different solvers are evaluated from a qualitative perspective first. Below, the mesh used for the computations is presented and an overview of the solver settings is given, and on the next page, the results for the different solvers are compared.

The wedge mesh For quality evaluation, a mesh of the size 75×50 is used (Fig. 3.21), where the rectangular part is of the size 25×50 volumes and the part above the wedge is 50×50. The spacing on the corresponding edges is uniform. As seen before, to solve a two-dimensional problem, a three-dimensional mesh has to provided, where the depth in z-direction is one volume only.

Solver setup For the four solvers under consideration, rhopSonicFoam, sonicFoam, rhoTurbFoam and sonicTurbFoam, the following settings are applied:

- The upper boundary and the horizontal part of the lower boundary are set to the symmetry type.
- The part of the lower boundary representing the wedge is set to the slip type[13].
- The outlet boundary has zeroGradient type for all quantities.
- The inlet boundary is of zeroGradient type for `epsilon` and `k`; for `p`, `T` and `U`, it is of fixedValue type with the values given in Subsection 3.2.1.

[13] I. e., zero gradient condition for scalars and a zero normal component for vectors.

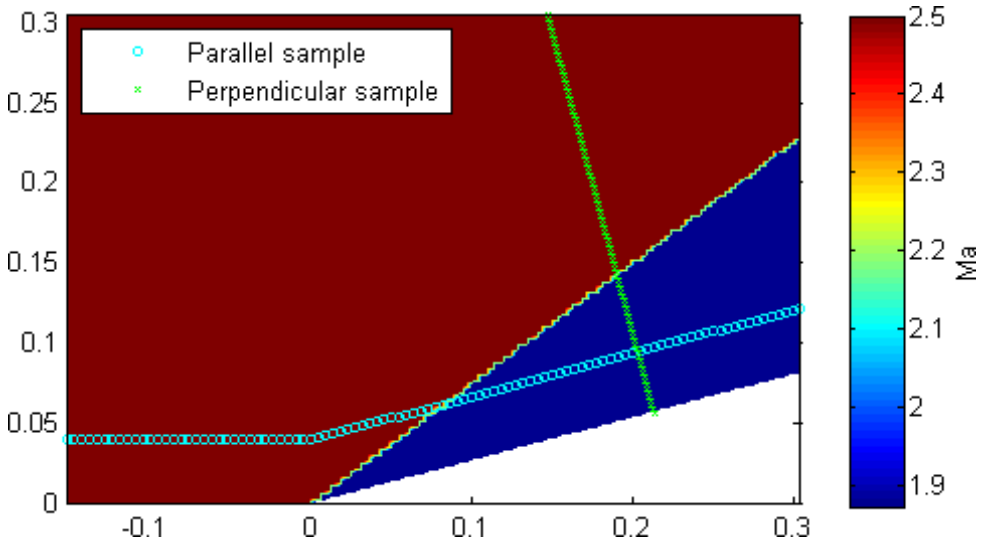


Figure 3.22: Analytical Mach number for the wedge problem, sampling locations.

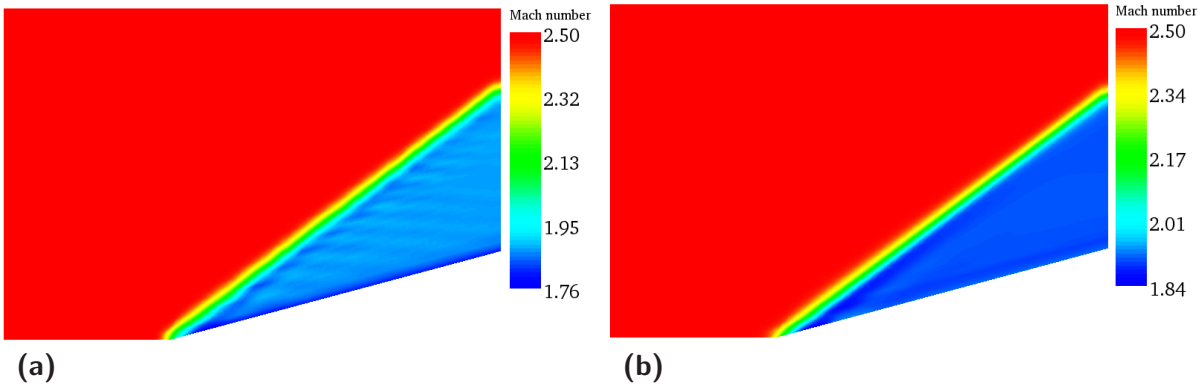


Figure 3.23: OpenFOAM Mach number results: (a) rhopSonicFoam at $t = 1.5\,\mathrm{ms}$; (b) sonicFoam at $t = 3\,\mathrm{ms}$.

- The flow field at time zero is set to the inlet values.

For later analysis, the locations for samples are determined: 100 samples are taken parallel to the lower boundary, 100 perpendicular to the wedge, as shown in Fig. 3.22, which also shows the Mach number field obtained using the MATLAB function described above.

Results In this paragraph, the Mach number fields for the laminar and turbulent solvers are evaluated.

Laminar solvers Figure 3.23 shows the Mach number fields for the two laminar solvers after convergence. rhopSonicFoam takes a bit longer to converge, but computation times are in the order of magnitude of a few minutes for both. The shock position and angle is captured well in both cases; rhopSonicFoam (Fig. 3.23a) however features some

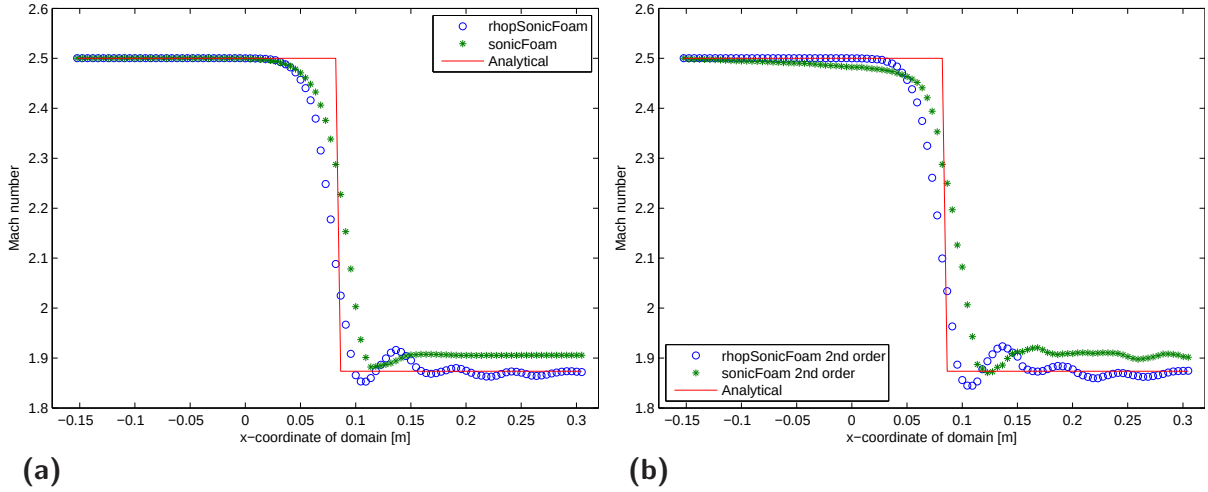


Figure 3.24: Comparison of laminar solver parallel samples to analytical solution: (a) 1st order spatial discretisation; (b) 2nd order spatial discretisation.

slight oscillations in the region downstream of the shock, whereas sonicFoam (Fig. 3.23b) seems to be smooth, just like theory predicts.

A closer look at the actual Mach values reveals that—although smoother—the sonicFoam solution's Mach numbers behind the shock are systematically too high. In Fig. 3.24a, it can be seen that both solvers smear out the solution a little around the shock, but while rhopSonicFoam oscillates around the correct value with a fading amplitude behind the shock, sonicFoam stays above the value predicted by theory.

This behaviour is confirmed by the samples taken perpendicularly to the wedge. The corresponding plots do not add much information, but for reasons of completeness, they are included in Section C.2 (Fig. C.5a).

To improve the behaviour of the solution in the neighbourhood of the shock, the selection of a more adequate spatial discretisation scheme can help. In OpenFOAM, these schemes are specified in the *fvSchemes* dictionary. Every type of terms has its own subdictionary, e. g., *gradSchemes* for gradient terms or *divSchemes* for divergence terms. Since almost all terms are, by default, set to some higher order scheme, we only change the divergence terms with 1st order methods to be treated with the MUSCL[14] approach, a method for the generation of second order upwind schemes via variable extrapolation; refer to Hirsch (1990, Section 21.1) for details.

However, this does not improve the results: the rhopSonicFoam solution is still oscillatory downstream of the shock, and the sonicFoam solution still predicts the Mach numbers too large behind the shock *plus* oscillates now also. In addition, the Mach number prediction upstream of the shock has worsened considerably. Figure C.5b in the appendix confirms these findings: it seems that the default settings are the better choice.

Turbulent solvers The two turbulent solvers, rhoTurbFoam and sonicTurbFoam, are designed for viscous flows, and the viscosity cannot simply be set to zero, so it is being

[14]Monotone Upstream-centred Scheme for Conservation Laws

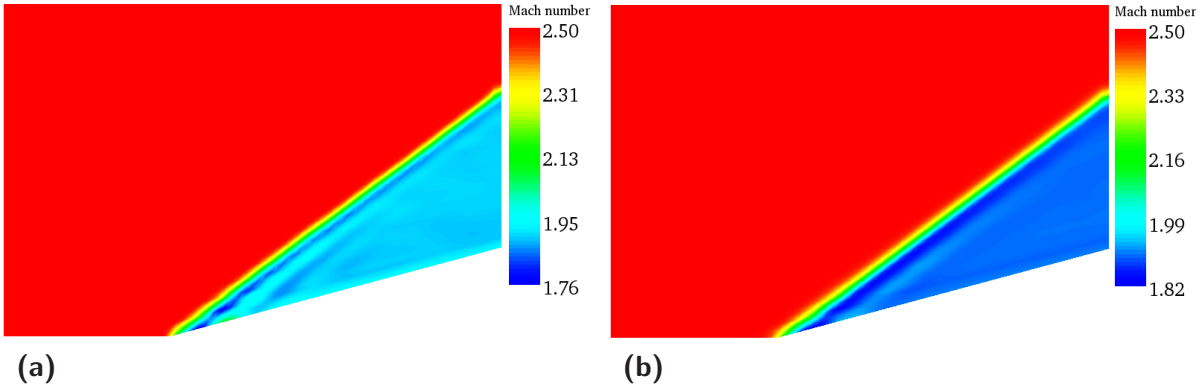


Figure 3.25: OpenFOAM Mach number results: (a) rhoTurbFoam at $t = 2\,\mathrm{ms}$; (b) sonicTurbFoam at $t = 2\,\mathrm{ms}$.

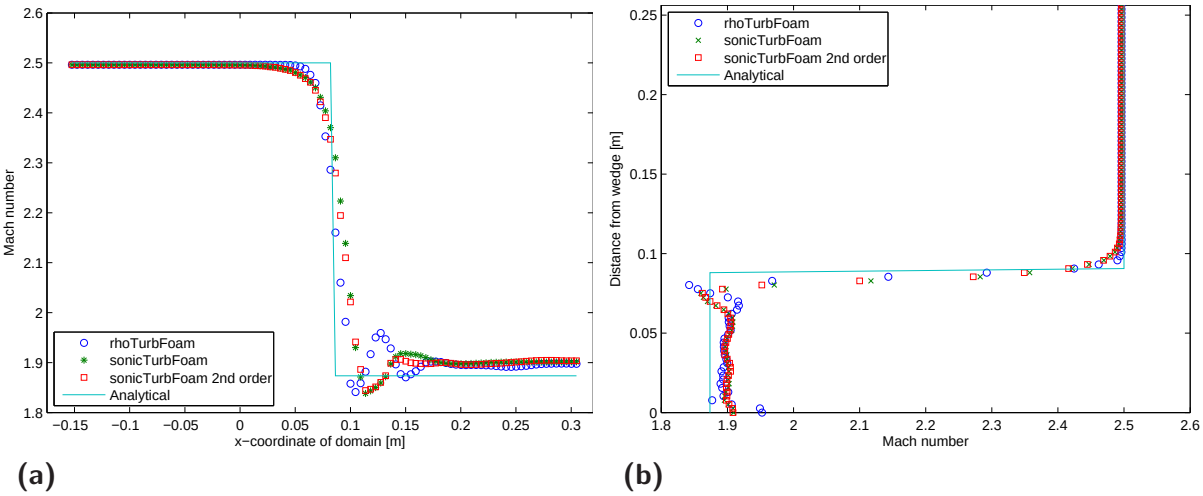


Figure 3.26: Comparison of turbulent solver samples to analytical solution: (a) parallel samples; (b) perpendicular samples.

kept at the value for air, $\mu = 1.84 \cdot 10^{-5}\,\mathrm{kg/(m \cdot s)}$.

The initial values for k and ε are estimated as in Subsection 3.1.3, based on a turbulence intensity of 5%. The resulting Mach number fields (convergence after 2 ms) are shown in Fig. 3.25.

Again, the shocks are at the right position with the correct angle. rhoTurbFoam, according to its description not explicitly designed for transonic or supersonic flows, exhibits quite a strong dip after the shock and a somewhat non-uniform Mach number field behind the shock in general. These shortcomings are likely to be a combination of viscosity effects and the inability to cope with shocks. sonicTurbFoam on the other hand, the turbulent solver designed for transonic and supersonic flows, features a much smoother Mach field behind the shock, yet still not perfectly so.

A look at the comparison of the sampled values to the analytical solutions in Fig. 3.26 confirms the suspected oscillations: rhoTurbFoam dives below the exact value, oscillates

strongly with an amplitude of roughly 20% of the shock height and comes at rest at a value slightly above the exact value. sonicTurbFoam oscillates as well, but reaches faster a constant value.

Overall, the turbulent solvers capture the shock more sharply than the laminar solvers, but oscillate and have a Mach number larger than predicted by theory behind the shock. An attempt at using exclusively second and higher order schemes with sonicTurbFoam (MUSCL leads to errors when used with rhoTurbFoam), shown by red squares in Fig. 3.26, results in only marginal improvement and is dropped as a remedy.

3.2.4 Spatial convergence

For a more quantitative way of comparing the solvers, a mesh convergence study is conducted. Five different meshes are used, of which the coarsest has half as many cells (38×25) as the original mesh from Subsection 3.2.3 and the finer meshes have 150×100, 300×200 and 600×400 volumes, respectively.

The source after which most verification and validation cases in this thesis are modelled, Slater (2005), uses the so called *grid convergence index* (GCI) for its order-of-accuracy studies. The GCI is defined in Roache (1994) as a means of uniform reporting of grid refinement studies in CFD; the method is based on an error estimation derived from the extrapolation method introduced in Richardson (1927). After considering to report grid refinement studies using the GCI, it is decided not do so, mainly because the main advantage of doing so is to be able to compare results achieved by different solvers for maybe different problems, while we are interested in comparing the performance of different OpenFOAM solvers for the same problem. Also, GCI results for the test cases done with CFD-ACE+ or other codes have not been recorded in earlier works such as Wolter (1997) or Mantilla Florez (2007).

Similar to Subsection 3.1.4, the RMS error is defined

$$\varepsilon_{\text{rms}} := \sqrt{\frac{1}{200}\sum_{i=1}^{200}\overline{\text{Ma}}_i^2} = \sqrt{\frac{\overline{\text{Ma}}_1^2 + \overline{\text{Ma}}_2^2 + \cdots + \overline{\text{Ma}}_{200}^2}{200}}$$

where $\overline{\text{Ma}}_1, \ldots, \overline{\text{Ma}}_{100}$ are the relative errors of the parallel samples and $\overline{\text{Ma}}_{101}, \ldots, \overline{\text{Ma}}_{200}$ the ones of the perpendicular samples. The relative errors are defined

$$\overline{\text{Ma}}_i = \frac{\text{Ma}_i - \text{Ma}_{\text{ex},i}}{\text{Ma}_{\text{ref}}}$$

where $\text{Ma}_{\text{ex},i}$ are the exact solutions and $\text{Ma}_{\text{ref}} = 2.5$, the inflow Mach number.

The resulting error behaviour is shown in Fig. 3.27. While for sonicFoam the mesh refinement does not effect any significant improvement—it seems we have reached grid independence—, rhopSonicFoam exhibits error convergence with an order of 0.5 (see below). For the turbulent solvers, increasing the mesh resolution *does* improve the error, but no convergence order can be seen. A look at the y-axis tick marks reveals that the improvements are almost negligible (in the order of magnitude of a few percent between the coarsest and the finest meshes).

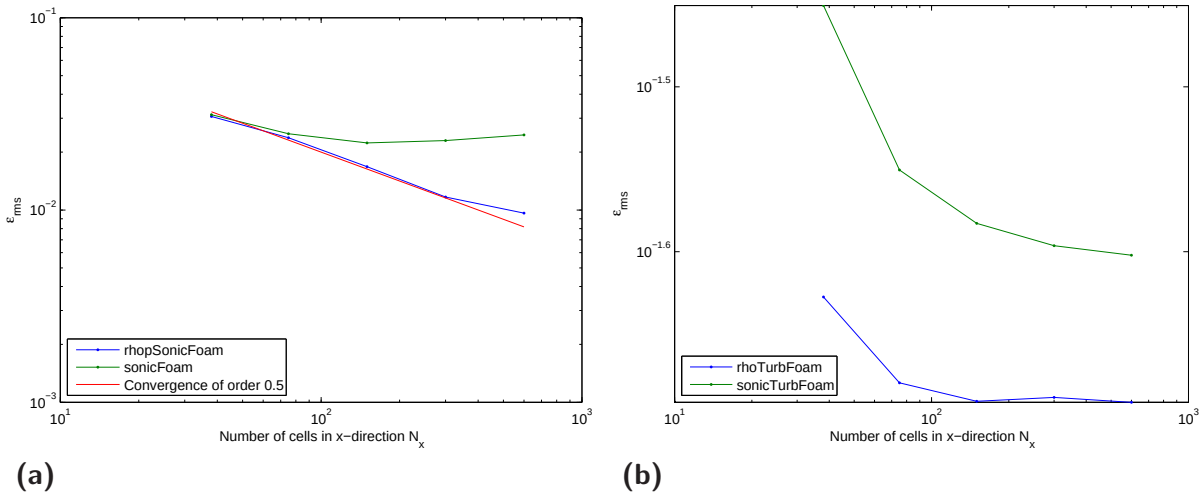


Figure 3.27: Mesh convergence for the wedge case: (a) laminar solvers; (b) turbulent solvers.

But why is the error convergence of rhopSonicFoam only 0.5? After the results in Subsection 3.1.5 and according to the settings in the *fvSchemes* dictionary, one would have expected it to be 1. A possible explanation is that the shock prevents the expected behaviour and that looking at this few points is too much of an arbitrary selection; in Slater (2005), only the values behind the shock are taken into account. We adapt the corresponding MATLAB scripts accordingly to only use values downstream and away from the shock: the samples shown in Fig. 3.28 are now chosen such that they cover more or less uniformly the region behind the shock.

In Fig. 3.29, the error plots for this new configuration are shown. On the one hand, rhopSonicFoam now exhibits 1st order convergence behaviour before reaching grid independence, as expected. On the other hand, only now it becomes obvious that the results are quite accurate: for rhopSonicFoam, the RMS error becomes smaller than 0.73%. sonicFoam seems to have reached grid independence for the coarsest configuration already.

The turbulent solvers seem to behave erratically, but the differences between the largest and smallest RMS error are even smaller than before: 0.9% for rhoTurbFoam and 0.83% for sonicTurbFoam, so also here, speaking of grid independence seems to be appropriate.

3.2.5 Comparison to CFD-ACE+

For comparison, three runs are performed in CFD-ACE+: two using a mesh with 150 cells in x-direction and one with 300 cells. Of the first two, one uses first order spatial discretisation, the other second order; the third case uses second order discretisation.

Figure 3.30 shows the parallel samples for these three runs. Overall, the agreement is very good; the first order solution smears out the shock a little, but switching to second order improves this, at the price of some oscillations behind the shock. The solution on the finer mesh is very similar to the coarse second order solution, just captures the shock more accurately.

In the region further behind the shock, no oscillations occur at all, and the Mach number estimation is very accurate, as opposed to some of the OpenFOAM solutions.

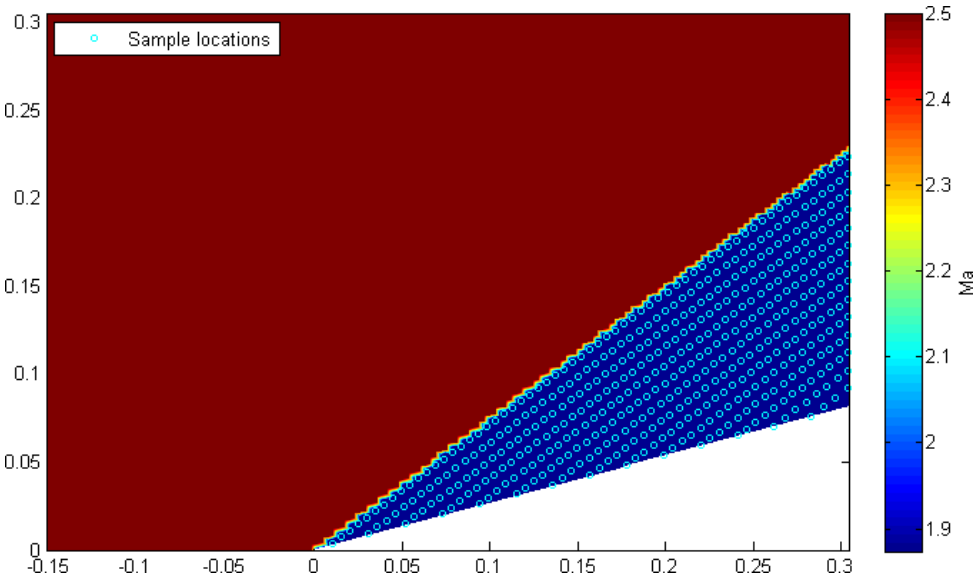


Figure 3.28: New sample locations for the wedge case.

3.2.6 Insights gained

The findings of the supersonic wedge investigations can be summarised as follows:

- None of the four solvers struggles fundamentally: all solutions converge to a steady state and the shock geometry is captured correctly.
- rhopSonicFoam is confirmed as the most accurate of the four solvers with a RMS error of less than one percent for the finer meshes.
- sonicFoam, although featuring a very smooth Mach number field behind the shock, deviates systematically from the analytical solution behind the shock.
- rhoTurbFoam, not designed for transonic or supersonic flows, exhibits the strongest oscillations of all solvers behind the shock.
- The turbulent solvers incorporate viscosity effects, whereas the analytical solution is based on the inviscid Euler equations, which probably accounts for the overall worse results of rhoTurbFoam and sonicTurbFoam.

Because of its less than optimal behaviour behind the shock, rhoTurbFoam is considered as unsuitable for this kind of flows and will not be included in the validation cases.

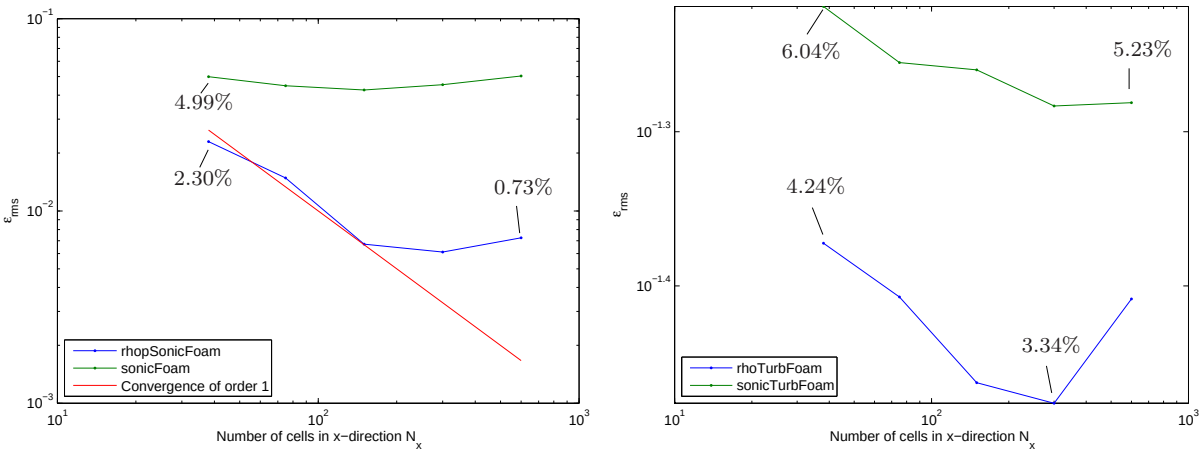


Figure 3.29: Mesh convergence for the wedge case, only samples downstream of the shock: (a) laminar solvers; (b) turbulent solvers.

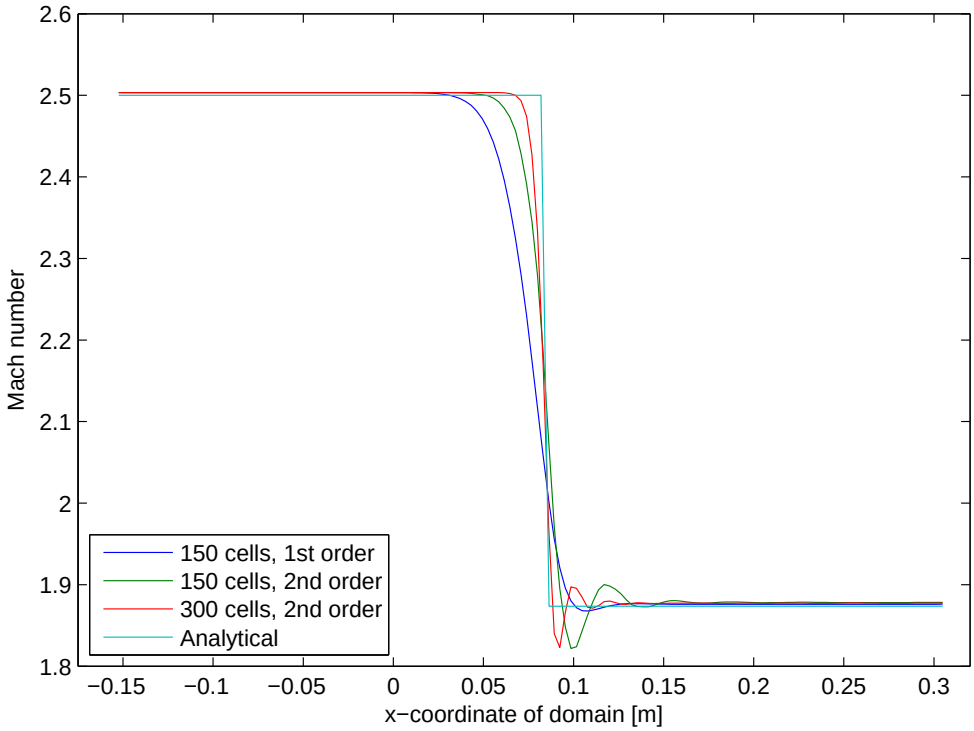


Figure 3.30: Parallel samples of three CFD-ACE+ runs for the supersonic wedge problem.

4 Validation cases

As opposed to verification, where comparison to an analytical solution is possible, the *validation cases* cannot be solved analytically. Simulation results have to be compared to experimentally obtained values (see Subsection 1.4.3).

4.1 The backward facing step problem

The *backward facing step problem* is the first validation case. In Subsection 4.1.1, the problem is described and its relevance pointed out; also, a short overview of the referenced literature is given. Subsection 4.1.2 shows simulation results and their comparison to the reference experiment and CFD-ACE+, and Subsection 4.1.3 summarises the findings of the section.

4.1.1 Description and relevance

The backward facing step problem cannot be solved analytically; because of its many interesting flow features, it is a popular test case for numerical schemes. In Slater (2005), tests are conducted for incompressible flow as well as compressible flow with supersonic incident flow; we focus on the compressible case here. Figure 4.1 shows the computational domain and the main features of the flow:

- Right after the step, an *expansion fan* is formed
- The flow *separates* after the step
- Behind the step, there is a *recirculation area*
- The end of the recirculation area is marked by a weak oblique *reattachment shock*

To evaluate the performance of a solver, interesting quantities to look at are the pressure field in general and, more specifically, the pressure along the elongation of the step as well as the location of the reattachment shock. In Slater (2005), the results are compared to Smith (1967). The results from this study come primarily in the shape of printed graphs, so an exact comparison to simulation results is not possible, but we still get an idea of how well OpenFOAM performs.

Setup used for testing In Smith (1967), three different geometries are used, differing by the step height. We only look at the configuration with the medium step height, $h = 11.25\,\text{mm}$. The other relevant dimensions[1] of the mesh shown in Fig. 4.2 are: distance from inlet to step $L_\text{i} = 0.1016\,\text{m}$, distance from step to outlet $L_\text{o} = 0.3048\,\text{m}$, and distance from step to upper boundary $L_\text{u} = 0.1475\,\text{m}$.

The block above the step has 30×40 volumes, the block to the right of it 60×40 volumes, and the block behind the step has 60×39 volumes, resulting in 5940 volumes in total.

[1] Again, these are round lots when using feet, e. g., the distance between inlet and step corresponds to exactly 4 ft.

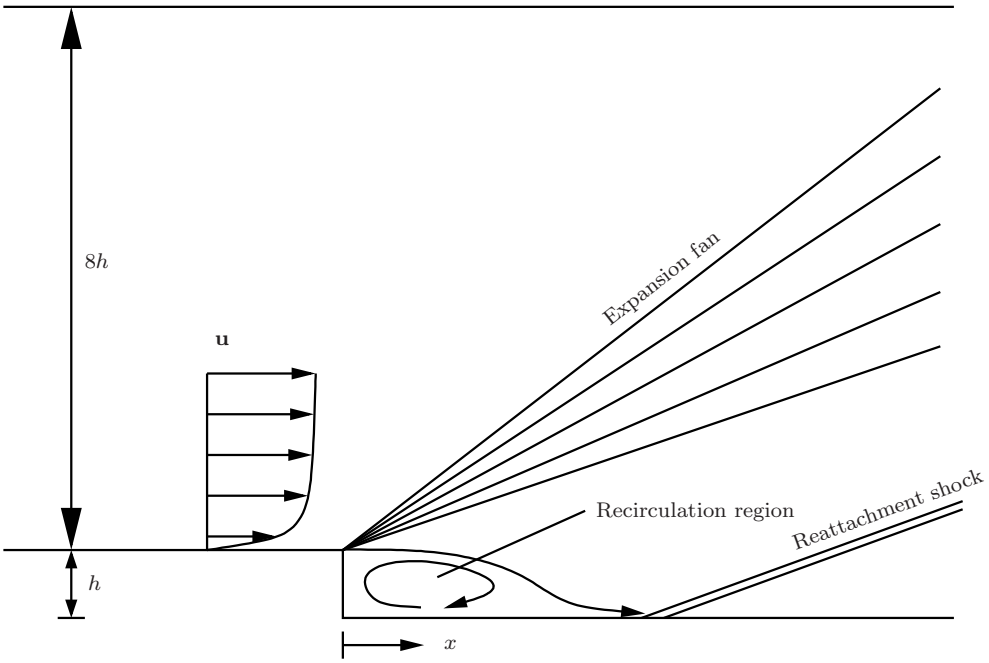


Figure 4.1: The backward facing step problem and its main flow features: expansion fan, separation/recirculation, reattachment shock.

The inflow Mach number is 2.5 Ma, which for the given static inflow pressure of $p_\mathrm{i} = 15.35\,\mathrm{kPa}$ equals a velocity of $u_\mathrm{i} = 651.9\,\mathrm{m/s}$. The temperature to the left of the step is initialised to 169.2 K, to the right of the step to 153 K; the velocity to the left of the step is set to the inflow velocity, to the right of and above the step to 619.9 m/s (results in continuous Mach number across the step) and behind the step to zero. The idea of setting the velocity to zero behind the step is to ensure proper development of the recirculation region.

The boundary conditions are as follows:

- Zero gradient for k and ε at all boundaries

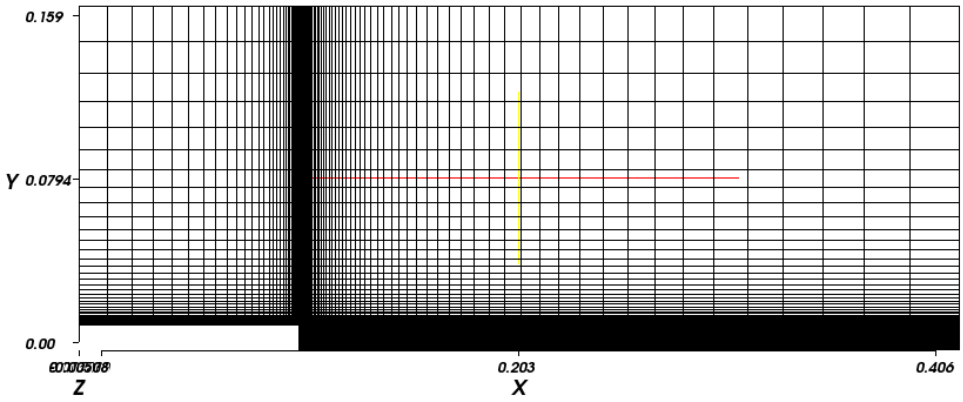


Figure 4.2: The mesh for the backward facing step case (5940 volumes).

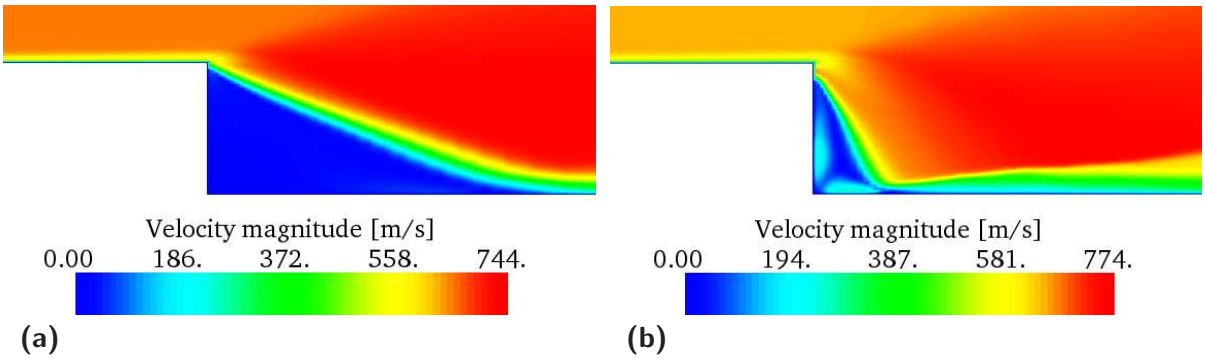


Figure 4.3: Velocity field in the neighbourhood of the step: (a) solution at $t = 1.5\,\mathrm{ms}$ (good agreement with experiment); (b) steady-state solution at $t = 3.5\,\mathrm{ms}$ (poor agreement with experiment).

- Fixed value $p = p_\mathrm{i} = 15.35\,\mathrm{kPa}$ at the inlet, zero gradient everywhere else for pressure
- Fixed value $T = T_\mathrm{i} = 169.2\,\mathrm{K}$ at the inlet, zero gradient everywhere else for temperature
- Fixed value $u = u_\mathrm{i} = 651.9\,\mathrm{m/s}$ at the inlet, no-slip walls at the lower boundary, slip wall at the upper boundary and zero gradient at the outlet for velocity

For turbulence, the standard k–ε model is chosen.

The settings in the *thermophysicalProperties* dictionary are, as in the supersonic wedge case, based on the values for air and the following choice of models:

- Constant thermodynamic coefficients, with a specific heat capacity $c_p = 1005\,\mathrm{J/(kg\cdot K)}$ and heat of fusion $H_\mathrm{f} = 2.544 \cdot 10^6\,\mathrm{J/kg}$
- Constant transport coefficients with the dynamic viscosity $\mu = 18.27 \cdot 10^{-6}\,\mathrm{kg/(m\cdot s)}$ and the Prandtl number $\mathrm{Pr} = c_p\mu/\kappa = 0.7$, where κ is the thermal conductivity, $\kappa = 0.0262\,\mathrm{W/(m\cdot K)}$.

The numerical schemes in the *fvSchemes* dictionary and the equation solvers, tolerances and algorithms in the *fvSolution* dictionary all remain at their default settings.

4.1.2 Solver quality evaluation

For the assessment of a first result, we focus on a basic aspect of the resulting flow field: the location of the reattachment shock, i. e., the length of the recirculation zone. According to the measurements in Smith (1967), this should take place at $\approx 2.4h$ once steady-state is reached. Since turbulence plays a vital role, the laminar solvers rhopSonicFoam and sonicFoam are not used for this case.

First results The results after some time, shown in Fig. 4.3a, are very promising, the recirculation zone ends pretty exactly at $2.4h$. However, in the steady-state solution in Fig. 4.3b, it has shrunken to $\approx 0.56h$, which agrees very poorly with measured data and is much worse than results achieved by NASA using their WIND code, as reported in Slater (2005). Also, the separation takes place slightly *under* the edge, which is unphysical.

Efforts towards improvement To improve this rather unsatisfactory result, various measures are taken:

- For turbulence, the RNG[2] k–ε model is chosen. While computationally only slightly more expensive than the standard k–ε model, its creators report very good predictions of the flow over a backward facing step, according to Versteeg & Malalasekera (1995). The idea, in brief, is to remove small scales of motion from the governing equations by expressing their effects in terms of larger scale motions and a modified viscosity; for details, refer to the original publication, Yakhot & Orszag (1986).
- The constant transport model, constTransport, is replaced by sutherlandTransport, the Sutherland transport model. μ is now a function of the temperature T, using a Sutherland coefficient A_s and a Sutherland temperature T_s:

$$\mu = \frac{A_\mathrm{s}\sqrt{T}}{1 + T_\mathrm{s}/T}$$

  where $A_\mathrm{s} = 1.452 \cdot 10^{-6}\,\mathrm{kg/(m \cdot s \cdot K^{1/2})}$ and $T_\mathrm{s} = 120\,\mathrm{K}$. Information about Sutherland's formula can be found in Anderson *et al.* (1984, Section 5-1.4) or Hirsch (1988, Section 2.1).

However, these measures do not improve the result perceivably. As a last resort, the mesh is refined, even though in Slater (2005), very good results are achieved with our current mesh. The new mesh is twice as fine as the old one in every direction and consists of 23'760 volumes; computation time until steady-state is reached is increased from around 4.5 hours for the old mesh to now around 43 hours, due to the smaller time step necessary.

The resulting velocity field around the step can be seen in Fig. 4.4a. With the reattachment at $\approx h$, this is an improvement indeed, but still far away from the experimental value. On the other hand, when looking at the stream lines around the recirculation area in Fig. 4.4b, it can be stated that the recirculation as such is handled well.

Even though there is obviously much room for improvement—we suspect that further mesh refinement, especially next to the boundaries, would give better results—, the solution is accepted for reasons of time limitations and the interesting cases in Section 4.2 and Chapter 5 still to be treated.

Pressure comparison Figure 4.5 shows the steady-state pressure field obtained using the refined mesh introduced in the last paragraph. The major features are easily recognised: the expansion fan at the step, the compression wave starting after the recirculation region, and the low pressure region behind the step.

[2] RNG stands for "Renormalization-Group".

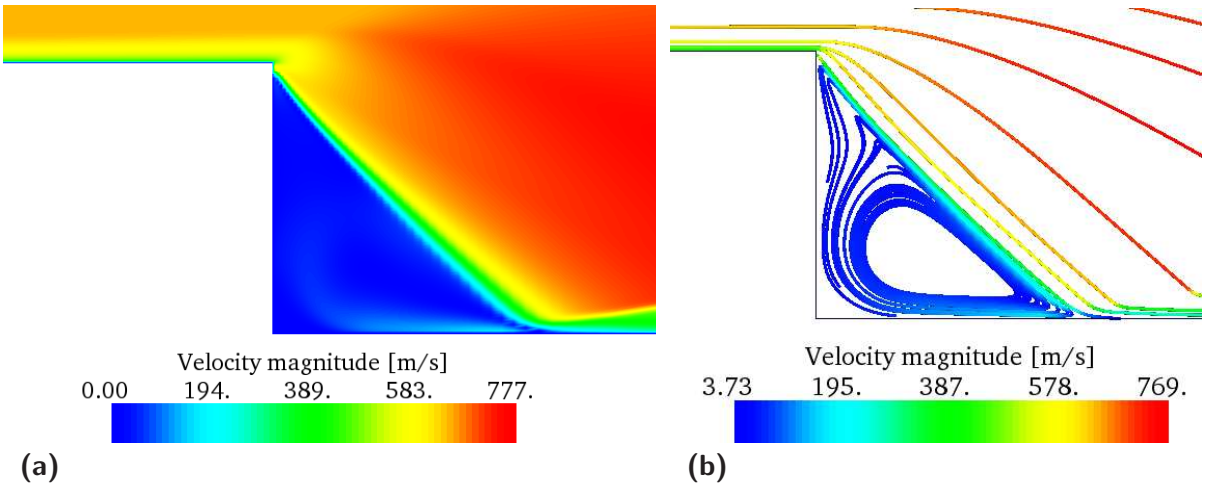


Figure 4.4: Solution obtained with the finer mesh: (a) velocity field; (b) streamlines, coloured by velocity.

Along the white line in Fig. 4.5, we take pressure samples and compare them to the experimental values given in Smith (1967) and results from CFD-ACE+ simulations, as shown in Fig. 4.6. The x-axes denote the distance from the step in inches[3], the y-axes stand for the pressure relative to the inflow static pressure, $p_\infty = 15.35\,\mathrm{kPa}$.

All curves feature low relative pressures directly after the step; the compression wave is crossed a bit after two inches, and the values approach the inflow pressure after it. However, upon closer inspection, many discrepancies become visible: the OpenFOAM simulation predicts values in the expansion fan of around 0.2, whereas measurements are around 0.4, and the actual compression wave crosses the two inch mark, while the simulation predicts it to be clearly further away. The CFD-ACE+ solutions are a little more accurate in the low pressure region (but still too low), smear out the shock and feature higher pressures further downstream. The shock is even further downstream than in the OpenFOAM prediction, so overall, it can be said that CFD-ACE+ performs a little better than OpenFOAM for this case, even though the difference between OpenFOAM and CFD-ACE+ is smaller than the one between CFD-ACE+ and the experiment.

4.1.3 Insights gained

The following points summarise this section:

- sonicTurbFoam, as a transient solver, "passes by" the correct solution, ending up at a steady state where the recirculation region is too small. Basic flow characteristics are captured nevertheless.
- Switching to the more realistic Sutherland transport model or the—in principle—better suited RNG k–ε turbulence model does not improve the result.

[3]This is not in SI units to facilitate the direct comparison to the data from Smith (1967), which is given in U. S. customary units and as a graph only.

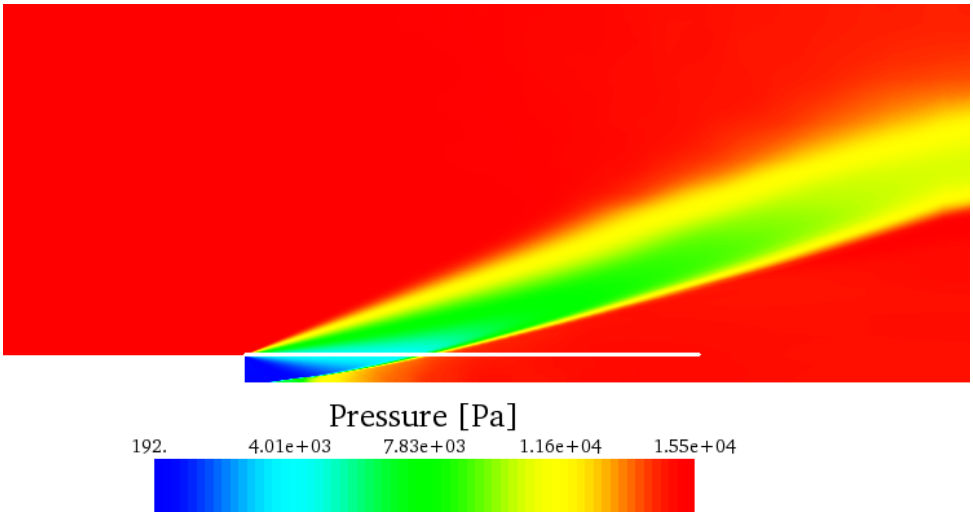


Figure 4.5: Pressure field for the steady-state solution with the refined mesh. The white line indicates the sample locations for the comparison to the experiment.

- Refining the mesh leads to a closer match to the experimental values, but implies also a massive increase in computational costs. Even further refinement or a more elaborate mesh based on more blocks[4] is suspected to improve the results again. A possible explanation is the reduction of numerical viscosity when refining a mesh, leading to a larger and more accurate recirculation region.

- A pressure sample comparison confirms the not quite satisfactory performance.

The backward facing step seems not to be a simple problem, so difficulties do not come completely unexpected: Fruth (2007) compared the performance of OpenFOAM and ANSYS CFX for the incompressible backward facing step case described in OpenCFD (2007a, Section 3.2) and reports reports troublesome convergence behaviour, even when using a simpler steady-state solver.

The lesson learned from this case is the fact that recirculation needs special care and fine meshes; satisfactory agreement to experimental values requires a lot of tweaking.

4.2 The transonic diffuser problem

The second validation case is the *transonic diffuser problem*. Subsection 4.2.1 describes the experimental and the simulation setup and gives an overview of the different sources, as consistent information is somewhat dispersed across several publications. Subsection 4.2.2 tests the performance of sonicTurbFoam on the basis of two different variants of the case and compares it to CFD-ACE+; Subsection 4.2.3 finally summarises the section.

[4]The mesh generation tool blockMesh does not allow for bigeometric refinement, so to emulate that blocks have to be split.

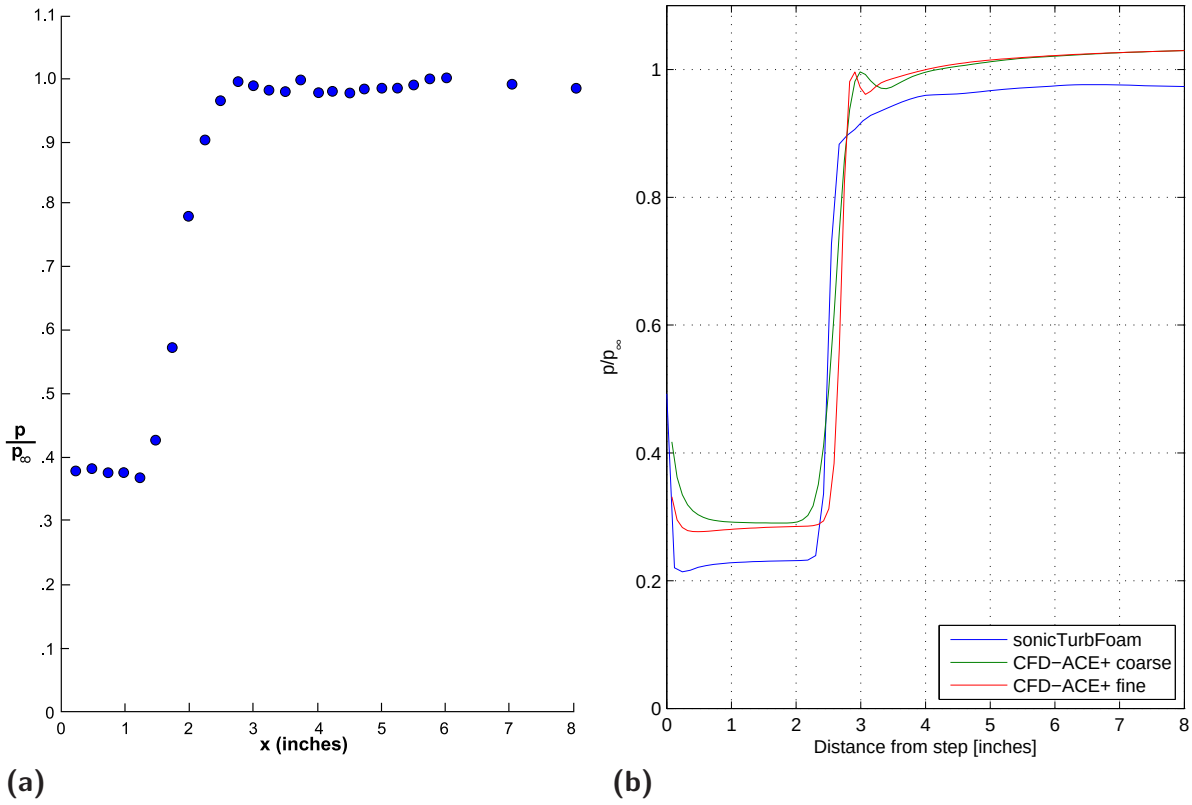


Figure 4.6: Pressure sample comparison for the backward facing step: (a) experimental values from Smith (1967); (b) simulation results using the finer mesh in OpenFOAM and two different meshes in CFD-ACE+.

4.2.1 Description and relevance

As the backward facing step, the transonic diffuser[5] is a popular benchmark for CFD codes. Figure 4.7 shows the setup: subsonic flow enters a duct with a constant cross-section. The duct then converges until a throat height h_* is reached, accelerating the flow to supersonic speed; the diverging part accelerates the flow further, until after a normal shock speeds go back to subsonic. Somewhere after the shock, we also find the flow separating from the upper wall.

Slater (2005) conducts studies on this case for NASA's WIND code and gives quite a comprehensive overview of the case. However, to get enough details to be able to conduct our own proper study, we have to consult several other publications: from Georgiadis *et al.* (1994), an evaluation of different turbulence models in NASA's older PARC code, we obtain more detailed information about the computational mesh; the exact description of the diffuser geometry is taken from Bogar *et al.* (1983), where oscillation frequencies for the transonic diffuser flow are examined; in Hsieh *et al.* (1987), a numerical investigation of unsteady inlet flow fields—the actual experiments we try to mimic here—are described, and in Salmon *et al.* (1983) finally, laser Doppler velocimetry measurements of the same flow are the subject.

[5]Sometimes also called "transonic nozzle".

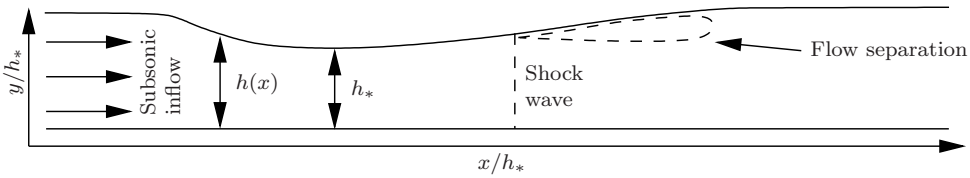


Figure 4.7: The transonic diffuser setup: subsonic inflow is accelerated to supersonic speed, a normal shock appears in the diverging part as well as flow separation.

Geometry and computational mesh The geometry of the diffuser is shown in Slater (2005), but defined more clearly in Bogar *et al.* (1983): the channel has a flat bottom and the top wall height $h(x)$ is given by

$$\tilde{h}(\tilde{x}) = \frac{\alpha \cosh \zeta}{(\alpha - 1) + \cosh \zeta} \tag{4.1}$$

where values with a tilde are made dimensionless by division by the throat height $h_* = 44\,\mathrm{mm}$, e. g., $\tilde{h} = h/h_*$. ζ in Eq. (4.1) is given by

$$\zeta = \frac{C_1(\tilde{x}/\tilde{\ell})\big(1 + C_2(\tilde{x}/\tilde{\ell})\big)^{C_3}}{(1 - \tilde{x}/\tilde{\ell})^{C_4}} \tag{4.2}$$

The values for the different constants in Eqs. (4.1) and (4.2) in the convergent and divergent part are given in Tab. 4.1. C_3 is not given for the diverging part because $C_2 = 0$ there and thus $\big(1 + C_2(\tilde{x}/\tilde{\ell})\big)^{C_3} = 1^{C_3} = 1$.

Constant	Converging	Diverging
α	1.4114	1.5
$\tilde{\ell}$	-2.598	7.216
C_1	0.81	2.25
C_2	1.0	0
C_3	0.5	—
C_4	0.6	0.6

Table 4.1: Constants for the calculation of the diffuser geometry.

Inlet and exit are defined to be at $\tilde{x}_\mathrm{i} = -4.04$ and $\tilde{x}_\mathrm{e} = 8.65$, respectively. A plot of the resulting shape is shown in Fig. 4.8 (with dimensionalised values). When calculating the geometry, attention has to be paid that the formulas are only to be used in the non-constant cross-section region $-2.598 \leq \tilde{x} \leq 7.216$. For the constant regions, $\tilde{h}(\tilde{x} < -2.598) = 1.4114$ and $\tilde{h}(\tilde{x} > 7.216) = 1.5$ are valid.

To create a mesh that enables enough flexibility from this data using blockMesh, 10 blocks are defined: five for the upper half and five for the lower half to emulate bigeometric refinement at the walls. In axial direction, block boundaries are set conveniently to ensure a fine mesh where it is required, i. e., in the region where the normal shock is expected. The block edges that are curved are approximated by polygonal lines with a sufficient number of nodes. The mesh has 81×51 volumes, just like the coarse one described in Georgiadis *et al.* (1994); it is shown in Fig. 4.9.

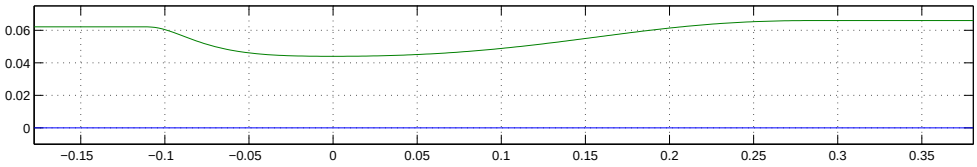


Figure 4.8: The geometry of the transonic diffuser, as calculated from Eqs. (4.1) and (4.2). Axes are labelled in metres.

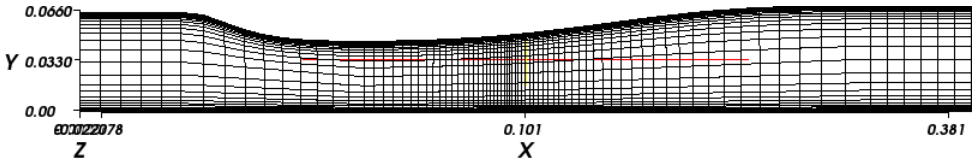


Figure 4.9: The 81×51 mesh for the transonic diffuser (axes labelled in metres).

Specific task description Two cases are being looked at, a so called *strong shock* and a *weak shock* case. They differ by the ratio R of exit static to inflow total pressure: for the strong shock it is $R = 0.72$, for the weak shock $R = 0.82$. The flow conditions are shown in Tab. 4.2.

Inflow		**Outflow**	
Total pressure [kPa]	134.4	Static pressure, weak shock [kPa]	110.7
Total temperature [K]	277.8	Static pressure, strong shock [kPa]	97.2
Mach number	0.9		

Table 4.2: Flow conditions for the transonic diffuser.

To specify these values in a more convenient way, the total inflow values are converted to static values, using the perfect gas relations

$$\frac{T_0}{T} = 1 + \frac{\gamma - 1}{2}\mathrm{Ma}^2 \tag{4.3}$$

$$\frac{p_0}{p} = \left(\frac{T_0}{T}\right)^{\gamma/(\gamma-1)} = \left(1 + \frac{\gamma - 1}{2}\mathrm{Ma}^2\right)^{\gamma/(\gamma-1)} \tag{4.4}$$

where the index 0 stands for the total values and γ is set to 1.4 for air. This results in the inflow values $T_\mathrm{i} = 266.6\,\mathrm{K}$ and $p_\mathrm{i} = 116.8\,\mathrm{kPa}$; the velocity, based on Mach number and inflow temperature, is $u_\mathrm{i} = 150.6\,\mathrm{m/s}$.

Solver setup and computation strategy The boundary conditions are set as follows:

- p: Fixed value at inlet and exit, zero gradient at the walls
- T: Fixed value at inlet, zero gradient at exit and walls
- $\mathbf{u}$: Fixed value at inlet, zero gradient at exit, no-slip at walls

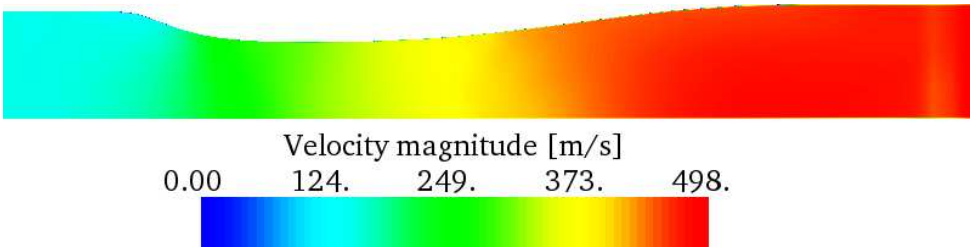


Figure 4.10: Steady-state diffuser velocity field for $R = 0.13$.

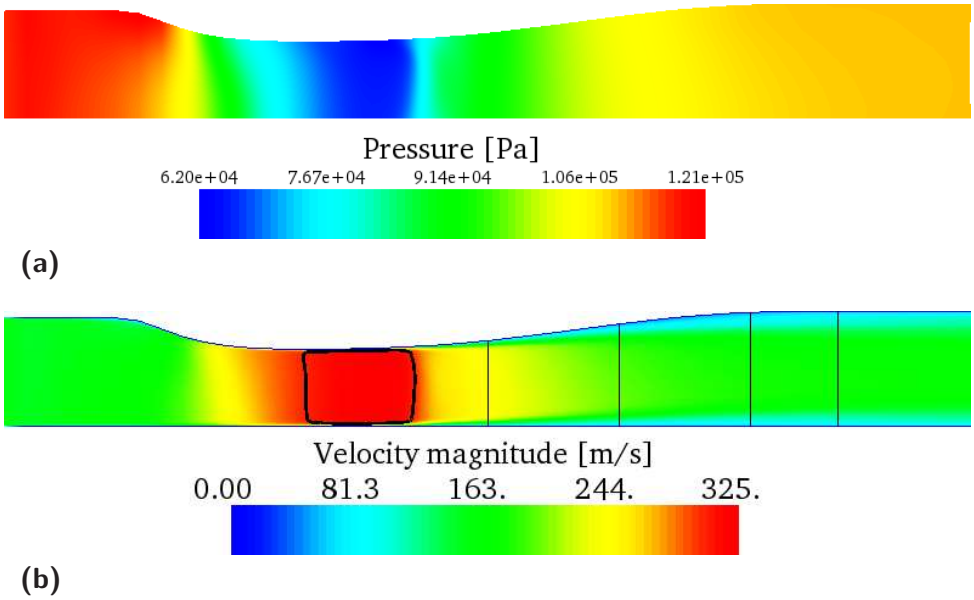


Figure 4.11: Weak shock solution for the diffuser: (a) pressure field; (b) velocity field with $\mathrm{Ma} = 1$ isoline. The vertical bars indicate the locations of the four velocity samples taken.

- k and ε: Zero gradient everywhere

To start with, R is set to a small value, $R = 0.13$, which corresponds to an exit static pressure of $p_\mathrm{e} = 17.24\,\mathrm{kPa}$, and the turbulence model is set to laminar; the idea is to end up with an initial flow field where no shock occurs. Starting from this smooth field, the weak and strong shock cases are obtained by setting the corresponding exit pressure conditions. The initial fields are set as follows:

- p: Exit value everywhere (discontinuity at the inlet)
- T: Inlet value everywhere
- $\mathbf{u}$: Zero velocity everywhere
- k and ε: Appropriate values, estimated based on a 5% turbulence intensity

At $t = 30\,\mathrm{ms}$ with a time step $\Delta t = 4 \cdot 10^{-6}\,\mathrm{s}$, a steady state is reached; the corresponding smooth velocity field is shown in Fig. 4.10.

The initial approach, switching to the RNG k–ε turbulence model and setting the pressure ratio to the weak shock configuration, results in a shock wave moving towards the inlet, where pressure oscillations with unphysically high peak values occur, blowing up the simulation in the end.

Instead, a different method leads to more stable results: the exit pressure is increased in small steps, each time kept fixed until a steady-state flow field is reached, and then increased again. This way, the strong shock results are obtained first, and the weak shock results afterwards.

The transport model is, as in the backward facing step case, set to Sutherland transport.

4.2.2 Solver quality evaluation

In Slater (2005), tables containing the exact numerical values of the measurements executed in Hsieh *et al.* (1987) are included, so we can numerically compare our simulation results to them instead of just visually compare different plots. Also, we include results from CFD-ACE+ simulations. The two quantities being looked at are the *pressure distribution* along the top and bottom wall and *velocity profiles* at four different positions, 2.882, 4.611, 6.340 and 7.493 (dimensionless values).

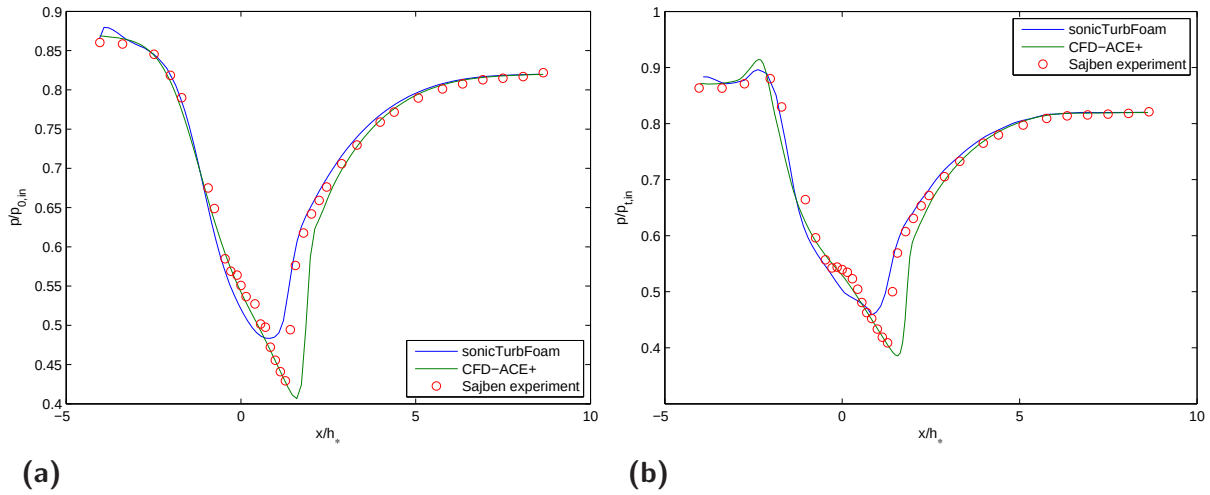
The weak shock case Figure 4.11 shows pressure and velocity fields for the weak shock solution, with an isoline for Ma = 1 in black, matching the pressure jump. Flow separation is not strongly developed, the low speed boundary layers after the shock are quite thin.

In Fig. 4.12, the pressure along the bottom and the top wall is shown. While the location of the shock is captured well, its minimum is estimated too high by sonicTurbFoam; the shock prediction, especially at the bottom (Fig. 4.12a), is also less sharp than the measurements. Agreement before and after the shock, however, is satisfactory. CFD-ACE+ captures the shock better and sharper, but predicts a minimum pressure value lower than the actual one, just like in the region immediately behind the shock, i. e., it predicts the shock a bit too far to the right.

Figure 4.13 shows velocity profiles (only the x-component of the velocity) at four different locations downstream of the throat (the vertical bars in Fig. 4.11b). Both sonicTurbFoam and CFD-ACE+ systematically underestimate velocities; the CFD-ACE+ results are more accurate, though. Even the highly optimised WIND and NPARC results in Slater (2005) underestimate velocity; the reason for the boundary layers being too large for sonicTurbFoam might again be numerical viscosity, caused by a mesh that is too coarse.

In an endeavour to improve these results, especially concerning the shock sharpness, a simulation with a mesh refined in x-direction (114×51 volumes) around the shock location is set up; however, because of stability issues and quite large demands on CPU time, it is decided to not follow the mesh refinement study approach any further.

The strong shock case The strong shock case ($R = 0.72$) pressure and velocity fields are shown in Fig. 4.14. The shock now appears further downstream than in the weak shock case, and the area where Ma > 1 is correspondingly larger. Figure 4.14b shows



(a) (b)

Figure 4.12: Pressure distribution for the weak shock case, comparison to CFD-ACE+ solution and experimental values: (a) bottom wall; (b) top wall

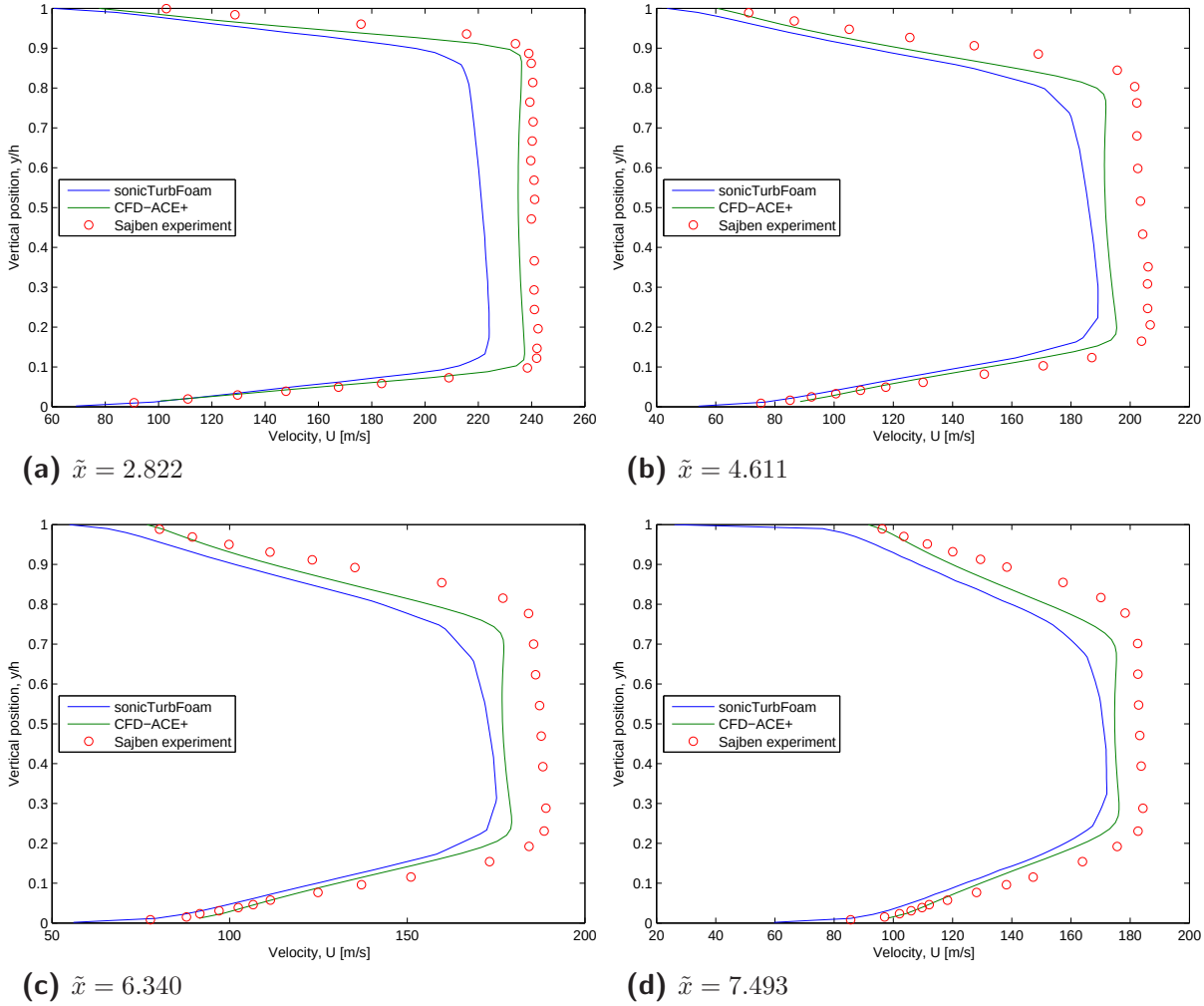
that the first velocity sample is taken right where the shock occurs, so prediction of the shock at the wrong place leads to very large deviation in the velocity sample.

Figure 4.14b features a low speed region after the shock which is much larger than for the weak shock case (see Fig. 4.11b); as can be seen in Fig. 4.15, this is the recirculation pocket for this case. There is a second recirculatory area near the bottom wall, as indicated by the small bump in the lowest streamline. We show the streamlines only for the strong shock case because the features are the same as for the weak shock case, but they are more developed and thus better visible in the strong shock case.

The pressure distribution shown in Fig. 4.16 shows very good agreement between sonicTurbFoam and the measurements. The shock is predicted a little too far downstream, but the minimum values agree very well. For the bottom wall (Fig. 4.16a), the CFD-ACE+ result—again—features the shock even further downstream than sonicTurbFoam, and again with a minimum value that is too small. We will see later that this slight difference has a quite strong effect on the velocities measured. For the top wall (Fig. 4.16b), the simulation results are almost identical, the CFD-ACE+ shock being only marginally downstream of the sonicTurbFoam one. Differences of this order of magnitude could easily be caused be slight differences in meshing—the two simulations do not use the same mesh.

Figure 4.17 shows the strong shock velocity profiles. In the sample right at the shock (Fig. 4.17a), sonicTurbFoam overpredicts the velocity everywhere except in the upper core flow region, where the values agree with the measurements. CFD-ACE+ however, as mentioned before, suffers from predicting the shock too far downstream: the profile shows no recirculation and velocity overpredictions of up to 100 m/s.

The agreement of CFD-ACE+ with the measurements is much better further downstream, even though the shape of the core flow is not quite captured in the two downstream samples (Figs. 4.17c and 4.17d). While for both solvers, the upstream velocity predictions at the top wall are too high, they agree excellently in the third sample and are even underpredicted in the last sample. sonicTurbFoam underpredicts core flow velocities in the



(a) $\tilde{x} = 2.822$ **(b)** $\tilde{x} = 4.611$

(c) $\tilde{x} = 6.340$ **(d)** $\tilde{x} = 7.493$

Figure 4.13: Weak shock velocity profiles at different $\tilde{x}$-locations, comparison to CFD-ACE+ and experimental values.

last three samples.

Potential improvements In Slater (2005), a different computation strategy is chosen: after setting up supersonic flow without shocks ($R = 0.12$), the pressure is directly set to the weak shock case ($R = 0.72$) and run for 1000 iterations with an SST[6] turbulence model, from which the k–ε model is initialised and run for 10'000 iterations. For the strong shock case, the weak shock solution is taken as the starting point and the simulation is run for 10'000 iterations directly using the k–ε turbulence model.

In addition to using an SST model, WIND offers two more correction factors:

- *Sarkar compressibility correction:* provides for an increase in the dissipation rate at higher Mach numbers to account for observed reduction in thin shear layer growth

[6]Shear Stress Transport: Near the walls, a turbulence/frequency based model (k–ω) is solved, in the bulk flow, k–ε is used. A blending function ensures smooth transition between the two models.

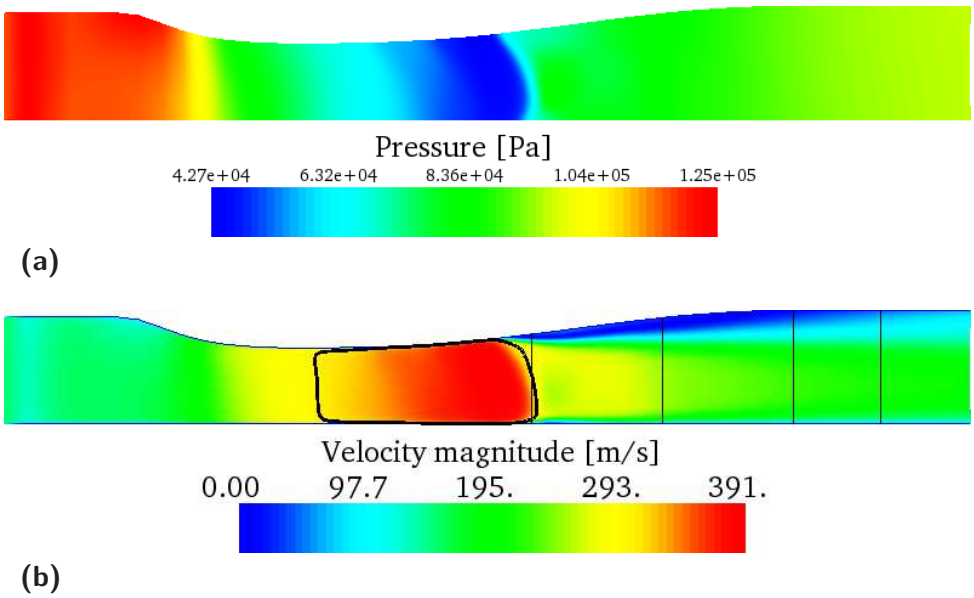


Figure 4.14: Strong shock solution for the diffuser: (a) pressure field; (b) velocity field with Ma = 1 isoline. The vertical bars indicate the locations of the four velocity samples taken.

rate with increasing Mach number.

- *Variable C_μ:* reduces turbulent viscosity in regions where the ratio of production to dissipation of turbulent kinetic energy becomes large.

Using both correction factors, the results in Slater (2005) are clearly improved. OpenFOAM features no SST turbulence model; neither Sarkar compressibility nor a variable C_μ option are included. The open architecture would of course allow to extend OpenFOAM to that effect, but doing so lies beyond the scope of this work.

4.2.3 Insights gained

The main learnings from this section can be summarised as follows:

- Flow results are not obtained easily: quite an elaborate procedure is required to obtain results.
- Results for the weak shock case are satisfactory, even though the shock is not captured very sharply and velocities are slightly underestimated. CFD-ACE+ outperforms sonicTurbFoam for this case.
- For the strong shock case, the results are even better; CFD-ACE+ suffers from predicting the shock too far downstream. Specifically, it predicts the location of shock waves at a larger cross-sectional area than measured, overestimating the compressibility of the flow.
- Further improvements would require to extend OpenFOAM, which is—contrary to most of the commercial CFD codes—possible. For this case, better physical modelling for boundary layer/shock interaction would have to be implemented.

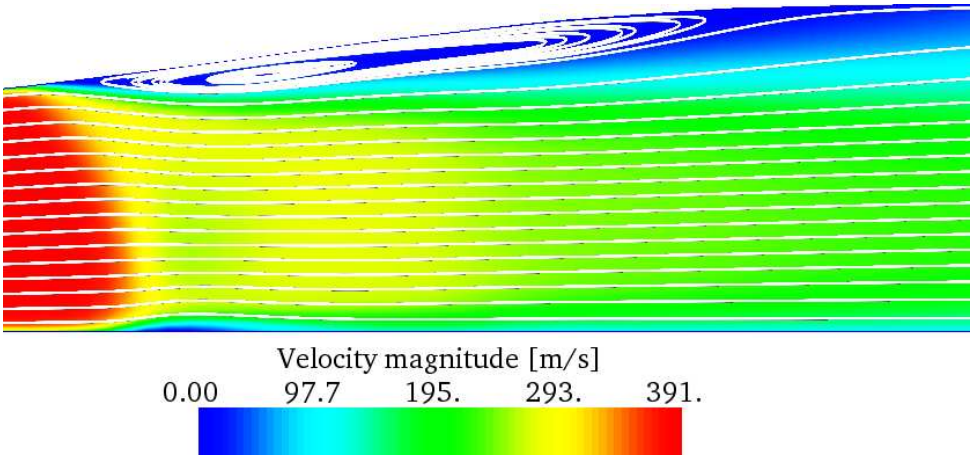
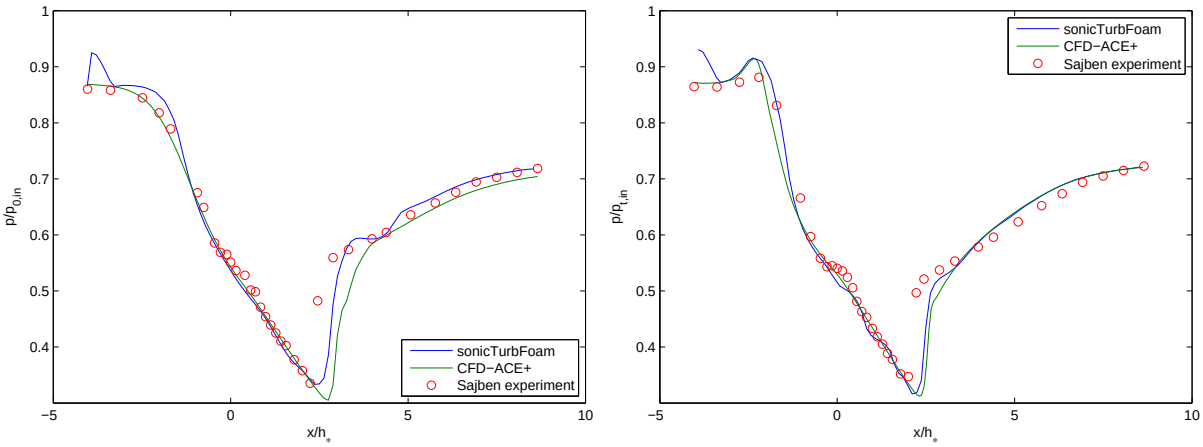


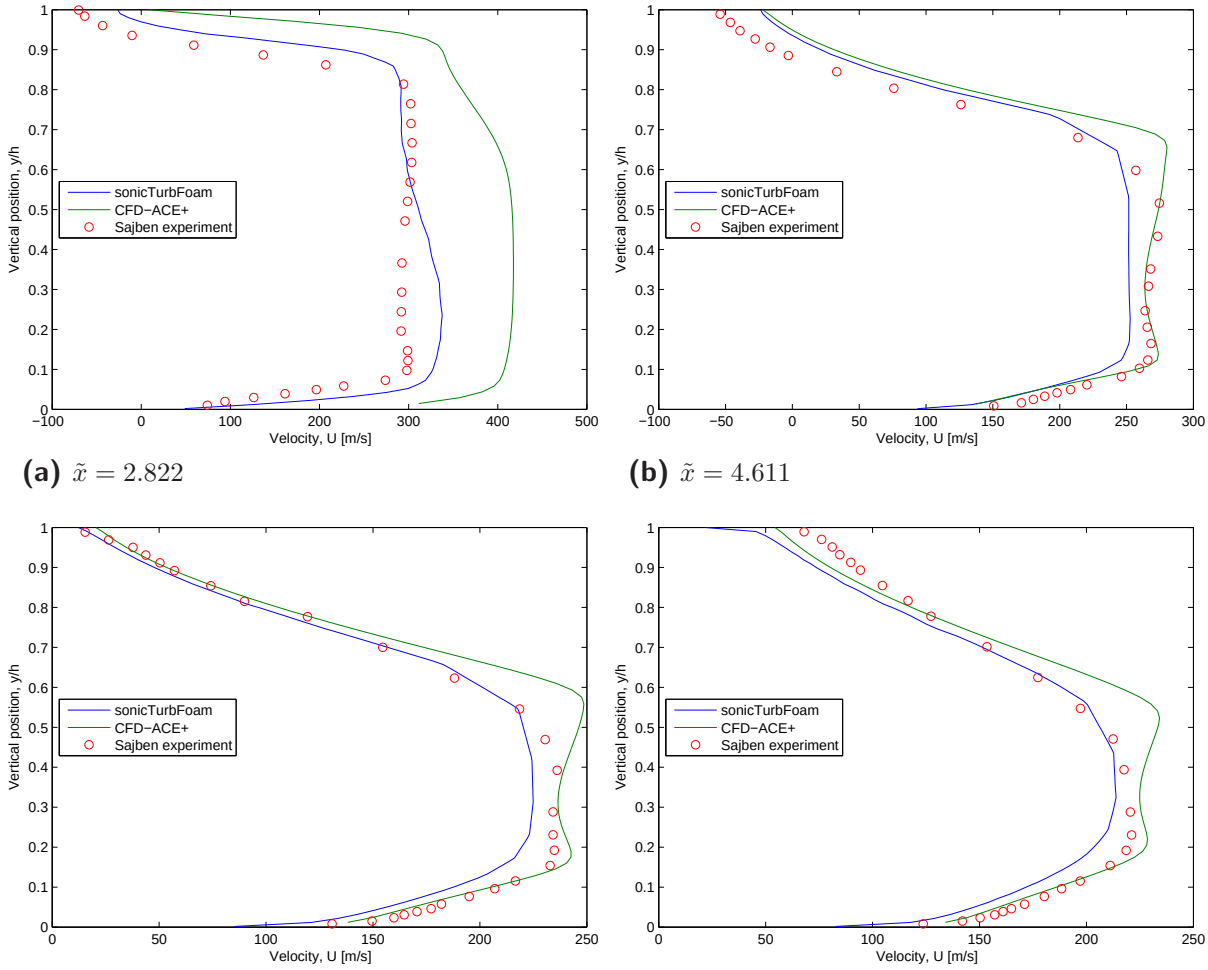
Figure 4.15: Detail of the strong shock velocity field with streamlines, showing separation and the recirculation pocket.

With the confidence that sonicTurbFoam is able to handle diffuser flows with shocks, separation and recirculation, the final case with a real world application, the cold gas flow in a circuit breaker nozzle geometry, is tackled in Chapter 5.



(a) **(b)**

Figure 4.16: Pressure distribution for the strong shock case, comparison to CFD-ACE+ solution and experimental values: (a) bottom wall; (b) top wall



(a) $\tilde{x} = 2.822$ **(b)** $\tilde{x} = 4.611$

(c) $\tilde{x} = 6.340$ **(d)** $\tilde{x} = 7.493$

Figure 4.17: Strong shock velocity profiles at different $\tilde{x}$-locations, comparison to CFD-ACE+ and experimental values.

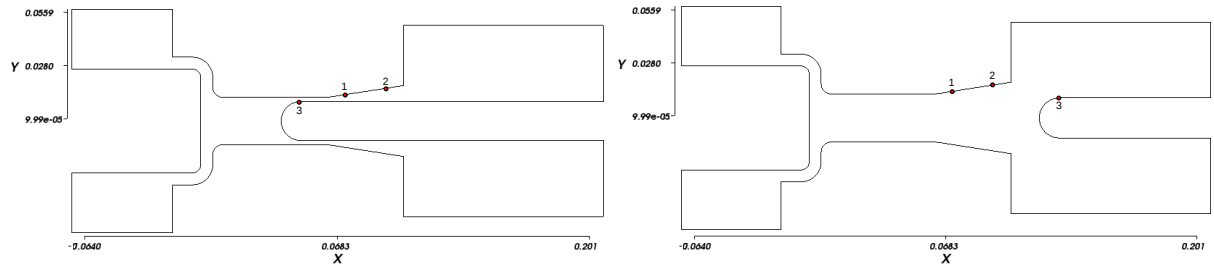


Figure 5.1: ABB breaker geometries (cross-sections): the high pressure reservoirs (inlets) are to the left, the constant pressure reservoirs (outlets) to the right. The red dots mark the sensor positions. (a) plug at 37 mm; (b) plug at 112 mm.

5 Cold gas flow in a circuit breaker

The final case—the real world application—is the simulation of cold gas flow in a geometry very similar to the one of a circuit breaker at different plug positions and inflow pressures. Subsection 5.1 describes the exact setup of the case; Subsection 5.2 gives some more information about the meshes used and the solver settings; Subsection 5.3 reports on challenges encountered during the simulations; in Subsection 5.4, typical results are shown, and in Subsection 5.5, the simulation results are compared to experimental values. Subsection 5.6 wraps up the chapter by summarising its findings.

5.1 Case description

As laid out in the description of the self-blast breaker functioning principle in Section 1.1, gas flows from a high pressure reservoir along the light arc to a lower pressure region. In Fig. 5.1, the corresponding geometries are shown in 2D; the real geometries would be obtained by rotating the cross-sections about their axis of symmetry. We see two snapshots with different distances between plug and tulip: in Fig. 5.1a, the plug has moved 37 mm, in Fig. 5.1b it is 112 mm away. These are the two most extreme cases we look at in this chapter.

In particular, the fourteen cases measured in Mantilla Florez (2007) are modelled, i. e., with the plug positioned at 37 mm, 42 mm, 47 mm, 52 mm, 57 mm, 62 mm, 67 mm, 72 mm, 77 mm, 82 mm, 87 mm, 92 mm, 102 mm and 112 mm.

The measurements in Mantilla Florez (2007) are performed as follows for every plug position:

- Initial pressure is 1 bar everywhere
- Initial temperature is 25 °C = 298.15 K everywhere
- Within 25 ms, the inflow pressure is linearly raised to 3 bar
- Every 2.5 ms, the pressure is measured with three sensors

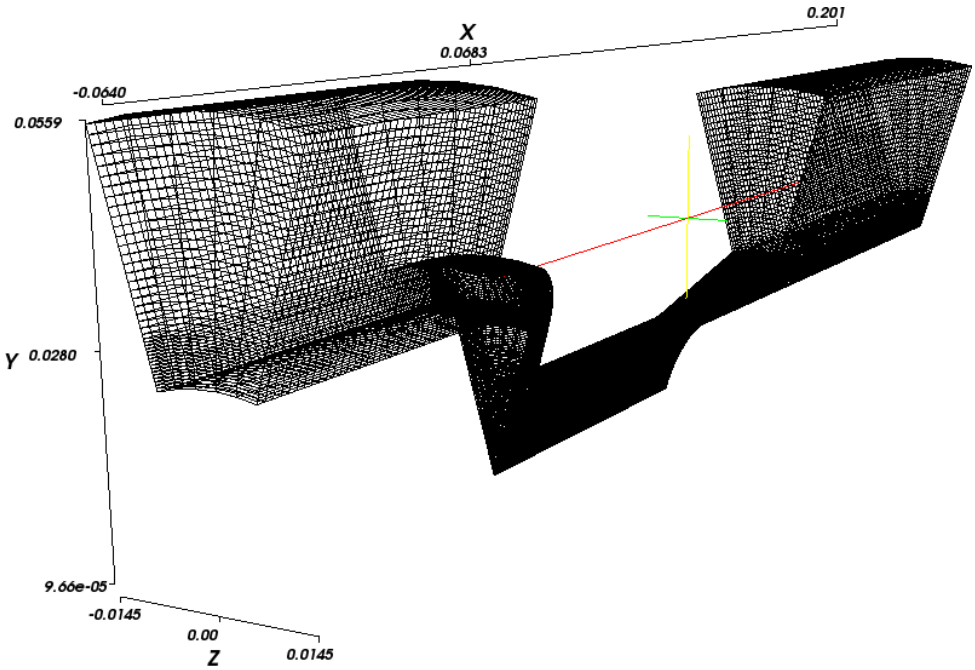


Figure 5.2: Mesh for the 57 mm circuit breaker case (80'740 volumes).

The approximate sensor placement is marked in Fig. 5.1 with numbered dots; sensors 1 and 2 are fixed on the diffuser, sensor 3 is being moved with the plug. In total, there are 10 measurements for each of the 14 plug positions, generating three values per measurement. The exact placement of the sensors is shown in Tab. 5.1.

Sensor number	x-coordinate	y-coordinate
1 (nozzle)	73.9479 mm	13.3117 mm
2 (diffuser)	92.7000 mm	16.6100 mm
3 (plug)	12.7321 mm + x_c	9.9928 mm

Table 5.1: Location of the control points for the ABB circuit breaker case. x_c is the plug location, e. g., 37 mm in for the 37 mm case.

5.2 Meshes and solver settings

For every plug position, a structured mesh is created using CFD-GEOM. It is then exported to GAMBIT, saved as a Fluent *.msh* file and finally converted to OpenFOAM format using the fluentMeshToFoam utility. These meshes feature between 75'500 volumes for the 37 mm case and 128'780 volumes for the 112 mm case. An example (the 57 mm case) is shown in Fig. 5.2.

The meshes have five volumes in azimuthal direction and constitute thus a compromise between a full 3D approach and a 2D axi-symmetric one. To get an idea of the dimensions: the length (axial direction) of the geometry is approximately 260 mm, the radius of the plug around 10 mm.

The outer walls, the tulip and the plug surfaces are assigned the wall type with no-slip for the velocity field and zero gradient for every other field. The outlet patch is of fixedValue type for pressure ($p = 1\,\text{bar}$) and zeroGradient for the other fields.

The large surface patches that constitute the "cut surfaces" are assigned the symmetryPlane type. While OpenFOAM would allow for wedge-shaped volumes, CFD-GEOM does not, so the axis of symmetry is actually a very narrow surface, which is also set to the symmetryPlane type.

The first idea for the inlet patch is to have it as a fixed pressure inlet and run one simulation for every pressure/plug position combination until steady-state is reached, thus requiring 140 simulation runs. However, first tests show that it takes very long until the residuals are acceptably small, so a different approach is chosen.

The inlet is assigned the timeVaryingUniformFixedValue type, which is not mentioned in the documentation, but does exactly what we are looking for: a series of points in time with corresponding pressure values can be specified, and OpenFOAM interpolates linearly between them, so we can exactly copy the experimental setup in Mantilla Florez (2007) (and also the CFD-ACE+ settings therein).

The experiments are conducted with N_2, so the *thermodynamicProperties* dictionary is adapted accordingly. Because temperature effects are expected to be negligible, the transport model is set to constant transport. For turbulence, RNG k–ε is chosen.

The solver of choice is, based on the encouraging experience in Section 4.2, sonicTurbFoam. The initial time step is set to $\Delta t = 5 \cdot 10^{-7}\,\text{s}$, and as the time scheme, `backward` is chosen, a second order implicit scheme. Intermediate volume fields are written out every 0.625 ms.

5.3 Progression and computational costs

The choice of `backward` as the time scheme turns out to be a bad one because of the scheme's unboundedness, so it is decided to switch to `CrankNicholson` with a blending coefficient of 0.6; this scheme is also second order implicit, but bounded.

sonicTurbFoam does not feature an adaptive time step. It would not be extraordinarily difficult to extend the solver to this end though, but for reasons of simplicity, we stick to the fixed time step and simply reduce it whenever problems come up.

Even though the scheme is bounded, it is not arbitrarily stable. Especially for the cases with a smaller plug distance where the nozzle effect is maximal, the initial step size leads quickly to the non-convergence of one of the variables. Whenever this happens, Δt is reduced and the simulation restarted from the last intermediate result. Figure 5.3 shows what this means for three selected geometries.

For the 37 mm case, the step size has to be reduced to $10^{-7}\,\text{s}$ after 1.25 ms. At 3.125 ms, Δt is already down to $10^{-8}\,\text{s}$. The 72 mm case gets a bit further and seems to need no smaller Δt than $2.5 \cdot 10^{-8}\,\text{s}$. The time step for the 102 mm case finally has to be reduced no earlier than at 7.5 ms.

A single time step requires roughly five seconds to compute on a single CPU of the parallel cluster at ABB, depending on the actual mesh size and various other parameters. At $\Delta t = 10^{-8}\,\text{s}$, 2.5 ms require 250'000 time steps; at five seconds per time step, this equals more than two weeks of computing time. Because of limited computing resources,

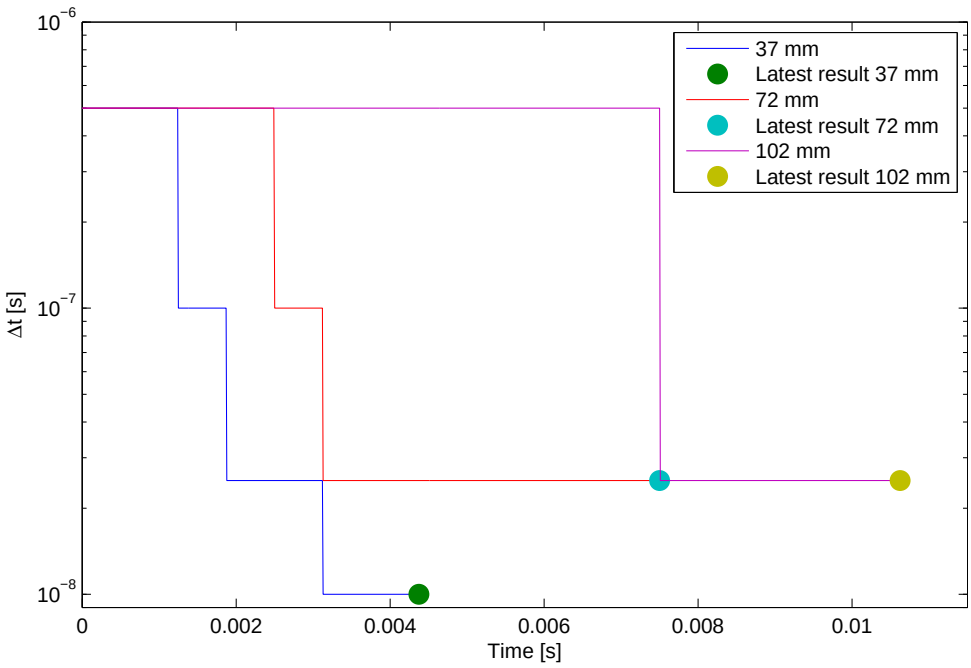


Figure 5.3: Necessary reduction of Δt with increasing inlet pressure, shown for three different geometries.

only around one third of all the measurement values are simulated; however, this should be enough to make a well-grounded statement about the performance of OpenFOAM for this case.

The discontinuities in the time step size are also reflected in the residual plots. As an example, we look at the initial residuals of the 87 mm case, shown in Fig. 5.4a: with the increasing pressure, the Courant number becomes larger and larger, until at 2.5 ms, the time step has to be reduced for the first time, because the simulation diverges shortly afterwards otherwise. The reduction is to $5 \cdot 10^{-8}$ s, because the first try with 10^{-7} s does not work out; this causes the sharp drop visible at 2.5 ms.

From then on, the residuals seem to behave pretty wildly, but a look at the magnification of just 1 ms (Fig. 5.4b) reveals that the spikes are just occasional and the residual behaviour is smooth otherwise. Also, only initial residuals are shown, which makes the plot seem more discontinuous than how the actual quantities really behave.

5.4 Exemplary Mach and pressure fields

To get an idea what flow patterns come up in the course of such a run, we take a look at a snapshot of the 62 mm case, shown in Fig. 5.5.

A variety of flow phenomena already encountered in the earlier cases of this thesis can be observed:

- A weak *expansion fan* occurs in front of the tulip at $x \approx 0.01$ m (consult Fig. 5.1 for the axis labels).

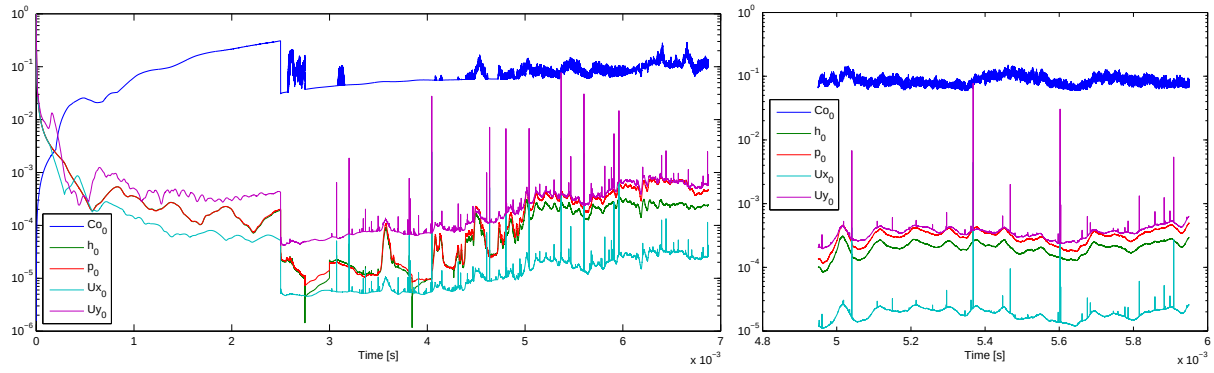


Figure 5.4: Initial residuals for the 87 mm case: maximum Courant number Co_0, entrophy h_0, pressure p_0, x- and y-components of velocity $U_{x,0}$ and $U_{y,0}$: (a) 0–7 ms; (b) 5–6 ms.

- Directly in front of the tulip, there is a *stagnation point* (not well visible with the colour scheme used).
- In the diffuser, the flow is accelerated to supersonic speeds until a *normal shock* occurs, visible in both the pressure and the Mach number field.
- After the shock, *separation* occurs and a recirculatory area around the plug is formed.

5.5 Comparison to measurements

These flow features all seem plausible, but cannot be validated because the measurements only yield values at individual points in the flow field. To assess the simulation on a more quantitative basis, we now compare the OpenFOAM predictions to the measured values from Mantilla Florez (2007).

To this end, the predicted and measured values are set relative to the inlet pressure; Figs. 5.6 to 5.8 show these relative pressures as functions of the plug position for all three sensors. OpenFOAM predictions and the measurements for a configuration are in the same plots for direct comparison. Some data points that are extremely off call for explanations, so after checking the corresponding pressure and velocity fields, usually a reason can be given for strong deviations.

Sensor 1, nozzle (Fig. 5.6) For the lower pressure cases in Fig. 5.6a, the pressure dip for the 57 mm configuration is the most obvious deviation. The reason is a normal shock being predicted a little too far downstream, i. e., behind the sensor. In Fig. 5.6b with the higher pressures, there are only few simulation results. The value at 77 mm lies exactly in a small flow separation area behind a shock; the values at 102 and 112 mm are, again, slightly upstream of a shock, behind which the pressure corresponds to the measurement.

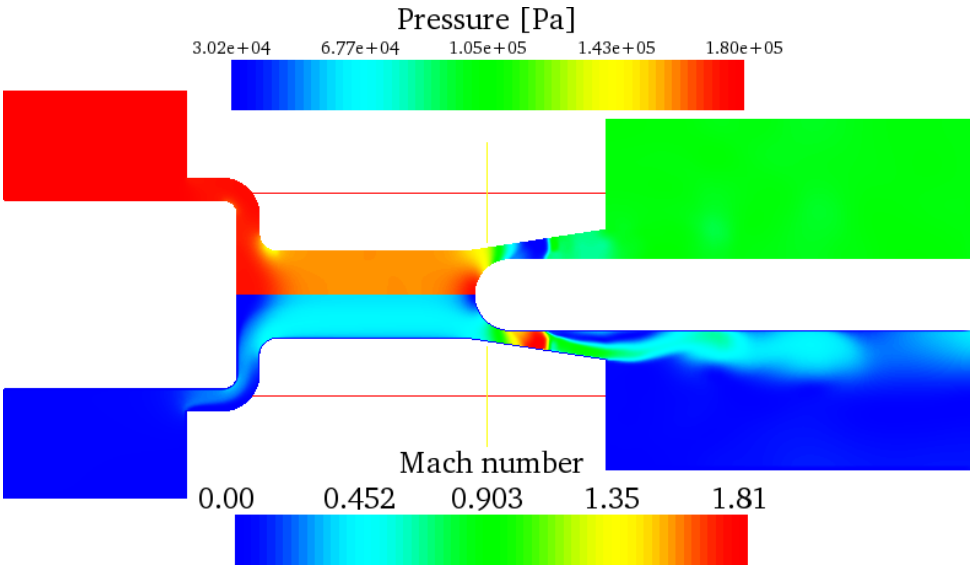


Figure 5.5: Pressure and Mach number field for the 62 mm case at $t = 10$ ms, i. e., inflow pressure at 1.8 bar.

Sensor 2, diffuser (Fig. 5.7) In Fig. 5.7a, the "off" point is at 72 mm for the 1.6 bar case; the reason is again a normal shock being estimated too far downstream. It is to suspect that inaccurate shock locations are also the reason for the wiggles in the 1.4 bar series. The few higher pressure values in Fig. 5.7b are all satisfactory or even almost perfect.

Sensor 3, plug (Fig. 5.8) Apart from the systematic underestimation of pressure for cases up to 57 mm—this happened also to the CFD-ACE+ simulation in Mantilla Florez (2007)—, the spikes at 82 mm are the most striking feature of Fig. 5.8a. Again, there is a shock next to the sensors for these configurations, but this time, it is estimated too far upstream, leading to increased pressure.

5.6 Summary for circuit breaker case

The following learnings are taken from this case:

- OpenFOAM's sonicTurbFoam is able to handle real world applications with satisfactory results.
- Importing meshes is not a big problem.
- The non-adaptivity of the time step is quite a nuisance: once reduced to a small value like 10^{-8} s, cases with meshes containing tens of thousands of volumes require a huge amount of computing time. On the other hand, nobody precludes the development of such an adaptive scheme.
- Arbitrarily selected Mach number and velocity fields seem to be plausible.

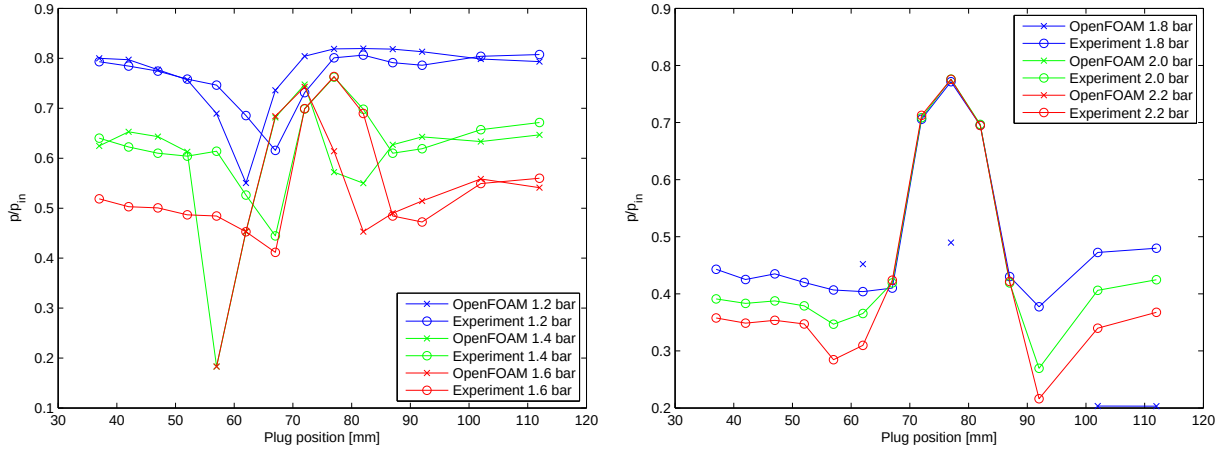
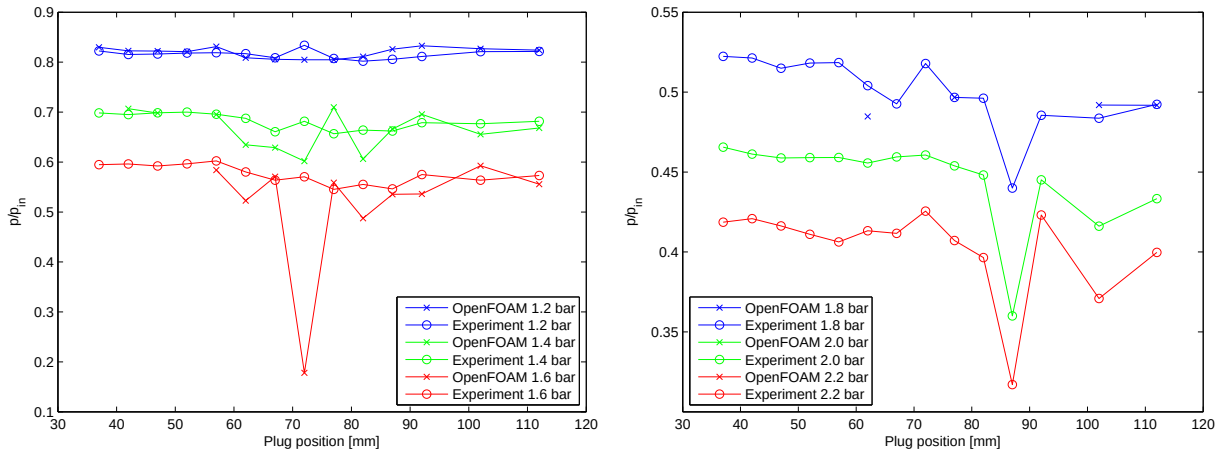


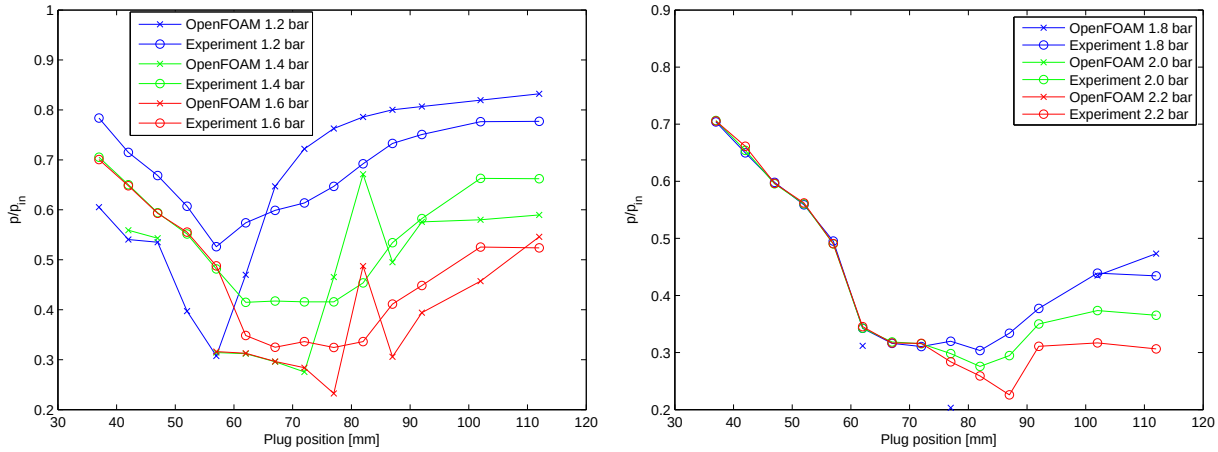
Figure 5.6: Comparison of simulation predictions and measured values for pressure values at sensor 1 (nozzle). Crosses stand for simulation values, circles for measured values; lines in the same colour are for the same pressure. (a) 1.2 bar–1.6 bar; (b) 1.8 bar–2.2 bar.

- Comparison to measurements yields respectable results; whenever a value is way off, the reason is a false estimation of a shock location.



(a) **(b)**

Figure 5.7: Comparison of simulation predictions and measured values for pressure values at sensor 2 (diffuser): (a) 1.2 bar–1.6 bar; (b) 1.8 bar–2.2 bar.



(a) **(b)**

Figure 5.8: Comparison of simulation predictions and measured values for pressure values at sensor 3 (plug): (a) 1.2 bar–1.6 bar; (b) 1.8 bar–2.2 bar.

6 Summary and outlook

This chapter summarises the findings of the thesis (Section 6.1) and gives an outline for suggested future research in the subject area (Section 6.2).

6.1 Lessons learned and recommendation

Each of the five test cases features its own section on what constitutes the key learnings of the case. This section here draws conclusions from these learnings and points out commonalities.

- All in all, OpenFOAM is *able to deal with every case* it is presented with: the outcomes are between excellent (less than 1% deviation for a fine grid using rhopSonicFoam in the supersonic wedge case, less than 2% in the shock tube case), good (transonic diffuser results) and barely satisfactory (recirculation in the backward step case).

- The outcome of the *application to the ABB nozzle problem is satisfactory*, even though time step limitations gets in the way of obtaining more than a third of all the measured values within feasible time. Large differences in the results can almost all be explained by false shock position estimations.

- The results are in some cases better than results obtained using CFD-ACE+, in some cases a little worse. Overall, OpenFOAM *does not have to fear the comparison to commercial codes*.

- For inviscid cases, we recommend using rhopSonicFoam, for viscous cases sonicTurbFoam.

- The *extensibility comes with a price*: pre- and post-processing are not very convenient. Importing a huge mesh and editing all the boundaries, which are only distinguishable by their name, can be very tedious; importing and exporting from and to third party tools is also cumbersome.

- Proper *treatment of recirculation requires special care*.

- As the two major weaknesses, the *absence of an adaptive time step* for the transient sonicTurbFoam and *false estimations for shock locations* in the more complicated flows are identified. However, thanks to the flexibility and extensibility of OpenFOAM, both points could be improved.

Based on all this, we come to the conclusion that OpenFOAM would serve well as the basis for extensions towards more complete simulations of circuit breaking.

However, it should be kept in mind that the correct prediction of shock locations is crucial for meaningful results; a type of compressible flow solver that is especially well

suited for this is the *Riemann solver*[1]. It could not just be added as another solver to OpenFOAM but would require changing and extending the core libraries, resulting in an immense work investment.

6.2 Outlook

Starting from the results achieved in this thesis, several directions for future research are thinkable, ranging from small projects to very challenging tasks:

- The simulations for the ABB nozzle case are still running, so within a few weeks these results will be complete.
- An adaptive time step for sonicTurbFoam could be implemented.
- Improved shock handling could be implemented, see the remarks about Riemann solvers above.
- A starting point for a more complete simulation could be a setting with a dynamic mesh, i. e., an actually moving plug.
- Efforts to strongly couple the Navier–Stokes equations with the Maxwell equations are already under way.
- Finally, an extension of OpenFOAM for arc simulations based on first principles, coupled with the existing flow solvers, would be a big step towards the accurate simulation of circuit breaking.

Thanks to the active and growing user base of OpenFOAM, it is to expect that the software will improve further towards a real multi-physics tool, while maintaining all the advantages of being open source.

[1] See for example Anderson (1995) for an introduction to Riemann solvers.

Acknowledgements

I would like to thank the following people (in alphabetical order):

- *Leonhard Kleiser* for agreeing, without hesitation, to supervise an external thesis and providing valuable input during our intermediate reviews.
- *Yong-Joong Lee* and *Henrik Nordborg* for extensive on-site support and guidance.
- *Carolyn Wardle-Davies* for proofreading key parts of this report. Remaining flaws are of course an oversight on my part.

References

Anderson, Dale A., Tannehill, John C., & Pletcher, Richard H. 1984. *Computational Fluid Mechanics and Heat Transfer*. New York: McGraw-Hill.

Anderson, Jr, John D. 1995. *Computational Fluid Dynamics: The Basics with Applications*. Singapore: McGraw-Hill.

Anderson, Jr, John D. 2003. *Modern Compressible Flow: With Historical Perspective*. Third edn. New York: McGraw-Hill.

Blatter, Christian. 1996. *Ingenieur Analysis 1*. Second edn. Heidelberg: Springer.

Bogar, T. J., Sajben, M., & Kroutil, J. C. 1983. Characteristic Frequencies of Transonic Diffuser Flow Oscillations. *AIAA Journal*, **21**(9), 1232–1240.

Fröhlich, Klaus. 2006 (Nov.). *Elektrische Energiesysteme: Systemtechnologie*. Script. ETH Zurich.

Fruth, Florian. 2007. *Benchmark of OpenFOAM and CFX*. Seminar thesis, ETH Zurich.

Georgiadis, N. J., Drummond, J. E., & Leonard, B. P. 1994 (Jan.). *Evaluation of Turbulence Models in the PARC Code for Transonic Diffuser Flows*. Technical memorandum 106391. NASA Lewis Research Center, Ohio.

Gerritsma, M. I. 2002 (July). *Computational Fluid Dynamics: Incompressible Flows*. Script. TU Delft.

Gremmel, Hennig, & Kopatsch, Gerald (eds). 2007. *ABB Switchgear Manual*. 11 edn. Berlin: Cornelsen.

Hirsch, Charles. 1988. *Fundamentals of Numerical Discretization*. Numerical Computation of Internal and External Flows, vol. 1. New York: Wiley.

Hirsch, Charles. 1990. *Computational Methods for Inviscid and Viscous Flows*. Numerical Computation of Internal and External Flows, vol. 2. New York: Wiley.

Hsieh, T., Wardlaw, Jr, A. B., Collins, P., & Coakley, T. 1987. Numerical Investigation of Unsteady Inlet Flowfields. *AIAA Journal*, **25**(1), 75–81.

Jasak, Hrvoje. 1996 (June). *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. Ph.D. thesis, Imperial College London.

Kim, Hong-Kyu, Park, Kyong-Yop, Im, Chang-Hwan, & Jung, Hyun-Kyo. 2003. Optimal Design of Gas Circuit Breaker for Increasing the Small Current Interruption Capacity. *IEEE Transactions on Magnetics*, **39**(3), 1749–1752.

Kundu, Pijush K., & Cohen, Ira M. 2004. *Fluid Mechanics*. Third edn. San Diego: Elsevier Academic Press.

Lindmayer, Manfred (ed). 1987. *Schaltgeräte: Grundlagen, Aufbau, Wirkungsweise*. Berlin: Springer-Verlag.

Mantilla Florez, Javier Dario. 2007 (Mar.). *Measurements and Simulations of Cold Gas Flows in Basic Gas Circuit Breaker Geometries*. Master thesis, RWTH Aachen.

OpenCFD. 2007a (Apr.). *OpenFOAM: The Open Source CFD Toolbox. Programmer's Guide Version 1.4*. OpenCFD Limited, Reading UK.

OpenCFD. 2007b (Apr.). *OpenFOAM: The Open Source CFD Toolbox. User Guide Version 1.4*. OpenCFD Limited, Reading UK.

Pope, Stephen B. 2000. *Turbulent Flows*. Cambridge UK: Cambridge University Press.

Richardson, Lewis F. 1927. The Deferred Approach to the Limit Part I. Single Lattice. *Philosophical Transactions of the Royal Society of London, Series A*, **226**, 299–361.

Roache, P. J. 1994. Perspective: A Method for Uniform Reporting of Grid Refinement Studies. *Journal of Fluids Engineering*, **116**(3), 405–413.

Salmon, J. T., Bogar, T. J., & Sajben, M. 1983. Laser Doppler Velocimeter Measurements in Unsteady, Separated, Transonic Diffuser Flows. *AIAA Journal*, **21**(12), 1690–1697.

Slater, John W. 2005 (Sept.). *CFD Verification & Validation: NPARC Alliance*. http://www.grc.nasa.gov/WWW/wind/valid/. Accessed 12th June 2007.

Smith, Howard E. 1967 (Mar.). *The Flow Field and Heat Transfer Downstream of a Rearward Facing Step in Supersonic Flow*. Technical report ARL 67-0056. Aerospace Research Laboratories, Ohio.

Versteeg, H. K., & Malalasekera, W. 1995. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Boston: Prentice Hall.

Wolter, Frank. 1997 (Oct.). *Untersuchung von CFD-Codes auf ihre Anwendbarkeit zur Gasströmungssimulation in Hochspannungs-Leistungsschaltern*. Diploma thesis, RWTH Aachen.

Yakhot, Victor, & Orszag, Steven A. 1986. Renormalization-Group Analysis of Turbulence. *Physical Review Letters*, **57**(14), 1722–1724.

A Contents of the CD

The enclosed CD contains the data created during working on this thesis. The following list describes briefly the contents of the various directories.

- *algorithm_source*: C++ source codes of the five solvers tested
- *intermediate_reports*: small reports to present intermediate results; directories labelled by date of creation
- *linux*: reference manuals for bash scripting and using vi (the editor)
- *matlab*: MATLAB scripts and functions
  - *ABB nozzle*: scripts to plot residuals for the 87 mm case, changes in Δt for the different cases and comparison of OpenFOAM results to measurements; residual log files; overview of results from Mantilla Florez (2007)
  - *backstep*: script to compare OpenFOAM and CFD-ACE+ pressure samples; script to plot residuals; residual log files and pressure data
  - *shocktube*: scripts to compare 1–3D and axi-symmetric solutions to exact solution; files with samples of T, $\mathbf{u}$ and p for different configurations; scripts to create animations of the exact solution and the resulting video files
    - *cfdace_comparison*: script to compare to CFD-ACE+ results, sample files from CFD-ACE+
    - *dt_convergence*: script to generate error plots for temporal convergence tests
    - *mesh_convergence*: script to generate error plots for spatial convergence tests
  - *supersonicwedge*: script to compare with CFD-ACE+ results; function to obtain analytical solution; script to generate analytical solution for test configuration; script to generate θ-β-Ma plot; scripts to compare OpenFOAM solution to exact solution; scripts for spatial convergence study; files with sampled Mach number values
  - *transonicnozzle*: scripts to compare OpenFOAM weak and strong solutions to experiment; script to generate diffuser geometry; files with sampled pressure and velocity data
- *openfoam*: data from the OpenFOAM case directories, contains a directory for every verification/validation case and within these, the root directories of the solvers used for the specific case; in the case directories, the *0*, *constant* and *system* directories are contained, thus everything required to perform the simulation. Results at later times are not included because of the large data volume (data produced during thesis: 29 GB)
- *presentations*: presentations from intermediate reports at ETH and ABB, final presentations at ETH and ABB

- *references*: all the references (except text books) from this thesis: journal articles, technical reports, PhD, Master and Seminar theses, scripts and manuals
- *report*: the `.tex` source files of the report and the PDF versions: `screen.pdf` with coloured hyperlinks for reading on a computer, `print.pdf` with all text in black for printing
  - *img*: all the `.eps` pictures for the report as well as pictures in other formats, as exported from OpenFOAM or required for presentations
  - *source*: MATLAB source code for inclusion in the report
- *varia*: the ABB nozzle case management spreadsheet for progress monitoring and an early draft of a mile stone plan

B MATLAB source code

B.1 The shock tube function

```
function [x_mesh,u,a,rho,T,p] = shocktube(time,p1,p4,T1,T4)
% SHOCKTUBE Analytical solution for unsteady wave motion in a shock tube.
%    [X_MESH,U,A,RHO,T,P] = SHOCKTUBE(TIME,P1,P4,T1,T4) solves the shock
%    tube problem analytically. The diaphragm is placed at 15.24 cm,
%    pressure and temperature to the left of it are P4 and T4, to the right
%    of it P1 and T1 (at time zero). The function returns X_MESH (1000
%    equally spaced points between 0 and 30.48 cm) and the mass velocity U,
%    the local speed of sound A, the density RHO, the temperature T and the
%    pressure P at time TIME for further analysis.
%
%    If only one argument is given (TIME), P1, P4, T1 and T4 are set to
%    default values.
%
%    SHOCKTUBE only treats right running shock waves and left running
%    expansion waves.
%
%    The gas in the tube is air with corresponding R and gamma; it is
%    treated as inviscid.

% ************************************************************************
% This m-file is part of the Master Thesis
%
% "Simulation and validation of compressible flow in nozzle geometries and
% validation of OpenFOAM for this application"
%
% by Benjamin Wuethrich, MSc student of Computational Science and
% Engineering at ETH Zurich.
%
% Work carried out at ABB Corporate Research in Baden-Daettwil from
% 15/04/07 until 14/09/07.
%
% Contact: benjamin.wuethrich@alumni.ethz.ch
% ************************************************************************

% Parse arguments.
if nargin==1
    % Set defaults.
    p1 = 6.897e3;   % Lower pressure (right chamber) in [Pa].
    p4 = 6.897e4;   % Higher pressure (left chamber) in [Pa].
    T1 = 231.11;    % Lower temperature (right chamber) in [K].
    T4 = 288.89;    % Higher temperature (left chamber) in [K].
elseif (nargin==5) && (p1 > p4)
    error('This would be a left-running shock wave, which is not supported.')
elseif nargin==5
    % Everything fine.
else
    error('Wrong number of arguments specified.')
end

% Set constants.
gamma = 1.4;    % Heat capacity ratio of air.
R = 287.05;     % Specific gas constant of air in [J/(kg*K)].
L1 = 0.1524;    % Initial position of the diaphragm in [m].$

% Calculate speeds of sound and densities.
a1 = sqrt(gamma*R*T1);
a4 = sqrt(gamma*R*T4);
rho1 = p1/(R*T1);
rho4 = p4/(R*T4);

% Calculate p2/p1, get p2.
p2p1 = fzero(@(p2p1)p2p1 * (1 - ((gamma-1)*(a1/a4)*(p2p1-1)) / ...
    sqrt(2*gamma*(2*gamma + (gamma+1)*(p2p1-1))) )^(-2*gamma/(gamma-1))...
```

```
64      - p4/p1,(p4/p1)/2);
65  p2 = p2p1 * p1;
66
67  % Calculate incident shock properties.
68  T2 = T1 * p2/p1 * (((gamma+1)/(gamma-1) + p2/p1) /...
69      (1 + (gamma+1)/(gamma-1)*p2/p1));
70  rho2 = rho1 * (1 + (gamma+1)/(gamma-1)*p2/p1) /...
71      ((gamma+1)/(gamma-1) + p2/p1);
72  a2 = sqrt(gamma*R*T2);
73  % Wave velocity of moving shock.
74  W = a1 * sqrt((gamma+1)/(2*gamma) * (p2/p1 - 1) + 1);
75  % Mass motion behind the wave.
76  u_p = a1/gamma * (p2/p1 - 1) * sqrt((2*gamma/(gamma+1)) /...
77      (p2/p1 + (gamma-1)/(gamma+1)));
78
79  % Pressure and velocity to the right of the expansion wave (constant
80  % across the contact surface).
81  p3 = p2;
82  u2 = u_p;
83  u3 = u_p;
84
85  % Other properties behind the expansion wave.
86  rho3 = rho4 * (p3/p4)^(1/gamma);
87  T3 = T4 * (p3/p4)^((gamma-1)/gamma);
88  a3 = sqrt(gamma*R*T3);
89
90  % Define mesh.
91  x_mesh = linspace(0,2*L1,100);
92
93  % Initialise vectors for all the quantities.
94  u = zeros(size(x_mesh));
95  a = zeros(size(x_mesh));
96  rho = zeros(size(x_mesh));
97  T = zeros(size(x_mesh));
98  p = zeros(size(x_mesh));
99
100 % Calculate boundaries of different zones.
101 % Boundary between leftmost driver gas and expansion wave.
102 x4_exp = L1 - time*a4;
103 % Boundary between expansion wave and lower pressure driver gas.
104 exp_x3 = L1 + time*(u3 - a3);
105 % Boundary between driver gas and driven gas.
106 x3_x2 = L1 + time*u_p;
107 % Location of the shock wave.
108 x2_x1 = L1 + time*W;
109
110 % Iterate through x_mesh and fill in all the quantities.
111 for i = 1:length(x_mesh)
112     if x_mesh(i) < x4_exp
113         % We are in region 4.
114         u(i) = 0;
115         a(i) = a4;
116         rho(i) = rho4;
117         T(i) = T4;
118         p(i) = p4;
119     elseif x_mesh(i) < exp_x3
120         % We are in the expansion wave.
121         [u(i),a(i),rho(i),T(i),p(i)] = expansion_wave(x_mesh(i)-L1);
122     elseif x_mesh(i) < x3_x2
123         % We are in region 3.
124         u(i) = u3;
125         a(i) = a3;
126         rho(i) = rho3;
127         T(i) = T3;
128         p(i) = p3;
129     elseif x_mesh(i) < x2_x1
130         % We are in region 2.
131         u(i) = u2;
```

```
132         a(i) = a2;
133         rho(i) = rho2;
134         T(i) = T2;
135         p(i) = p2;
136     else
137         % We are in region 1.
138         u(i) = 0;
139         a(i) = a1;
140         rho(i) = rho1;
141         T(i) = T1;
142         p(i) = p1;
143     end
144 end
145
146     function [u_exp,a_exp,rho_exp,T_exp,p_exp] = expansion_wave(x)
147         % Calculate properties within expansion wave. The expressions here
148         % are valid for -a4 <= x/time <= u3 - a3.
149         % Calculate mass velocity.
150         u_exp = 2/(gamma+1) * (a4 + x/time);
151         % Calculate speed of sound.
152         a_exp = a4 * (1 - (gamma-1)/2 * u_exp/a4);
153         % Calculate temperature.
154         T_exp = T4 * (1 - (gamma-1)/2 * u_exp/a4)^2;
155         % Calculate pressure.
156         p_exp = p4 * (1 - (gamma-1)/2 * u_exp/a4)^(2*gamma/(gamma-1));
157         % Calculate density.
158         rho_exp = rho4 * (1 - (gamma-1)/2 * u_exp/a4)^(2/(gamma-1));
159     end
160 end
```

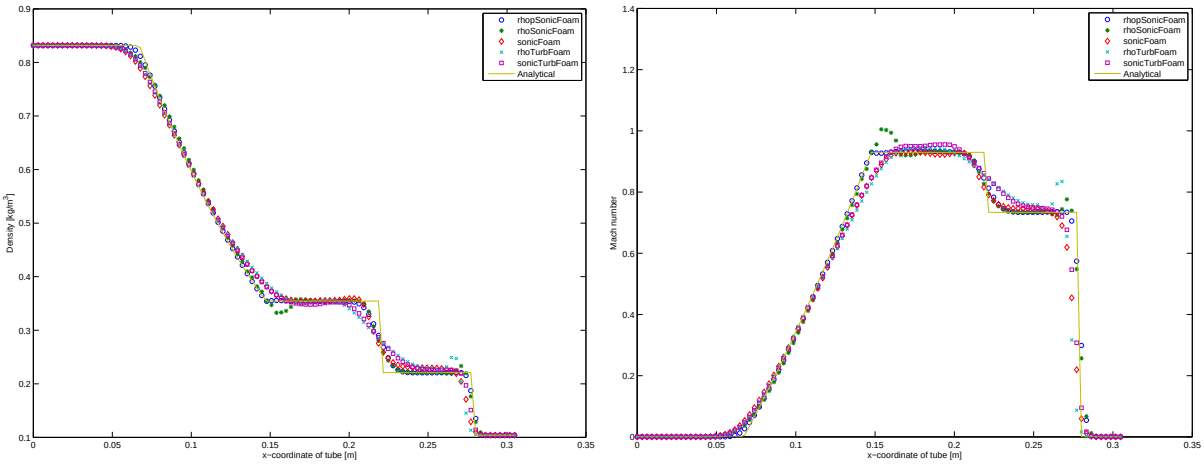
B.2 The oblique shock function

```
1  function [Ma,p,T,rho] = obliqueshock(x,y,Ma1,p1,T1,theta)
2  % OBLIQUESHOCK   Analytical solution for the oblique shock problem.
3  %    [MA,P,T,RHO] = OBLIQUESHOCK(X,Y,MA1,P1,T1,THETA) computes Mach number,
4  %    pressure, temperature and density at the coordinates given by X and Y
5  %    for an oblique shock problem specified by MA1, P1 and T1 (free flow
6  %    Mach number, pressure and temperature) and THETA (deflection angle).
7  %
8  %    THETA should be positive (otherwise, it would be an expansion problem.
9  %    If THETA is large enough for the shock to become detached, an error
10 %    comes up. If THETA is not specified, it is set to 15/180*pi radians.
11 %
12 %    If MA1, P1 and T1 are not specified, they are set to 2.5 Mach, 101'350
13 %    Pascal and 288.9 Kelvin, respectively.
14 %
15 %    If X and Y are specified such that the point is in the wedge or in the
16 %    lower half-plane, an error message is issued. Points exactly on the
17 %    shock wave are treated as upstream of the shock wave. The origin is
18 %    treated as part of the shockw wave.
19
20 % ************************************************************************
21 % This m-file is part of the Master Thesis
22 %
23 % "Simulation and validation of compressible flow in nozzle geometries and
24 % validation of OpenFOAM for this application"
25 %
26 % by Benjamin Wuethrich, MSc student of Computational Science and
27 % Engineering at ETH Zurich.
28 %
29 % Work carried out at ABB Corporate Research in Baden-Daettwil from
30 % 15/04/07 until 14/09/07.
31 %
32 % Contact: benjamin.wuethrich@alumni.ethz.ch
33 % ************************************************************************
34
35 % Parse arguments.
36 switch nargin
```

```
37      case 2
38          % Set defaults.
39          Ma1 = 2.5;          % Freestream Mach number.
40          p1 = 101.35e3;      % Pressure upstream of shock in [Pa].
41          T1 = 288.9;         % Temperature upstream of shock in [K].
42          theta = 15/180*pi;  % Deflection angle.
43      case 5
44          theta = 15/180*pi;  % Deflection angle.
45      case 6
46          % Everything fine.
47      otherwise
48          error('Wrong number of arguments specified.')
49  end
50
51  % Set constants for air.
52  R = 287.05;     % Specific gas constant of air in [J/(kg*K)].
53  gamma = 1.4;    % Heat capacity ratio of air.
54
55  % Compute rho1 from perfect gas relation.
56  rho1 = p1/(R*T1);
57
58  % Check deflection angle (is it compression or expansion?)
59  if theta < 0
60      error('This would result in an expansion wave (theta is negative).')
61  elseif theta >= pi/2
62      error('theta cannot be equal or larger than pi/2.')
63  end
64
65  % Calculate and check shock wave angle using the beta-theta-Mach relation.
66  lambda = sqrt((Ma1^2 - 1)^2 - 3*(1 + (gamma-1)/2*Ma1^2)*...
67      (1 + (gamma+1)/2*Ma1^2)*tan(theta)^2);
68  chi = ((Ma1^2 - 1)^3 - 9*(1 + (gamma-1)/2*Ma1^2)*...
69      (1 + (gamma-1)/2*Ma1^2 + (gamma+1)/4*Ma1^4)*tan(theta)^2) / lambda^3;
70  beta = atan((Ma1^2 - 1 + 2*lambda*cos((4*pi + acos(chi))/3)) /...
71      (3*(1 + (gamma-1)/2*Ma1^2)*tan(theta)));
72  if (beta < 0 || ~isreal(beta))
73      error('The shock is detached, choose a smaller theta or a larger Ma1.')
74  end
75
76  % Check if the given point is upstream or downstream of the shock wave (or
77  % in a nonsensical position).
78  [phi,r] = cart2pol(x,y+eps);
79  if (phi < theta && r > 0)
80      % The point lies IN the wedge or in the lower half-plane.
81      error(['The specified coordinates belong to a point IN the wedge ',...
82          'or in the lower half-plane.'])
83  elseif (phi < beta && r > 0)
84      % The point is downstream of the shock wave.
85      Ma_n1 = Ma1 * sin(beta);    % Normal component of upstream Ma number.
86      rho = rho1 * (gamma + 1) * Ma_n1^2 / ((gamma - 1)*Ma_n1^2 + 2);
87      p = p1 + p1*2*gamma / (gamma + 1) * (Ma_n1^2 - 1);
88      Ma_n2 = sqrt((Ma_n1^2 + 2/(gamma-1)) / (2*gamma / (gamma-1) * Ma_n1^2 - 1));
89      T = T1 * p/p1 * rho1/rho;
90      Ma = Ma_n2 / sin(beta - theta);
91  else
92      % The point is upstream of the shock wave or at the origin.
93      Ma = Ma1;
94      rho = rho1;
95      p = p1;
96      T = T1;
97  end
```

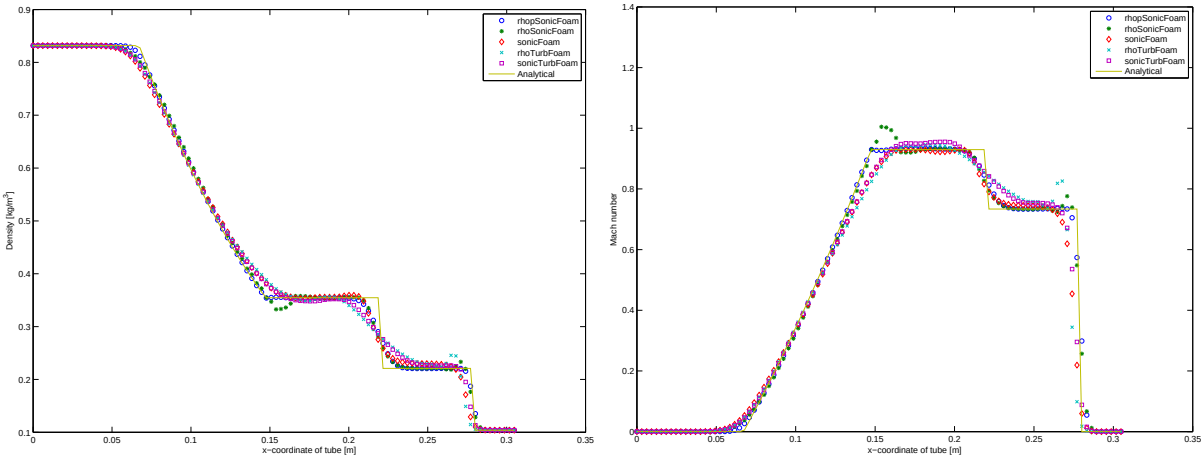
C Additional results

C.1 Shock tube plots from the solver quality evaluation



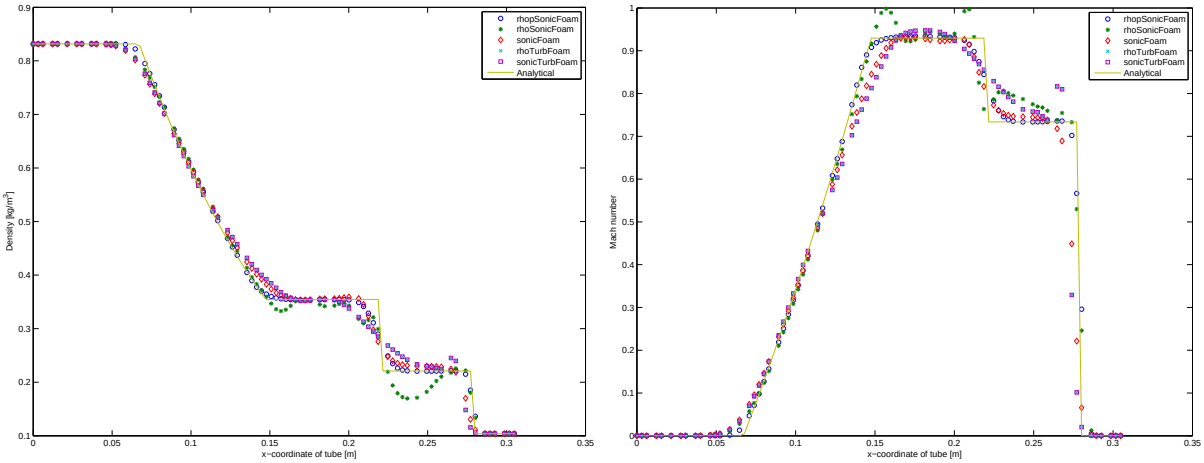
(a) Density distribution **(b)** Mach number distribution

Figure C.1: Density and Mach number distribution at $t = 2.5 \cdot 10^{-4}$ s: comparison of OpenFOAM solvers for the 1D case.



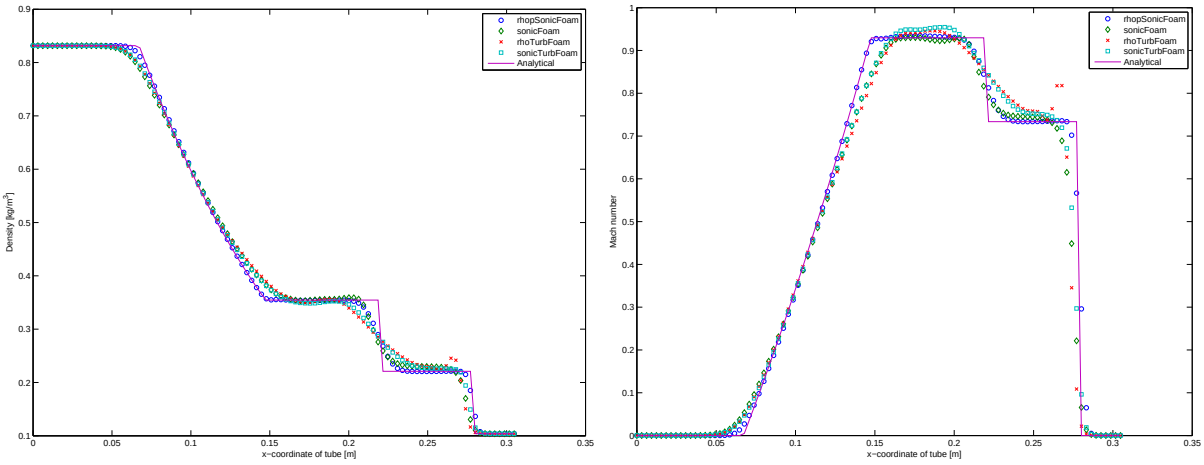
(a) Density distribution **(b)** Mach number distribution

Figure C.2: Density and Mach number distribution at $t = 2.5 \cdot 10^{-4}$ s: comparison of OpenFOAM solvers for the 2D case.



(a) Density distribution **(b)** Mach number distribution

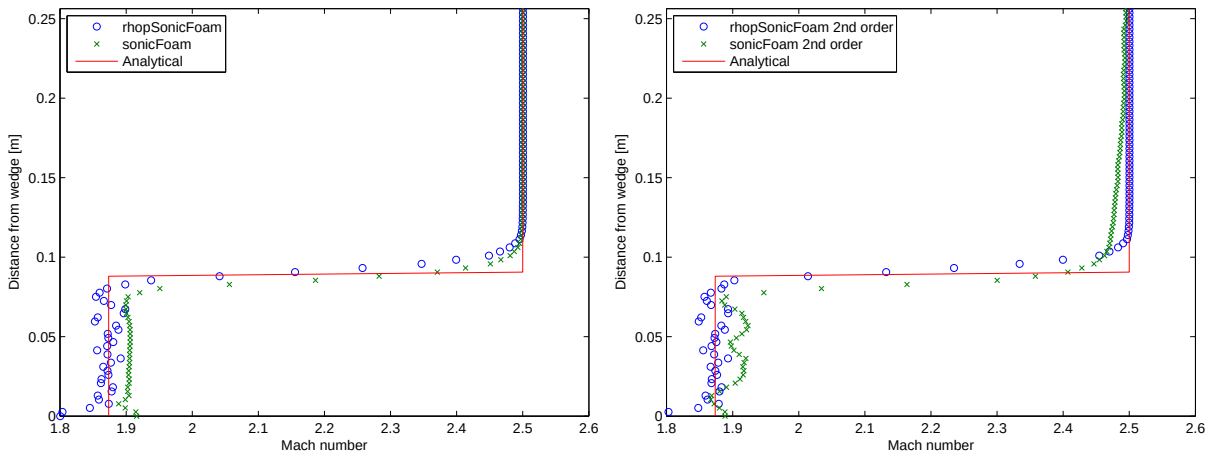
Figure C.3: Density and Mach number distribution at $t = 2.5 \cdot 10^{-4}$ s: comparison of OpenFOAM solvers for the axi-symmetric case.



(a) Density distribution **(b)** Mach number distribution

Figure C.4: Density and Mach number distribution at $t = 2.5 \cdot 10^{-4}$ s: comparison of OpenFOAM solvers for the 3D case.

C.2 Supersonic wedge plots from the solver quality evaluation



(a) **(b)**

Figure C.5: Comparison of laminar solver perpendicular samples to analytical solution: (a) 1st order spatial discretisation; (b) 2nd order spatial discretisation