



HAL
open science

UnDiFi-2D: an Unstructured Discontinuity Fitting code for 2D grids

Lorenzo Campoli, Alessia Assonitis, Mirco Ciallella, Renato Paciorri, Aldo Bonfiglioli, Mario Ricchiuto

► **To cite this version:**

Lorenzo Campoli, Alessia Assonitis, Mirco Ciallella, Renato Paciorri, Aldo Bonfiglioli, et al.. UnDiFi-2D: an Unstructured Discontinuity Fitting code for 2D grids. *Computer Physics Communications*, 2021, 271, pp.108202. 10.1016/j.cpc.2021.108202 . hal-03408117

HAL Id: hal-03408117

<https://inria.hal.science/hal-03408117>

Submitted on 28 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UnDiFi-2D: an Unstructured Discontinuity Fitting code for 2D grids.

L. Campoli^a, A. Assonitis^b, M. Ciallella^c, R. Paciorri^b, A. Bonfiglioli^d, M. Ricchiuto^c

^a*Saint-Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, Russia*

^b*Dipartimento di Ingegneria Meccanica e Aerospaziale, Sapienza University, Rome, Italy*

^c*Team CARDAMOM, INRIA, Univ. Bordeaux, CNRS, Bordeaux INP, IMB, UMR 5251, 200 Avenue de la Vieille Tour, 33405 Talence cedex, France*

^d*Scuola di Ingegneria, Università degli Studi della Basilicata, Potenza, Italy*

Abstract

UnDiFi-2D, an open source (free software) *Unstructured-grid, Discontinuity Fitting* code, is presented. The aim of *UnDiFi-2D* is to model gas-dynamic discontinuities in two-dimensional (2D) flows as if they were *true* discontinuities of null thickness that bound regions of the flow-field where a smooth solution to the governing PDEs exists. *UnDiFi-2D* therefore needs to be coupled with an unstructured CFD solver that is used to discretize the governing PDEs within the smooth regions of the flow-field. Two different, in-house developed, CFD solvers are also included in the current distribution.

The main features of the *UnDiFi-2D* software can be summarized as follows:

Programming Language *UnDiFi-2D* is written in standard Fortran 77/95; its design is highly modular in order to enhance simplicity of use, maintenance and allow coupling with virtually any existing CFD solver;

Usability, Maintenance and Enhancement In order to improve the usability, maintenance and enhancement of the code also the documentation has been carefully taken into account. The `git` distributed versioning system has been adopted to facilitate collaborative maintenance and code development;

Copyrights *UnDiFi-2D* is a free software that anyone can use, copy, distribute, change and improve under the GNU Public License version 3.

The present paper is a manifesto of the first public release of the *UnDiFi-2D* code. It describes the currently implemented features, which are the result of more than a decade of still ongoing CFD developments. This work is focused on the computational techniques adopted and a detailed description of the main characteristics is reported. *UnDiFi-2D* capabilities are demonstrated by means of examples test cases. The design of the code allows to easily include existing CFD codes and is aimed at ease code reuse and readability.

Keywords: CFD, Fortran, C, shock-fitting, shock-capturing,, unstructured grids

Contents

1	Introduction	4
1.1	Background	4
1.2	Related software packages	7
1.3	Motivations and aims	8
1.4	Documentation	8
1.5	Collaborative framework	8
1.6	<i>UnDiFi-2D</i> : general description of the directory tree	8
1.7	Copyrights	9
1.8	Paper layout	9
2	Fluid flow model and CFD codes	10
2.1	Governing equations	10
2.2	Shocks, Rankine-Hugoniot relations, and weak solutions	11
2.3	Residual distribution methods	12
3	Unstructured Shock-Fitting: Algorithmic Features	15
3.1	Cell Removal Around the Shock Front	19
3.2	Local Re-Meshing Around the Shock Front	19
3.3	Calculation of the Unit Vectors Normal to the Shock Front	21
3.4	Solution Update Using the Shock-Capturing Code	22
3.5	Enforcement of the Jump Relations	22
3.6	Shock Displacement	23
3.7	Interpolation of the Phantom Nodes	24
4	Test cases	25
4.1	Hypersonic flow past a Circular cylinder (<i>CircularCylinder-1</i>)	26
4.2	Interaction between two shocks of opposite families (<i>SSInteraction1-2</i>)	27
4.3	Interaction between two shocks of the same families (<i>SSInteraction2-1</i> , <i>SSInteraction2-2</i>)	29
4.4	Shock-wall interaction: regular reflection. (<i>RegularReflection-1</i> , <i>RegularReflection-2</i>) 31	
4.5	Steady Mach reflection (<i>MachReflection-1</i> , <i>MachReflection-2</i>)	33
4.6	Planar source flow (<i>Q1D</i>)	35
4.7	Shock formation due to the coalescence of compression waves (<i>CoaSHCK</i>)	37
4.8	Transonic flow past a NACA0012 profile (<i>NACA0012_M080_A0</i>)	38
4.9	Shock-vortex interaction (<i>ShockVortex</i>)	39

*Please address correspondence to L. Campoli

Email addresses: l.kampoli@spbu.ru (L. Campoli), alessia.assonitis@uniroma1.it (A. Assonitis), mirco.ciallella@inria.fr (M. Ciallella), renato.paciorri@uniroma1.it (R. Paciorri), aldo.bonfiglioli@unibas.it (A. Bonfiglioli), mario.ricchiuto@inria.fr (M. Ricchiuto)

5	Concluding remarks and future perspectives	42
5.1	Key features	43
5.2	Known issues	43
5.3	Perspectives	44

Program summary

Program title: *UnDiFi-2D*

Catalogue identifier:

Program summary URL:

Program obtainable from: CPC Program Library, Queen’s University, Belfast, N. Ireland

Licensing provisions: GNU General Public Licence, version 3

No. of lines in distributed program, including test data, etc.: 380222

No. of bytes in distributed program, including test data, etc.: 3734808

Distribution format: github repository

Programming language: Fortran; developed and tested with Intel Fortran Compiler v. 18.0.3 and GNU gfortran.

Computer: Any computer system with a Fortran compiler is suited.

Operating system: designed for POSIX architecture and tested on GNU/Linux one.

Has the code been vectorized or parallelized?: no.

Classification:

External routines: the code depends on several libraries and third-party packages which are detailed in the corpus of the text.

Nature of problem: numerical computation of flows with discontinuities.

Solution method: shock-fitting technique.

Restrictions: At present, *UnDiFi-2D* is validated for inviscid steady and unsteady two-dimensional flows without changes in the number of discontinuity lines and interaction points.

Unusual features: *UnDiFi-2D* implements a shock-fitting algorithm and can be coupled with unstructured cell-vertex solvers, with an Arbitrary Lagrangian-Eulerian (ALE) formulation.

Additional comments: *UnDiFi-2D* project adopts `git` [1], a free and open source distributed version control system. A public repository dedicated to *UnDiFi-2D* project [2] has been created on `github` [3], a web-based hosting service for software development projects using `git` versioning system. Finally, a comprehensive documentation is provided in the form of user manual developed in Pandoc [4].

References

- [1] Git, a free and open source distributed version control system, <http://git-scm.com>.
- [2] *UnDiFi-2D* documentation, <https://github.com/UnDiFi/UnDiFi-2D/wiki>.
- [3] Github, a web-based hosting service for software development projects using git versioning system, <https://github.com>.
- [4] Dominici, M. (2014). An overview of Pandoc. *TUGboat*, 35(1), 44-50.

1. Introduction

1.1. Background

Open-source Computational Fluid Dynamics (CFD) tools are nowadays gaining increasing popularity among both CFD users and developers. Software packages like **OpenFoam** [1] and **SU2** [2], to name the two most popular ones, are not just CFD codes, but general-purpose multi-physics packages capable of simulating several fluid-related phenomena typically encountered in a wide range of engineering applications. Access to the source code makes them appealing also to CFD developers, because the cost of learning how to code new algorithms inside an existing infrastructure can be significantly less (and a *una-tantum* effort) than that required for developing a new in-house code from scratch.

Regardless of whether commercial or open-source codes are used, when it comes to simulating high-speed compressible flows, all CFD codes rely upon the so-called shock-capturing approach to model gas-dynamic discontinuities, such as shock-waves and slip-lines. Shock-capturing, which can be traced back to the work by von Neumann and Richtmyer [3] lays its foundations in the mathematical theory of weak solutions, which allows to compute all kind of flows, including those affected by shock-waves, using the same discretization of the conservation-law-form of the governing equations at all grid cells. This yields obvious consequences in terms of coding simplicity, since the same set of operations is repeated within all control volumes of the mesh, no matter how complicated the flow might be. Coding simplicity comes not for free, however, and shock-capturing calculations of shocked-flows are plagued by several troubles, all of which are rooted to the fact that “the thickness of a von Neumann shock is unacceptable” [4].

Whether it was “von Neumann’s reputation [that] helped the shock-capturing myth to become a religion” [4] or coding simplicity, shock-capturing modeling of gas-dynamic discontinuities is the *de-facto* standard in modern CFD codes and has overshadowed an even older shock-modeling technique, which dates back to a World War II NACA technical report by Emmons [5], see also [6]. Starting in the late 1960s [7], Emmons’ technique has been re-discovered and further developed by Gino Moretti and co-workers under the name of “shock-fitting” and, since the 1980s [8, 9, 10], by James Glimm and co-workers under the name of “front-tracking”.

Fitting/tracking methods consist in first locating and then tracking the motion of the discontinuities, which are treated as boundaries between regions where a smooth solution to the governing partial differential equations (PDEs) exists. The flow variables on the two sides of the discontinuities can be analytically computed by using the Rankine-Hugoniot (R-H)

jump relations, algebraic equations connecting the states on both sides of the discontinuity and its local speed. Then, this solution is used to compute the space-time evolution of the discontinuity, that is, to track its motion.

Shock-fitting and front-tracking methods share many similarities and differ primarily in the range of applications addressed, with shock-fitting leaning toward gas-dynamics and front-tracking toward other kind of interfaces. We shall hereafter primarily refer to shock-fitting methods, simply because Gino Moretti has been very influential in the Italian academic community, despite the fact that most of his professional career took place in the Americas.

In the early days of the CFD era, when computers were very slow compared to present-day standards, shock-fitting methods enjoyed a remarkable popularity, because [11] “the more discontinuities are fitted, the fewer grid points are needed”. In addition to this, shock-fitting methods are immune from most of the numerical troubles (such as carbuncles [12] and accuracy degradation downstream of a captured shock [13]) incurred by shock-capturing. Despite these clearly recognized advantages [14, 15], the tracking of the shocks and their interactions raise a number of topological and logical problems that make shock-fitting algorithms harder to code than shock-capturing ones. Over the years, algorithmic complexity seems to have alienated the majority of CFD practitioners from shock-fitting or, put in Moretti’s words [11]: “Analysts who are more familiar with calculus than logic shy away from shock-fitting and prefer to pay the price of inaccuracy”.

As a consequence, only a handful of research teams are nowadays still developing and using shock-fitting algorithms. To the best of the authors’ knowledge, and apart from isolated examples, the only groups that in recent years have published a significant amount of literature on the subject are: *i*) Xiaolin Zhong and co-workers from UCLA, who perform high-order shock-fitting DNS studies of high-speed flows [16]; *ii*) the LLNL group headed by Tariq Aslam, who uses high-order shock-fitting methods to simulate explosions [17] and *iii*) Marcello Onofri and Francesco Nasuti [18] from the University of Rome, “La Sapienza”, who developed a second-order-accurate, multi-block, shock-fitting code to simulate the flow through propulsive nozzles. The aforementioned references are clearly not exhaustive, but should only be considered as entry points to a much larger body of literature.

All three aforementioned research teams developed shock-fitting algorithms using structured meshes and the use of structured meshes is deemed responsible for at least some of the algorithmic difficulties encountered when coding shock-fitting algorithms; see for instance reference [19] which details the implementation of high-order differencing schemes within the shock-fitting framework.

Taking advantage of the gradual shift that has taken place in the CFD community from structured towards unstructured grids, two of the authors of this paper started developing a shock-fitting algorithm that inherits features of Moretti’s technique while taking advantage of the geometrical flexibility offered by the use of unstructured triangular and tetrahedral meshes. The unstructured shock-fitting technique described in this paper made its first journal appearance in 2009 [20]; at that time the algorithm was capable of simulating steady, two-dimensional flows featuring only one “fitted” shock-wave. Multiple shocks could be handled using a hybrid approach, whereby only one shock was fitted and all other discontinuities

and their mutual interactions were captured. Later developments [21, 22] made the algorithm capable of fitting contact discontinuities (beside shocks), shock–shock and shock–wall interactions. An order-of-accuracy analysis of the steady, two-dimensional scheme has been conducted in [23] showing that shock-fitting allows to preserve the design order of the spatial discretization scheme within the entire shock-downstream region. The unstructured, shock-fitting algorithm was also used to simulate real-gas effects in hypersonic, two-dimensional steady flows [24, 25], whereas time-accurate simulations of two-dimensional, unsteady flows are reported in [26, 27]. Finally, the technique has been recently used to simulate two-dimensional, laminar and turbulent viscous flows featuring shock-wave/boundary-layer interactions [28].

A significant number of international collaborations have been established during the almost fifteen years of algorithmic development.

The first collaboration was established with Mikhail Ivanov of Itam, Russian Academy of Sciences. It dealt with the numerical study of weak Mach reflections by means of the shock-fitting technique and led to the development of the numerical model for the treatment of the triple points [21].

Joint work [25] with Andrea Lani, at that time at the Von Karman Institute for Fluid Dynamics (VKI) in Belgium, led to a C++ version [29] of the shock-fitting algorithm described in this paper and its coupling with the public-domain CFD code CoolFluid [30]. This activity showed that the modular approach adopted in the shock-fitting code allows to plug in different gas-dynamic solvers with a limited coding effort.

The ongoing collaboration with Mario Ricchiuto at INRIA Bordeaux focused, at first, on the coupling between the shock-fitting algorithm described in this paper and yet another CFD code, *NEO*, with the aim of improving the capability of the shock-fitting algorithm to deal with unsteady flows [26, 27]. More recent joint work borrows ideas from Shifted Boundary Methods (SBM) [31] and has led to what we call “Extrapolated Discontinuity Tracking” (EDiT). EDiT has a number of distinctive features compared to the shock-fitting technique described in this paper; for this reason it is not currently included in the distribution and the interested reader is referred to [32] for details.

Even though all the developments described so far apply to two-dimensional flow configurations, a three-dimensional version of the algorithm appeared in 2013 in a journal publication [33]; it is capable of dealing with steady flows featuring one or more fitted shocks but, in contrast to the two-dimensional case, the interaction between different shocks cannot be “fitted”, but only “captured”. The aforementioned limitation is expected to be overcome thanks to an on-going collaboration with Carl Ollivier-Gooch from the University of British Columbia in Vancouver. The algorithm that he has been developing [34, 35, 36] for inserting a surface as an internal boundary into existing unstructured meshes, although originally motivated by different applications, naturally finds its way in the present shock-fitting algorithm for the reasons that will be clarified in Sect. 3.

Finally, Lorenzo Campoli, who contributed to the development of the time-accurate version of the algorithm [26, 27] while he was a PhD student at the University of Rome, “La Sapienza”, now based at St. Petersburg State University is the promoter of the project described in the present paper.

Apart from the aforementioned collaborations, the increasing awareness within the CFD community of the benefits offered by shock-fitting is confirmed by the increasing number of research teams that, over the last few years, have been actively developing shock-fitting/tracking methods using unstructured grids.

The group headed by Jun Liu at Dalian University of Technology, China, has developed a Mixed Capturing and Fitting Solver (MCFS) by combining a shock-fitting algorithm, in many respects similar to the one described here, with an existing shock-capturing, cell-centered Finite Volume (FV) solver [37, 38, 39, 40].

Shock-fitting/front-tracking ideas made their way also through the Finite Element community. We refer to the SUPG technique of [41] and the Discontinuous Galerkin (DG) Finite Element methods (FEM) independently developed by two different research teams: [42, 43] and [44, 45]. All three aforementioned techniques simultaneously solve for the location of the grid-points, in addition to the flow-variables, so as to constrain certain edges of the tessellation to be aligned with the discontinuities. The use of shape-functions that are continuous across the element interfaces, which is the case with SUPG, or discontinuous, such as in DG, has implications on how discontinuities are fitted. Similarly to the algorithm described in this paper, in the SUPG-FEM of [41] the discontinuities are internal boundaries of zero thickness: by doing so, a finite jump in the dependent variables can take place while crossing the discontinuity. On the other hand, numerical methods that employ a data representation which is discontinuous across the cell interfaces, which is the case with DG-FEM, but also with cell-centred FV methods, allow to fit discontinuities as a collection of edges of the mesh, without introducing internal boundaries.

The rising interest towards shock-fitting/front-tracking techniques for simulating compressible flows gave rise to the idea of making our shock-fitting code publicly accessible in order to further promote its collaborative development.

In its current stage, the project combines contributions from research teams at four different research institutions and is willing to further extend the list of contributors.

The present and future versions of the code can be downloaded in the open-source repository made available at <https://github.com/UnDiFi/UnDiFi-2D>. At present, only the two-dimensional version of the algorithm is going to be released, because it is the one that has been most actively developed over the years and has reached a sufficient level of maturity and generality.

1.2. Related software packages

As already mentioned, a C++ implementation of the shock-fitting algorithm described in this paper, has been developed in collaboration with the Von Karman Institute for Fluid Dynamics (VKI). It is described in [29] and it has been made publicly accessible at <https://github.com/andrealani/ShockFitting>. At the time of writing, however, it appears to be not yet fully operational.

The FronTier++ library package [46], developed and maintained at Stony Brook University, demonstrates the capabilities achieved by the decades long activity in developing front-tracking algorithms by James Glimm and co-workers [8, 47, 48, 49, 10, 50]. According

to the FronTier++ website [46], the library provides "high resolution tracking for contact discontinuities and internal boundaries in continuum medium simulations".

An extensive set of sample problems¹, both in two and three space dimensions, shows the wide range of applications that can be dealt with using FronTier++.

Moreover, according to [51], the front-tracking algorithm has been inserted in the plasma-physics code FLASH [52].

1.3. Motivations and aims

The *UnDiFi-2D* code has been developed with the aim of having an effective method to definitely cure most if not all the issues and pathologies affecting traditional shock-capturing solutions and to satisfy the following requirements:

- it is publicly and freely available;
- it is well documented;
- it allows easy maintenance and enhancement within a collaborative framework;
- it is actively maintained and developed.

1.4. Documentation

Free, open source softwares have often poor documentation. This lack compromises the diffusion of this kind of codes and makes their usage very difficult. Since *UnDiFi-2D* has also a didactic purpose, its easiness and accessibility is taken into account by shipping it with an in-depth user-manual.

1.5. Collaborative framework

Nowadays, one of the keys for the success of a free software is its capability to be easily maintained and improved within a collaborative framework. For this reason, the *UnDiFi-2D* code adopts `Git`² as distributed versioning system and a public repository dedicated to the *UnDiFi-2D* project has been created on `github`³. It facilitates the tracking of each modification while, by means of repositories, a worldwide collaboration is fostered.

1.6. *UnDiFi-2D*: general description of the directory tree

The main directory `UnFiDi-2D` contains the following sub-directories:

1. `bin`: where all the executables are installed;
2. `lib`: where various libraries and their source codes are stored;
3. `doc`: which contains the documentation;

¹<http://www.ams.sunysb.edu/~linli/FTruns/index.html>

²Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency, see <https://git-scm.com>.

³Github is a web-based hosting service for software development projects using git versioning system and it is free for open source projects, see <https://github.com>.

4. **source**: where the source files of the *UnDiFi-2D* code are stored;
5. **source_utils**: contains
 - the source files of various I/O format converters;
 - the **Triangle** [53, 54] mesh-generator;
6. **tests**: contains the various test-cases described in Sect.4;
7. **tools**: contains the source code of the **f77split** and **f90split** programs [55];
8. **EulFS.3.7**: where the source files of the **EulFS** [56, 57] gas-dynamic solver are stored.
9. **NEO**: where the source files of the **NEO** solver [58, 59, 60] are stored;

In addition to these sub-directories, the main directory contains the script (`compile_all.sh`) which compiles all the software packages.

Full description on how to download, compile and run the code can be found at the documentation page <https://github.com/UnDiFi/UnDiFi-2D/wiki>.

The directory tree highlights the fact that the software is made up of three key components:

1. the shock-fitting module *UnDiFi-2D* which handles the motion of the discontinuities and their interactions (if any), but also drives the other two components, i.e.
2. the gas-dynamic solver, either **EulFS** or **NEO**, which is used to discretize the governing PDEs in smooth regions of the flow-field;
3. the meshing software **Triangle**, which is used to locally re-mesh while the discontinuities move throughout the computational domain.

Communication among the driver *UnDiFi-2D*, the gas-dynamic solver and the meshing software is handled using format converters (to be found in the **source_utils** folder) that rely on disk I/O. This programming approach is certainly not the best from the standpoint of computational efficiency, one of the reasons being that one has to switch among the different data-structures used by the three different modules. However, this approach is very convenient, since it allows us to use off-the-shelf gas-dynamic solvers and mesh generation tools that are treated as black boxes and can be replaced by similar ones only by changing the format converters, with a modest coding effort.

1.7. Copyrights

UnDiFi-2D is a free software. The authors encourage anyone to use, copy, distribute, study, change and improve the code. *UnDiFi-2D* is distributed under the GNU Public License version 3. Users are kindly requested to cite the present paper when publishing results obtained by means of *UnDiFi-2D*.

1.8. Paper layout

The present article is organized as follows. The mathematical model and the numerical methods adopted are presented in Sect. 2. The unstructured shock-fitting algorithm is described in Sect. 3. A selected set of representative numerical results is given in Sect. 4, whereas Sect. 5 addresses some currently unsolved issues and presents ongoing work aimed at overcoming these limitations.

2. Fluid flow model and CFD codes

This section is devoted to the description of the compressible flow equations solved in *UnDiFi-2D*, as well as of the numerical methods implemented in the CFD solvers included in the distribution. We will detail these aspects sufficiently for the reader to appreciate the numerical results discussed in Sect 4. For more information, the reader is referred to the extensive bibliography provided hereafter, as well as to the documentation included with the distributed codes.

2.1. Governing equations

The current version of *UnDiFi-2D* allows to simulate compressible, non-viscous, non-heat-conducting, perfect gases modelled by the Euler equations which can be written in conservative form as:

$$\mathbf{U}_t + \nabla \cdot \mathcal{F} = 0 \quad (1)$$

where, U and \mathcal{F} denote the arrays of conservative variables and fluxes defined as:

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho E \\ \rho \mathbf{u} \end{pmatrix}, \quad \mathcal{F} = \begin{pmatrix} \rho \mathbf{u} \\ \rho \mathbf{u} H \\ \rho \mathbf{u} \mathbf{u} + p \mathbf{I} \end{pmatrix} \quad (2)$$

with standard notation for all physical variables. The numerical methods implemented in the CFD codes distributed in *UnDiFi-2D* often require the use of the flux Jacobians. If d denotes the number of space dimensions, given a direction $\boldsymbol{\xi} \in \mathbb{R}^d$, it is useful to define the matrix:

$$\mathbf{K}_{\boldsymbol{\xi}}^{\mathbf{U}} = \sum_{i=1}^d \mathbf{A}_i^{\mathbf{U}} \xi_i \quad \text{where} \quad \mathbf{A}_i^{\mathbf{U}} = \frac{\partial \mathcal{F}_i}{\partial \mathbf{U}} \quad (3)$$

The hyperbolic character of (1) guarantees that $\mathbf{K}_{\boldsymbol{\xi}}$ admits a set of real eigenvalues, with a set of eigenvectors which are linearly independent almost everywhere. In particular, one can show that:

$$\mathbf{K}_{\boldsymbol{\xi}}^{\mathbf{U}} = \mathbf{R}_{\boldsymbol{\xi}}^{\mathbf{U}} \boldsymbol{\Lambda}_{\boldsymbol{\xi}} \mathbf{L}_{\boldsymbol{\xi}}^{\mathbf{U}} \quad (4)$$

having denoted by $\boldsymbol{\Lambda}_{\boldsymbol{\xi}}$ the diagonal matrix of the eigenvalues, and with $\mathbf{R}_{\boldsymbol{\xi}}^{\mathbf{U}}$ and $\mathbf{L}_{\boldsymbol{\xi}}^{\mathbf{U}} = (\mathbf{R}_{\boldsymbol{\xi}}^{\mathbf{U}})^{-1}$ the matrices whose columns/rows represent the corresponding eigenvectors. For system (1)-(2) the eigenvalues can be shown to be given by the wave speeds:

$$\boldsymbol{\Lambda}_{\boldsymbol{\xi}} = \text{diag}(\mathbf{u} \cdot \boldsymbol{\xi}, \mathbf{u} \cdot \boldsymbol{\xi}, \mathbf{u} \cdot \boldsymbol{\xi}, \mathbf{u} \cdot \boldsymbol{\xi} + a, \mathbf{u} \cdot \boldsymbol{\xi} - a) \quad (5)$$

For brevity, we omit the explicit form of the eigenvectors which can be computed by standard means, see e.g. [61]. The above eigenvalues represent the components along $\boldsymbol{\xi}$ of the wave speeds carrying the information in the flow. If $\boldsymbol{\xi}$ is the velocity direction, one recovers the well known key role played by the Mach number $\|\mathbf{u}\|/a$ in determining whether all information travels downstream, or some of it goes back upstream via the acoustic wave travelling at speed $\mathbf{u} \cdot \boldsymbol{\xi} - a$. This is one of the acoustic Riemann invariants, will be referred to as the negative acoustic Riemann invariant [61, 62], and denoted by R^- .

2.2. Shocks, Rankine-Hugoniot relations, and weak solutions

An important property of system (1) is that, as all nonlinear conservation laws, discontinuous solutions may occur in finite times even with smooth initial/boundary conditions. In this case, the differential form of the model is not the relevant one. Weak discontinuous solutions are defined as solutions to the differential form in the sub-domains empty of discontinuities, connected across each point of the discontinuity by the Rankine-Hugoniot (or jump) relations: [62, 61, 63]

$$w_n \llbracket \mathbf{U} \rrbracket = \llbracket \mathcal{F} \rrbracket \cdot \mathbf{n} \quad (6)$$

where by $\llbracket \cdot \rrbracket$ we have denoted the jump of a quantity, with $\mathcal{F} \cdot \mathbf{n}$ the flux in the direction normal to the discontinuity, and with w_n the speed of the discontinuity along the direction \mathbf{n} .

Compressible flows modelled by the Euler system develop discontinuities of two kinds:

- shock waves for which the flow goes into the discontinuity. The (relative) upstream Mach number is higher than unity, so in (6) all the upstream quantities are given data. The (relative) downstream Mach number is lower than one, which means that the downstream negative acoustic Riemann invariant is also transported toward the discontinuity. So constantness of R^- , together with Eq. (6), allows to have a nonlinear algebraic system with enough relations to compute the downstream state, and the shock speed;
- slip-lines (or slip-streams) with respect to which the (relative) normal flow speed is zero. In this case one finds trivially that there is no jump in the pressure. The main unknowns are in this case the (unique) value of the pressure and of the discontinuity speed/normal flow component which are compatible with the given data, and in particular with the two acoustic Riemann invariants transported into the shock on either side of the discontinuity, thus again this is a closed algebraic system.

In simple flow configurations, such as those addressed in Sect. 4.2, 4.3 and 4.4, the above rules allow to compute exact solutions to discontinuous flows.

The resulting solutions can be also characterized in terms of an entropy which is transported along the streamlines and has a finite jump across discontinuities. In particular, for steady flows with homo-entropic boundary conditions, we know that the entropy is constant before the discontinuity, and conserved along streamlines after the shock. Moreover, entropy increases along physically acceptable discontinuities (see e.g. [64, 62]). For the Euler equations with perfect gas equation of state a possible definitions of entropy is:

$$S = \frac{p}{\rho^\gamma} + S_0 \quad (7)$$

with γ the ratio of the specific heats at constant pressure/volume, and S_0 a reference value depending on the boundary/initial conditions.

2.3. Residual distribution methods

As noted in Sect. 1.6, *UnDiFi-2D* has been coupled to two different codes, **EulFS** and **NEO**, bearing many similarities in terms of numerical methods used, and both included in the *UnDiFi-2D* repository.

EulFS and **NEO** provide different implementations of a class of numerical methods known as Residual Distribution (RD) or Fluctuation Splitting (FS) schemes [65, 66]. In its most classical formulation, the RD approach provides discrete approximations of the compressible Euler equations on simplicial grids, starting from values of the dependent variables stored at the vertices of the mesh. The second order variant of the methods exploits a classical continuous piece-wise linear finite element interpolation of the unknowns, for which we will use the standard notation \mathbf{U}^h , with h the mesh size.

The main idea behind these methods is summarized in Fig. 1: discrete equations for the steady state values of the unknowns are assembled over each element. The steady solution can be obtained as the limit of the (pseudo-)time iteration:

$$|C_i| \frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} + \sum_{T \ni i} \Phi_i^T = \text{B.C.s} \quad (8)$$

with Δt a (pseudo-)time step, and $|C_i|$ usually taken as the area of the median dual obtained joining the gravity centers of the cells surrounding i to the mid-points of the edges running into the node (the polygonal-shaped boundaries in Fig. 1). The *nodal* fluctuations (or nodal residuals) verify in each triangle, T the consistency constraint:

$$\sum_{j \in T} \Phi_j^T = \Phi^T := \int_T \nabla \cdot \mathcal{F}(\mathbf{U}^h) dV \quad (9)$$

In Eq. (8) the right hand side represents the boundary condition terms, which are left out of the discussion. Several design criteria exist for (8)-(9), impacting both the practical evaluation of the *element* fluctuation (or residual) Φ^T , as well as the definition of the nodal residuals (8). Among the most important we mention:

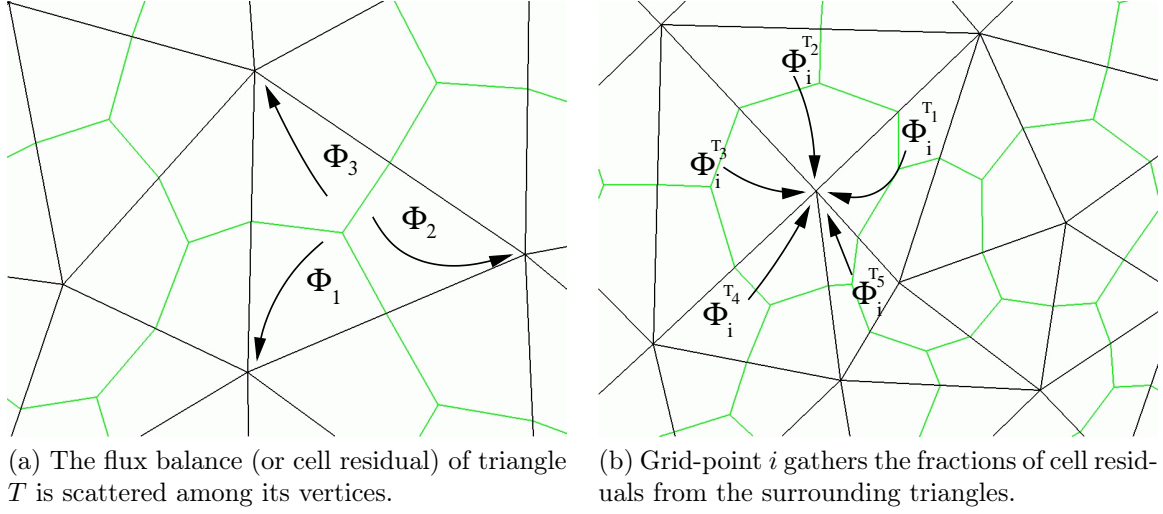


Figure 1: Residual distribution concept.

Conservation The consistency condition (9) implies conservation if the evaluation of the element residual is such that the equality:

$$\Phi^T = \oint_{\partial T} \mathcal{F}(\mathbf{U}^h) \cdot \mathbf{n} dS \quad (10)$$

holds for some continuous approximation of the normal flux. The condition above can be shown to be essential for solutions of scheme (8)-(9) to converge to weak solutions [67, 68]. In practice there exist two approaches to fulfill this condition. The first is to introduce a local conservative linearization of the equations. For the Euler equations with perfect gas equation of state, this can be achieved by means of a multi-dimensional extension of Roe's linearization [69, 70]. This is the approach used in *EuLFS*, which also uses the conservative linearization to evaluate all quantities necessary for the splitting.

In alternative, one can compute the element residual by approximating directly the contour integral on ∂T by some quadrature formula, and with some assumption on the (continuous) polynomial interpolation used to perform the flux evaluation in quadrature points [71, 72]. This is the formulation implemented in *NEO*, which makes use of a set of physical variables (pressure, density, velocity) for both the interpolation, and the averaging of the flux Jacobians where necessary.

Accuracy The consistency/truncation error condition originally introduced in [73] (see also [66, 65]) provides a necessary condition for second order of accuracy. Schemes verifying this condition *at steady state* can be generally cast in a FEM like form:

$$\Phi_i^T = \int_{\Omega} \omega_i \nabla \cdot \mathcal{F}(\mathbf{U}^h) dV \quad \sum_{j \in T} \omega_j|_T = \mathbf{I} \quad (11)$$

where ω_i is a bounded test function, and the second relation equivalent to the consistency condition (9).

Monotonicity The theory of positive coefficient schemes has been used systematically to study these schemes in the scalar case, proving a discrete maximum principle equivalent to the preservation in time of the initial bounds on the discrete solution (cf. [66] and references therein). Formal matrix generalizations of this condition have been considered in several works [74, 75]. Linear monotone schemes are only first order accurate, and several different non-linear approaches exist to combine second (or higher) order of accuracy with a monotonicity preserving property.

Upwinding and multidimensional upwinding A bias of the residual distribution in the direction of propagation of the information is often present. This notion is clear in one space dimension [69], and a geometrical generalization to the multi-dimensional case can be provided for scalar problems on linear finite elements [76, 77, 78], see also [66]. For multidimensional systems, this notion becomes less clear, unless one focuses on steady two-dimensional supersonic flows for which exact decompositions in scalar waves exist [56, 79]. In practice, this notion is embedded either by means of such decompositions, as it is the case in `EuLFS`, or in a formal matrix generalization of the scalar discretizations [74], as done in `NEO`.

Two families of distribution methods are available in `EuLFS` and `NEO`. Both codes implement the most classical multidimensional upwind methods, known as the high order Low Diffusion A (LDA) scheme, and the monotone Narrow (N) scheme. In the scalar case, the first corresponds to a bounded area (volume in 3D) weighted splitting [80], while the second is the optimal low diffusion upwind first order positive coefficient scheme on simplicial meshes [76, 77]. For flows with discontinuities, the non-linear blending of the above two schemes, called LDAN [73, 71], is available in both codes.

The codes also provide implementations of non-upwind methods, in the form of a centered distribution with an upwind biasing term bearing close similarities to the classical streamline upwind stabilization [81]. More particularly, `EuLFS` provides an implementation of the cell-vertex Lax-Wendroff method [82], while `NEO` features implementations of both the streamline-upwind (SU) method, as well as of a nonlinear variant of the latter in the form of a blended central (Bc) distribution between the SU method and a non-linear limited Lax-Friedrich's distribution. We refer the reader to [59, 60, 65] and references therein for more details. For more information, the interested reader may also refer to the repository documentation at <https://github.com/UnDiFi/UnDiFi-2D/wiki>.

Extension to time dependent flows. For time dependent solutions, the prototype (8) is in general inconsistent with the accuracy conditions provided e.g. in [66]. For example, the application of definition (11) to the time dependent case leads in general to the introduction of a mass matrix associated to the term $\int_{\Omega} \omega_i \partial_t \mathbf{U} dV$. This matrix has a non-diagonal structure, and may depend on the solution, which leads to a discrete prototype substantially more expensive than (8), which is closer to what is usually obtained in finite volume methods.

To cope with the cost of inverting the resulting (non-linear) system of algebraic equations several approaches have been proposed:

- fully implicit methods, often based on dual time stepping or space-time formulations [83, 84, 85, 72, 86, 57];
- predictor-corrector and defect-correction strategies which allow to keep a resolution cost very close to that of (8), while accounting for the presence of the mass matrix in the correction iterations [59, 87, 88, 65]
- Lax-Wendroff formulations [82] relying on the classical truncated Taylor series development in time to achieve high order accuracy, and on a central approximation in space coupled with mass lumping which is compatible with (8) and second order of accuracy,

The code `EulFS` features both implicit time integration with mass matrix, and an implementation of the explicit Lax-Wendroff scheme in a modified Arbitrary Lagrangian-Eulerian (ALE) formulation. Conversely, in `NEO` a second order predictor-corrector formulation of linear and non-linear schemes is implemented, both in a fixed and in an ALE form. Please refer to [58], for additional details on the schemes.

3. Unstructured Shock-Fitting: Algorithmic Features

Figure 2 shows the algorithmic workflow of the *UnDiFi-2D* code, including the sequence of subroutines and external programs being called during a typical run. The CFD codes (either `NEO` or `EulFS`) and the `Triangle` mesh-generator are invoked as black boxes, communication being handled through disk I/O, see the 1 and 5 circles in Fig. 2. Figure 2 also includes those optional parts that ensure the time-accurate integration (circled points 2, 3, 4).

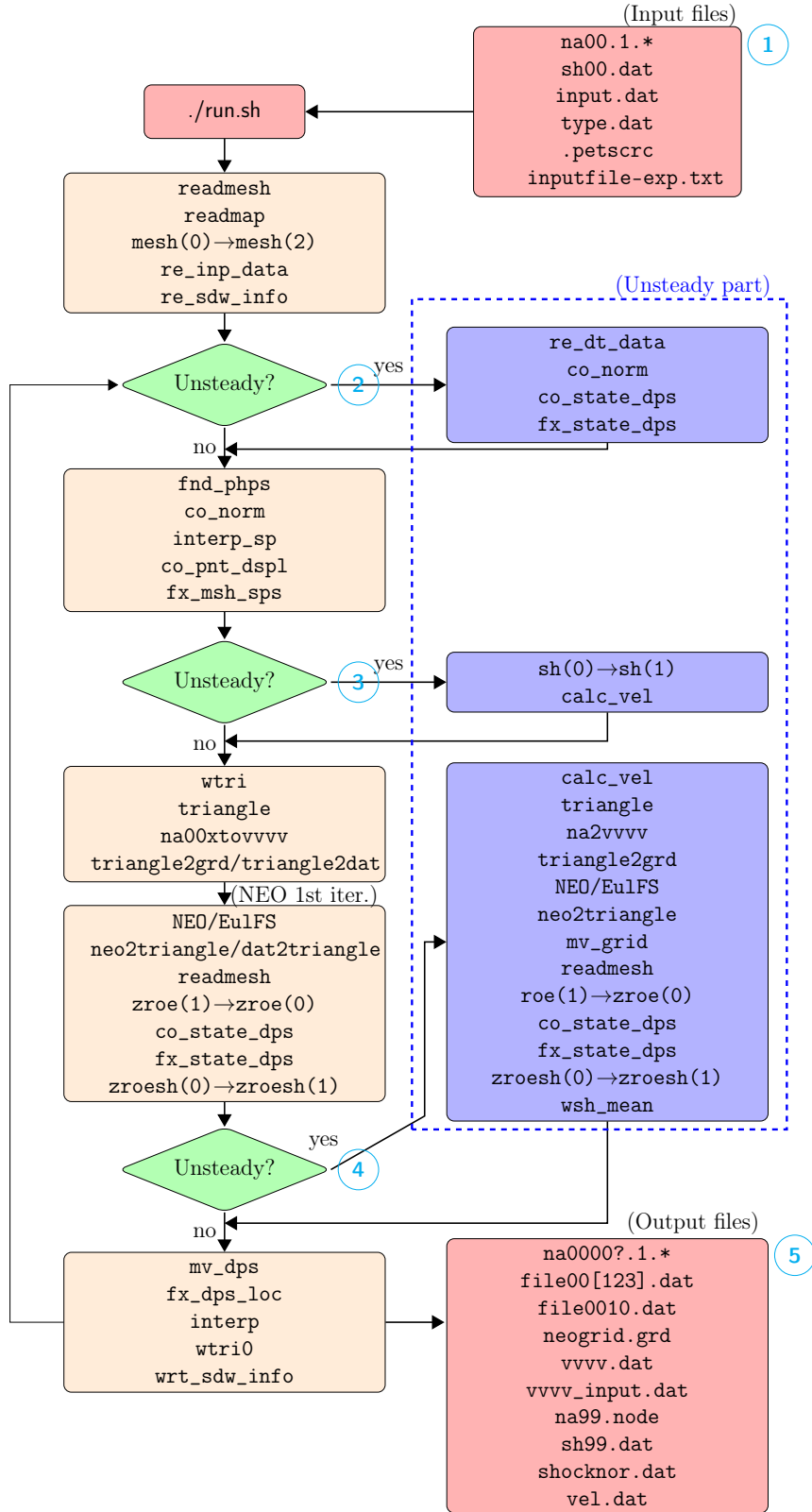


Figure 2: Typical *UnDiFi-2D* workflow.

Even though a thorough description of the various procedures listed in Fig. 2 is available in the *UnDiFi-2D* documentation, it is appropriate to give a brief description of the data storage and key algorithmic ingredients of the shock-fitting algorithm.

Regardless of whether steady or time-accurate simulations are performed, the approach is inherently time-dependent, because both the solution and the grid change with time, due to the displacement of the fitted discontinuities. When a steady solution exists, the shock speed will asymptotically vanish and the tessellation of the flow domain will not any longer change.

As far as data storage is concerned, the dependent variables and grid velocity vector are available within all grid-points of a two-dimensional triangulation that covers the entire computational domain; this is what we call the *background* mesh. In addition to the background mesh, the fitted discontinuities (either shocks or slip-lines) are discretized using a collection of grid-points (the shock-points) which are mutually joined to form a connected series of line segments (the shock-edges); shock-points and shock-edges make up what we call the *shock*-mesh. In contrast to the grid-points of the background mesh, where a single set of dependent variables is stored, the shock-points are duplicated items that share the same geometrical location, but store two different sets of dependent variables, corresponding to the two sides of the discontinuity. This is schematically shown in Fig. 3d. Shock-edges, which connect the shock-points on both sides of the discontinuity (see Fig. 3d where the width of the discontinuity has been increased to improve readability) also overlap, so that each fitted discontinuity behaves like a double-sided internal boundary of zero thickness. As shown in Fig. 3a), the spatial location of the fitted discontinuities is independent of the location of the grid-points that make up the background grid.

The sequence of operations that leads from the available mesh and solution at time t to an updated mesh and solution at time $t + \Delta t$ can be split into the seven steps that will be described in Sects. 3.1 to 3.7.

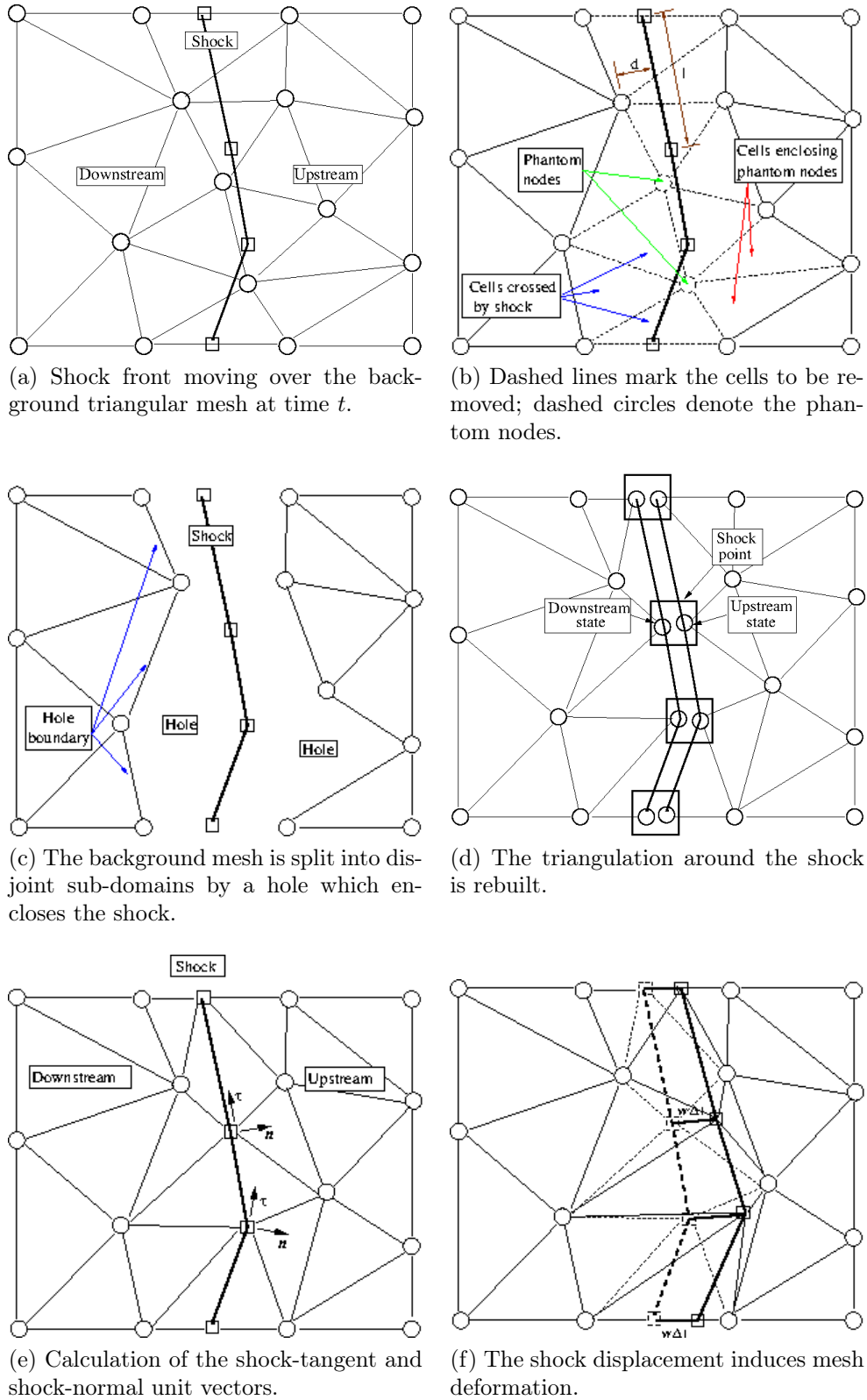


Figure 3: Unstructured shock-fitting: summary of the key algorithmic steps.

3.1. Cell Removal Around the Shock Front

In this first step, the fitted discontinuities are laid on top of the background mesh, as shown in Fig. 3a. All those cells that are crossed by the fitted discontinuities and those mesh points that are located too close to it are temporarily removed from the background mesh, as shown in Fig. 3b. This operation is performed in the subroutine `fnd_phps`, whose interface is shown in Listing 1. Here, the arguments referring to the background mesh are passed with index zero. We call phantom those grid-points of the background mesh (shown using dashed circles in Fig. 3b) that have been temporarily removed. All cells having at least one phantom node among their vertices are also removed from the background triangulation; these are the cells shown using dashed edges in Fig. 3b. Further details concerning the criteria used to identify and remove the phantom nodes can be found in [20].

Listing 1: Cell removal around the shock front.

```

call fnd_phps(
.   nedge(0),           ! number of faces
.   istak(lbndfac(0)), ! boundary faces index pointer
.   nbfac(0),          ! number of boundary faces
.   istak(lcelnod(0)), ! cell to node index pointer
.   nvt,               ! number of vertices of a cell
.   nelem(0),          ! number of elements of the background grid
.   dstak(lcorg(0)),   ! point coordinates
.   xysh,              ! shock point coordinates
.   istak(lnodcod(0)), ! node code flag
.   npoin(0),          ! number of points of the background grid
.   istak(lnodptr(0)), ! node index pointer
.   nbpoin(0),         ! number of boundary points
.   nshocks,          ! number of shocks
.   nshockpoints,     ! number of shock points
.   nshocksegs,       ! number of shock segments
.   nphanpoints,      ! number of phantom points
.   istak(lpmap(0))) ! pointer to integer array

```

It is important to underline that all the quantities that refer to the background or the computational mesh (such as the coordinates of the grid-points) are stored inside a large double precision array (`dstak`) where integers (`istak`) can also be stored by using an `EQUIVALENCE` statement. Therefore, when calling the various subroutines, these quantities are passed providing the corresponding pointers inside the `dstak` array; for example: `dstak(lcorg(0))` represents the pointer to the coordinates of the background mesh. On the contrary, the quantities that refer to the discontinuities (for example the coordinates of the shock-points of the shock-mesh) are stored in specific arrays; for example: `xysh` contains the coordinates of the shock-points.

3.2. Local Re-Meshing Around the Shock Front

Following the cell removal step, the background triangulation has been split into two or more disjoint sub-domains, as shown in Fig. 3c. The hole dug by the fitted front is then re-meshed using a Constrained Delaunay Tessellation (CDT): the edges that make up the

fitted discontinuity and the boundary of the hole are both constrained to be part of the final tessellation; this is illustrated in Fig. 3d. This operation is performed in the subroutines `fx_msh_sps`, `wtri` and by calling `Triangle` whose interfaces are shown in Listings 2, 3 and 4. Observe that re-meshing should be localized around the discontinuities. The code currently included in the repository is not fully compliant with the algorithm described above, because the CDT is applied to the entire computational domain and not only within the hole carved around the shock. Future releases will fix this open issue. Upon completion of this stage, the computational domain is discretized using what we call the *computational* mesh, which differs from the background mesh only in the neighbourhood of the fitted discontinuities. Further details concerning the software used to construct the CDT have been given in Sect. 1.6.

Listing 2: Local fix of special points.

```
call fx_msh_sps(
.   istak(lbndfac(0)), ! boundary faces index pointer
.   istak(lnodcod(0)), ! node code flag
.   nbfac(0),         ! number of boundary faces
.   nbfac_sh,        ! number of shock faces
.   nvt,             ! number of vertices of a cell
.   nelem(0),        ! number of elements of the background grid
.   dstak(lcorg(0)), ! point coordinates
.   xysh,           ! shock point coordinates
.   dstak(lcorg(0)+npoin(0)*ndim), ! upstream
.   dstak(lcorg(0)+npoin(0)*ndim+nshmax*npshmax*ndim), ! downstream
.   npoin(0),        ! number of points of the background grid
.   nshocks,         ! number of shocks
.   nshockpoints,    ! number of shock points
.   nshocksegs,      ! number of shock segments
.   nspepoints,      ! number of special points
.   typespepoints,   ! type of special points
.   shinspps,        ! special points in shock
.   ispcclr)         ! special points color
```

Listing 3: Local re-meshing around the shock front.

```
call wtri(
.   istak(lbndfac(0)), ! boundary faces index pointer
.   nbfac(0),         ! number of boundary faces
.   nbfac_sh,        ! number of shock faces
.   istak(lcelnod(0)), ! cell to node index pointer
.   nvt,             ! number of vertices of a cell
.   dstak(lcorg(0)), ! point coordinates
.   xysh,           ! shock point coordinates
.   dstak(lcorg(0)+npoin(0)*ndim), ! upstream coor.
.   dstak(lcorg(0)+npoin(0)*ndim+nshmax*npshmax*ndim), ! downstream coor.
.   dstak(lzroe(0)), ! roe variables
.   dstak(lzroe(0)+npoin(0)*ndof), ! upstream state
.   dstak(lzroe(0)+npoin(0)*ndof+nshmax*npshmax*ndof), ! downstream state
.   istak(lnodcod(0)), ! upstream node code flag
.   istak(lnodcod(0)+npoin(0)), ! downstream node code flag
```

```

.   npoin(0),           ! number of points of the background grid
.   fname(1:7),        ! mesh input file name to Triangle
.   nshocks,           ! number of shocks
.   nshockpoints,      ! number of shock points
.   nshocksegs,        ! number of shock segments
.   nphanpoints)       ! number of phantom points

```

Listing 4: Mesh generation with Triangle.

```

write(*,1001,advance='no')'triangle --> '
execmd = bindir(1:10)//'triangle_'//hostype(1:6)//
.         '-nep '//fname(1:7)//' > log/triangle.log'
ifail = system(execmd)
call flush(6)
if (ifail /= 0) then
  write(6,*)'Triangle has returned an error code ifail = ', ifail
  call exit(ifail)
endif
write(*,1002)' ok'

```

3.3. Calculation of the Unit Vectors Normal to the Shock Front

In order to apply the jump relations, normal (\mathbf{n}) and tangent ($\boldsymbol{\tau}$) unit vectors are needed within each pair of grid-points located along the discontinuities, see Fig. 3e. These unit vectors are computed using finite-difference (FD) formulae which involve the coordinates of the shock-point itself and those of its neighboring shock-points. This operation is performed in the subroutine `co_norm` whose interface is shown in Listing 5. Depending on the local, shock-downstream flow regime, it may be necessary to use upwind-biased formulae to avoid the appearance of geometrical instabilities along the fitted discontinuity. Full details describing how to select the stencil can be found in [20]. Be aware that the FD formulae reported in [20] contain a typo and should be replaced by those published in [32].

Listing 5: Calculation of the unit vectors normal to the shock front.

```

call co_norm(
.   xysh,                ! shock point coordinates
.   dstak(lzroe(0)+npoin(0)*ndof),           ! upstream
.   dstak(lzroe(0)+npoin(0)*ndof+nshmax*npshmax*ndof), ! downstream
.   norsh,               ! normal vectors
.   nshocks,             ! number of shocks
.   nshockpoints,        ! number of shock points
.   typeshocks,          ! type of shock
.   nspecpoints,         ! number of special points
.   typespecpoints,      ! type of special points
.   shinspps,            ! special points in shock
.   ispcclr,             ! special points color
.   istak(lia(0)),        ! pointer for integer arrays in setbndrynodeptr
.   istak(lja(0)),        ! pointer for integer arrays in setbndrynodeptr
.   istak(liclr(0)),      ! pointer for integer arrays in setbndrynodeptr
.   nclr(0),              ! colours of boundary patches
.   dstak(lcorg(0)))     ! point coordinates

```

3.4. Solution Update Using the Shock-Capturing Code

Using the computational mesh as input, a single time step calculation is performed using one of the available shock-capturing solvers which returns updated nodal values at time $t + \Delta t$, as shown in Listing 6 for the `EulFS` solver. Since the discontinuities are seen by the shock-capturing code as internal boundaries (of zero thickness) moving with the velocity of the discontinuity, there is no need to modify the spatial discretization scheme already implemented in the PDEs solver to account for the presence of the fitted discontinuities. In practice, the shock-capturing solver is used as a black-box: it receives in input the computational grid, the nodal values of the solution and grid velocity at time t and returns the updated solution at time $t + \Delta t$. The solution returned by the shock-capturing solver at time $t + \Delta t$ is however missing some boundary conditions. These missing pieces of information will be determined as described in Sect. 3.5.

Listing 6: Call to the gasdynamics solver.

```
execmd = bindir(1:10)//"eulfs_"//hostype(1:6)//"-itmax 1 > log/eulfs.log"
ifail = system(execmd)
call flush(6)
if (ifail /= 0) then
  write(6,*) "Eulfs has returned an error code ifail = ", ifail
  call exit(1)
endif
```

3.5. Enforcement of the Jump Relations

As explained in Sect. 2.2, the missing pieces of information that are needed to correctly update the solution within all pairs of grid-points located along the discontinuities are obtained by enforcing the R-H jump relations; this also provides the local velocity of the discontinuity along its normal. The R-H jump relations are a set of non-linear algebraic equations that can be solved within each pair of grid-points located along the discontinuities by means of Newton-Raphson's algorithm. In order to match the number of unknowns with the available equations, one or more additional pieces of information are required within both or either of the two sides of the fitted discontinuity, depending on whether this is a shock or a contact discontinuity. These additional pieces of information are obtained from the characteristic formulation of the Euler equations and correspond to those characteristic quantities that are convected towards the discontinuity from the sub-domain that is attached to that side of the discontinuity. Using an upwind-biased discretization within the shock-capturing solver, one can reasonably assume that the spatial and temporal evolution of these characteristic quantities has been correctly computed.

The enforcement of the R-H jump relations is performed in the subroutine `co_state_dps` whose interface is shown in Listing 7. Full algorithmic details concerning the practical implementation of the jump relations for shocks and contact discontinuities are reported elsewhere [21, 20, 22] and will not be repeated here. Furthermore, an ad-hoc treatment is required within those special points where different discontinuities interact (triple or quadruple points) or whenever a discontinuity interacts with a solid or free boundary. The algorithmic

details are described in [21, 22] and their implementation can be found in the subroutines `co_utp`, `co_uqp`, respectively.

Listing 7: Enforcement of the jump relations.

```

call co_state_dps (
.   xysh,           ! shock point coordinates
.   dstak(lzroe(0)+npoin(0)*ndof),           ! upstream
.   dstak(lzroe(0)+npoin(0)*ndof+nshmax*npshmax*ndof), ! downstream
.   zroeshuold,     ! upstream zroe states at previous iteration
.   zroeshdold,     ! downstream zroe states at previous iteration
.   norsh,         ! normal vectors
.   wsh,           ! shock velocity
.   nshocks,       ! number of shocks
.   nshockpoints,  ! number of shock points
.   nshocksegs,    ! number of shock segments
.   typeshocks,    ! type of shock
.   iter)          ! current iteration

```

3.6. Shock Displacement

The enforcement of the jump relations provides the speed, w , at which each pair of grid-points located on the discontinuity move along its local normal unit vector, \mathbf{n} . The position of the discontinuity at time $t + \Delta t$ is computed in a Lagrangian manner by displacing all its grid-points, as shown in Fig. 3f where the dashed and solid lines represent the discontinuity at time t , resp. $t + \Delta t$. When simulating steady flows, this can be accomplished using the following first-order-accurate (in time) integration formula:

$$\mathbf{P}_i^{t+\Delta t} = \mathbf{P}_i^t + w_i^t \mathbf{n}_i^t \Delta t \quad (12)$$

which returns the spatial coordinates of the i -th shock-point at time $t + \Delta t$. The low temporal accuracy of Eq. (12) does not affect the spatial accuracy of the steady state solution which only depends on the spatial accuracy of the gas-dynamic solver and that of the tangent and normal unit vectors.

On the contrary, when dealing with unsteady flows, the temporal accuracy of the shock motion has to be the same as that of the spatial discretization, i.e. second order accurate in our case. This can be accomplished using a predictor-corrector type temporal integration scheme, or a Runge-Kutta multi-step scheme. In the former case, the predictor step estimates the position of the discontinuity at time level $n + 1/2$ using the explicit Euler scheme:

$$\mathbf{P}_i^{n+\frac{1}{2}} = \mathbf{P}_i^n + w_i^n \mathbf{n}_i^n \frac{\Delta t}{2} \quad (13)$$

The speed of the discontinuity $w_i^{n+\frac{1}{2}}$ and the normal unit vector $\mathbf{n}_i^{n+\frac{1}{2}}$ at time level $n + 1/2$ are then computed using the intermediate position of the discontinuity $\mathbf{P}_i^{n+\frac{1}{2}}$ and, finally, the position of each shock-point is updated at time level $n + 1$ in the corrector step:

$$\mathbf{P}_i^{n+1} = \mathbf{P}_i^n + w_i^{n+\frac{1}{2}} \mathbf{n}_i^{n+\frac{1}{2}} \Delta t \quad (14)$$

This operation is performed in the subroutine `mv_dps` whose interface is shown in Listing 8.

Listing 8: Shock displacement.

```

call mv_dps(
.   xysh,           ! shock point coordinates
.   dstak(lzroe(0)+npoin(0)*ndof+nshmax*npshmax*ndof), ! downstream
.   wsh,           ! shock velocity
.   i,             ! current iteration
.   nshocks,       ! number of shocks
.   nshockpoints, ! number of shock points
.   nshocksegs,    ! number of shock segments
.   typeshocks)    ! type of shock

```

One further observation is in order concerning the discontinuity displacement step. Figure 3f shows that even when the background mesh is fixed in space, the triangular cells that abut on the discontinuity have one of their edges that moves with the discontinuity, thus deforming the cell. This implies that the shock-capturing solver used in Step 3.4 must be capable of handling moving meshes, i.e. it must be capable of solving the governing PDEs written using an Arbitrary Eulerian Lagrangian (ALE) formulation.

3.7. Interpolation of the Phantom Nodes

Upon completion of the previous steps, all grid-points of the computational mesh have been updated at time $t + \Delta t$. The computational mesh is made up of shock-mesh and all grid-points of the background mesh, except those that have been declared *phantom*. Therefore, the nodal values within the phantom nodes have not been updated to time $t + \Delta t$. However, during the current time step, the discontinuities might have moved sufficiently far away from their previous position, that some of the phantom nodes may re-appear in the computational mesh at the next time step. It follows that also the nodal values within the phantom nodes need to be updated to time $t + \Delta t$. This is easily accomplished by transferring the available solution at time $t + \Delta t$ from the current computational mesh to the grid-points of the background one, using linear interpolation. This operation is performed in the subroutine `interp` whose interface is shown in Listing 9. Once the phantom nodes have been updated, the computational mesh used in the current time interval has completed its task and can be discarded. At this stage the numerical solution has correctly been updated at time $t + \Delta t$ within all grid-points of the background and shock meshes.

The next time interval can be computed re-starting from step 3.1 of the algorithm.

Listing 9: Interpolation of the phantom nodes.

```

call interp(
.   istak(lbndfac(1)), ! boundary faces index pointer
.   nbfac(1),         ! number of boundary faces
.   istak(lcelnod(1)), ! cell to node index pointer
.   nvt,              ! number of vertices of a cell
.   nelelem(1),       ! number of elements of the shocked grid
.   dstak(lcorg(1)),  ! point coordinates

```

```

.   dstak(lzroe(1)),      ! point roe states
.   xysh,                ! shock point coordinates
.   dstak(lcorg(1)+npoin(0)*ndim),      ! upstream
.   dstak(lcorg(1)+npoin(0)*ndim+nshmax*npshmax*ndim), ! downstream
.   nphampoints,        ! number of phantom points
.   dstak(lcorg(0)),    ! background point coordinates
.   dstak(lzroe(0)),    ! background points roe states
.   istak(lnodcod(0)), ! background grid node code flag
.   npoin(0),           ! number of points of the background grid
.   nshocks,            ! number of shocks
.   nshockpointsold,   ! old number of shock points
.   nshockpoints,      ! new number of shock points
.   istak(lia(1)),     ! pointer for integer arrays in setbndrynodeptr
.   istak(lja(1)),     ! pointer for integer arrays in setbndrynodeptr
.   istak(liclrl(0)),  ! pointer for integer arrays in setbndrynodeptr
.   nclr(0)            ! colours of boundary patches

```

4. Test cases

In order to illustrate the capabilities of *UnDiFi-2D* several test-cases can be run within each of the sub-directories (listed in Fig. 4) of the folder `tests`. In addition to this, these test-cases are intended to test different parts of the code and, therefore, the successful execution of all test-cases (which can be performed with the script `run_all_x86.sh`) represents a verification of the code. These checks are particularly important before a new version of the code is released to verify that changes and modifications of the code have not produced unwanted side-effects elsewhere in the code.

More specifically, in the shock-shock or shock-wall interaction test-cases addressed in Sect.4.2, 4.3 and 4.4, the various discontinuities bound regions of uniform flow, where an analytical solution can be computed using the jump relations. Therefore, these test-cases provide a powerful code validation tool, because the shock-fitting algorithm is expected to return the exact solution everywhere, even if a first-order accurate discretization of the governing PDEs is used.

An analytical solution is also available for the transonic test-case addressed in Sect. 4.6 which features spatially variable fields both upstream and downstream of the shock, so that it can be used to measure the order-of-convergence of the spatial discretization.

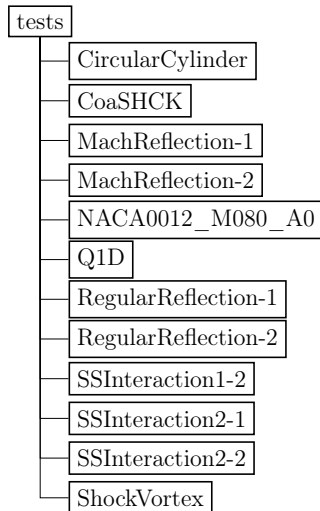


Figure 4: Test-cases available in the `tests` directory.

4.1. Hypersonic flow past a Circular cylinder (*CircularCylinder-1*)

The first test-case deals with the hypersonic flow past a circular cylinder at free-stream Mach number, $M_\infty = 20$. It was one of the first flow configurations to be set-up and tested during the *UnFiDi-2D* code development, because this apparently simple test-case allows to check the basic subroutines of the code, such as those in charge of calculating the shock normal unit vectors and applying the R-H relations, under all possible post-shock conditions. This is because by moving along the bow shock is like sweeping the entire shock polar, starting from a normal shock at the symmetry axis, which then turns into a strong oblique shock and finally becomes a weak wave.

As shown in Fig. 5, the computational domain surrounds the fore half of a circular cylinder having radius $R = 1$. The background mesh has been created using the `deLaundo` [89] mesh generator by specifying a evenly spaced distribution of boundary nodes with spacing $h = 0.08 R$. The numbers of triangles and nodes of the background mesh are reported in Tab. 1.

Table 1: Nodes and cells of the background and computational meshes

Background Mesh		Computational mesh at steady-state		
Triangles	Nodes	Triangles	Nodes	Shock nodes
610	351	624	358	29

The solution computed on the background mesh using the unstructured code in shock-capturing mode has been used to initialize the flow-field and to determine the parabolic shape of the initial position of the shock front, see Fig. 5.

In the shock-fitting simulation, the initial upstream state in the shock-points has been set equal to the free-stream conditions, while the initial shock-downstream state has been computed from the upstream state and the local shock slope, assuming zero shock speed: $w = 0$. The flow-field and shock position are then integrated in pseudo-time until steady-state is reached. Fig. 5a shows the shock displacement that occurs between the initial shock-position and the one reached at steady state. During the computation, the number of grid-points and triangles of the computational mesh varies with respect to that of the background mesh, as shown in Tab. 1. Figure 5b also displays the background mesh and the computational mesh at steady-state. The smaller frame of Fig. 5b, which shows an enlargement of the near-shock region, allows comparing the two meshes. It can be seen that these are superimposed everywhere except in the region adjacent to the shock, where the differences between the background (dashed lines) and the computational mesh (solid lines) are due to the addition of the shock-points and shock-edges. Fig. 5b, as well as Tab. 1, clearly show that the re-meshing technique does not significantly increase the number of grid-points and triangles with respect to those of the background mesh.

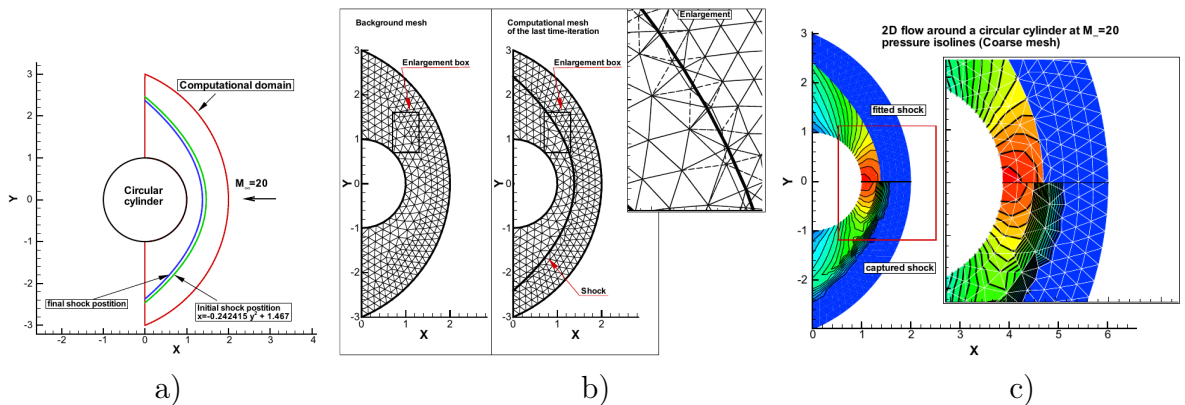


Figure 5: Computational domain: initial and final shock position. Comparison between the coarse background and the computational mesh at steady-state. Comparison between the shock-fitting and shock-capturing solutions.

Figure 5c allows to compare the solutions computed by the `EuLFS` code working in shock-capturing and shock-fitting mode; similar results can be obtained using `NEO`. Inside this folder (as well as the other folders that will be described below) there is a script (`run.sh`) that allows to run the shock-capturing or shock-fitting solutions with either `EuLFS` or `NEO`. Figure 5c also includes a detailed view of the stagnation point region showing the modifications introduced by the shock-fitting algorithm to the background mesh in order to take into account the presence of the shock front. A detailed analysis of the numerical results of this test-case can be found in [20].

4.2. Interaction between two shocks of opposite families (*SSInteraction1-2*)

A uniform supersonic ($M_\infty = 3$) stream is crossed by two incident shocks, I1 and I2, of opposite families and different strength which interact in the quadruple point, QP, giving

rise to two outgoing reflected shocks, R1 and R2, and a slip-line (or slip-stream), SS, located between the two reflected shocks. These five discontinuities bound as many uniform flow regions (numbered as shown in Fig. 6) surrounding the quadruple point.

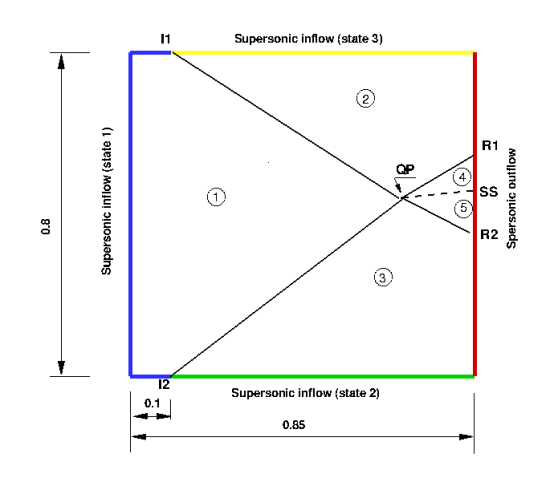


Figure 6: Interaction between two shocks of opposite families: flow configuration.

Given the free-stream Mach number and flow deflections ($\delta_2 = -15^\circ$ and $\delta_3 = 20^\circ$) through the two incident shocks, the constant states in regions 2 to 5 can be analytically computed and are listed in Tab. 2, which includes: the Mach number (M), the deflection angle (δ) measured with respect to the horizontal axis and the pressure (p/p_1) and density (ρ/ρ_1) ratios. This test-case checks the code capability of computing multiple shocks and

Table 2: Interaction between two shocks of opposite families: analytical steady state solution.

	1	2	3	4	5
M	3.0	2.255	1.994	1.461	1.432
δ	0.0	-15°	20°	4.796°	4.796°
p/p_1	1.0	2.822	3.771	8.353	8.353
ρ/ρ_1	1.0	2.034	2.420	4.263	4.211

slip-lines, but it also verifies the computation of the quadruple point.

The numerical solutions have been computed inside a rectangular domain ($0.85 L \times 0.8 L$) with boundary conditions prescribed as shown in Fig. 6. The background mesh is made of 7988 grid-points and 15644, mostly equilateral, triangles. The characteristic mesh spacing is about $0.01 L$.

The shock-fitting simulation has been initialised using the shock-capturing solution obtained on the background mesh. This solution has also been used to supply the approximate position of the four shocks, the slip-stream and the quadruple point which are needed to construct the internal boundaries corresponding to the various discontinuities. Starting from this initial flow-field, the shock-fitting calculation has been integrated in pseudo-time until

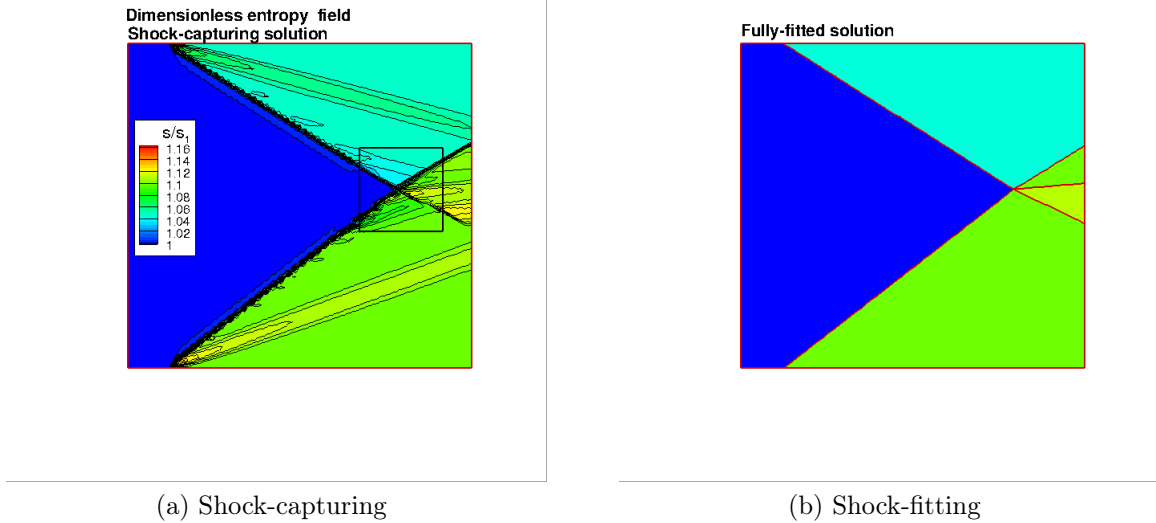
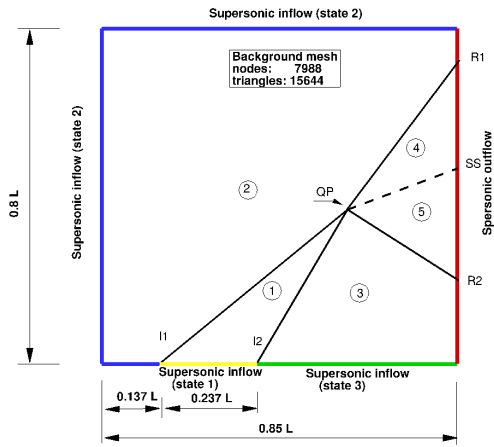


Figure 7: Interaction between two shocks of opposite families: computed entropy iso-contours.

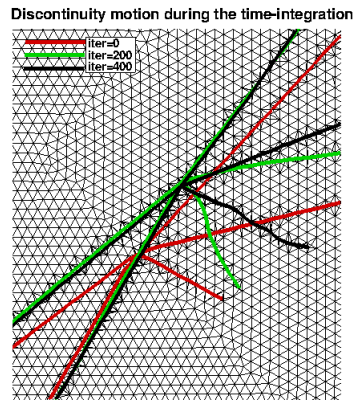
steady state is reached. The numerical solution obtained using the `eulfs` shock-capturing solver is compared in Fig. 7 with that obtained in fully-fitted mode. Since the exact solution is uniform within each of the five regions bounded by the discontinuities, the shock-fitting calculation is expected to return the exact solution of Tab. 2. This is indeed the case, as revealed by Fig. 7b. Further analyses dealing with this particular test-case can be found in [22], where the analytical and algorithmic treatment of the slip-lines and the quadruple point are also described in detail.

4.3. Interaction between two shocks of the same families (*SSInteraction2-1*, *SSInteraction2-2*)

This test-case is similar to the one addressed in Sect. 4.2, except that the two incident shocks, I1 and I2, belong to the *same* family. Their interaction in the quadruple point, QP, gives rise to a reflected shock, R1, which is stronger than the two incident ones, a slip-stream, SS, and a second reflected shock wave, R2, of nearly negligible strength. If we name the various zones and discontinuities as shown in Fig. 8a, the only topological difference between the interaction of shocks of the same or opposite families lies in the orientation of the unit vector normal to I1, which points from zone 1 to zone 2 in the present case, whereas the orientation is reversed, see Fig. 6, when the interacting shocks belong to opposite families. Because of this, the same subroutine `co_uqp` is used to compute the QP, regardless of whether the interacting shocks belong to the same or opposite families. The analytical steady state solution, which is listed in Table 3, has been analytically computed from the known free-stream state ($M_2 = 2.0$) and the known flow deflections through the incident shocks: $\delta_1 = 10^\circ$ and $\delta_3 = 20^\circ$. The present interaction has been numerically simulated both in hybrid and fully-fitted mode: the two different set-ups are available in the *SSInteraction2-1* and *SSInteraction2-2* folders. The computational domain and the background triangular grid are identical to those described in Sect. 4.2, whereas the boundary conditions have been changed as described in Fig. 8a.



(a) Computational domain and boundary conditions.



(b) Motion of the discontinuities during pseudo-time integration.

Figure 8: Interaction between two shocks of the same family.

Table 3: Interaction between two shocks of the same family: analytical steady state solution

	1	2	3	4	5
M	1.641	2.000	1.285	1.218	1.280
δ	10°	0°	20°	19.87°	19.87°
p/p_1	1.707	1.000	2.803	2.822	2.822
ρ/ρ_1	1.460	1.000	2.074	2.035	2.084

In the hybrid simulation (the middle row of Fig. 9) only the incident and one of the reflected shocks, resp. I1 and R1, have been fitted, whereas all other discontinuities have been captured. This makes the hybrid shock-fitting approach algorithmically simpler than the fully-fitted one, not only because I1 and R1 are treated as a single fitted-shock, but also because the QP is captured and, therefore, does not require any modeling. In the fully-fitted simulation (the bottom row of Fig. 9), on the contrary, all discontinuities have been fitted, including their interaction in the QP, which has been modelled as described in Sect. 4.2. Note, also, that in the fully-fitted simulation the two reflected shocks and the slip-stream do not reach the domain boundaries, but have been fitted only up to a preset location, downstream of which it is left to the shock-capturing code to capture each of these three discontinuities. This choice has been deliberately made to demonstrate the code capability to treat the same discontinuity in a hybrid manner, i.e. to “fit” only part of it, whereas the remainder is captured. This requires an *ad-hoc* modeling of the pair of shock-points where the fitted-front terminates within the flow-field. Therefore, the present test-case not only completes the testing of the QP modeling, but also of those subroutines dealing with the aforementioned end-points.

As mentioned earlier, we rely upon the solution computed in shock-capturing mode (the

top row of Fig. 9) to provide the initial location of all the fitted entities: shocks, slip-line and interaction points, along with a reasonable initial flow-field. By doing so, the initial position of the fitted discontinuities turns out to be very close to the position reached at steady-state. For the present test-case, however, in order to challenge the proposed shock-fitting technique, the initial position of the various discontinuities and their interaction point have been deliberately prescribed in locations that are far from those predicted by the shock-capturing simulation. Fig. 8b shows how the position of the various discontinuities and the computational mesh of the fully-fitted solution evolve during the calculation. This test-case clearly demonstrates that the shock-fitting technique is able to move correctly all discontinuities, along with the QP, towards their steady state location, even though this is quite far from the initial guess.

Figure 9 shows the entropy field obtained using the shock-capturing (top row), hybrid (middle row) and fully-fitted (bottom row) approaches. All these solutions have been obtained using the `eulfs` solver. The three frames in the left column of Fig. 9 show the entire computational domain, whereas those in the right column show a detailed view of the entropy distribution superimposed on the computational mesh in the neighbourhood of the QP. Further details and results concerning this test-case can be found in [22].

4.4. Shock-wall interaction: regular reflection. (*RegularReflection-1, RegularReflection-2*)

When the two incident shocks of opposite families of Sect. 4.2 have equal strength, the flow-field is symmetric w.r.t. the SS and the flow-field describes the regular reflection of a weak oblique shock from a flat, solid boundary, as sketched in Figure 10. Regular reflection is possible as long as the flow deflection through the incident shock, I1, is lower than the maximum deflection for the Mach number in region 2. When I1 impinges (in point IP) on the straight wall, it gives rise to a reflected oblique shock, R1, of the opposite family. The supersonic stream undergoes the same deflection, although opposite in sign, through both shocks. For the given flow parameters ($M_1 = 2$ and $\delta_2 = 10^\circ$), Table 4 shows the analytically computed Mach number, normalised pressure, density and flow direction (measured w.r.t. the wall) in all three regions shown in Fig. 10.

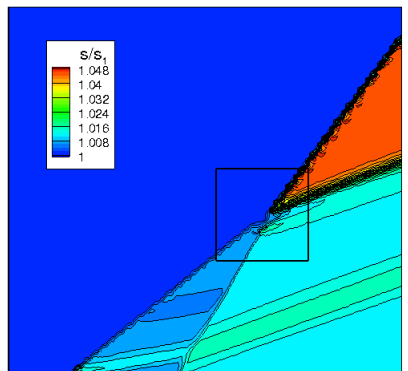
Table 4: Regular reflection: steady state analytical solution.

	1	2	3
M	2.0	1.641	1.287
δ	0°	-10°	0°
p/p_1	1.0	1.706	2.797
ρ/ρ_1	1.0	1.458	2.069

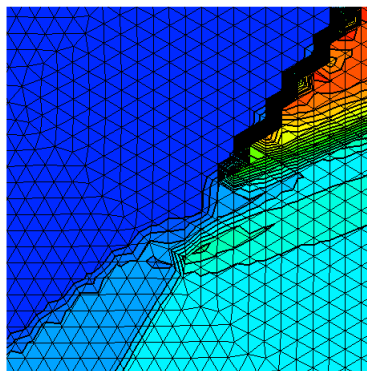
The present test-case has also been computed using both the hybrid and fully-fitted mode; the corresponding computational setups can be found in the `RegularReflection-1` and `RegularReflection-2` folders, respectively. The purpose of the present test-case, when run in fully-fitted mode, is to check the numerical modeling of the regular wall reflection.

The computational domain is a $2L \times L$ rectangle, which is shown in Fig. 10 along with the boundary conditions. The background mesh, which has been generated using the

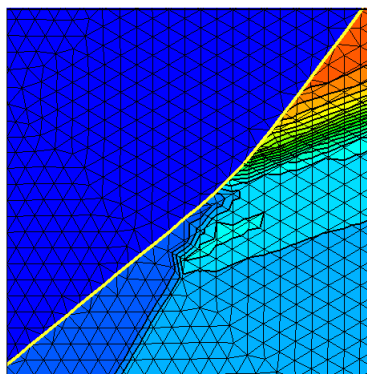
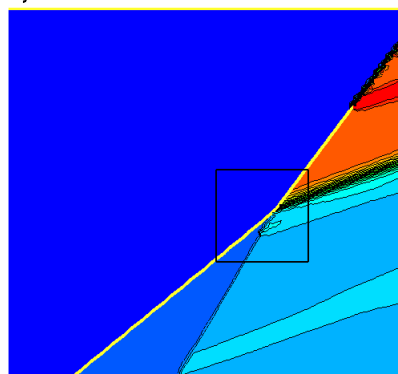
Entropy field
shock-captured solution



Enlarged view around QP



Hybrid solution



Fully-fitted solution

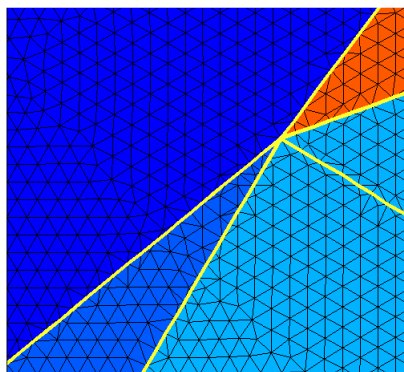
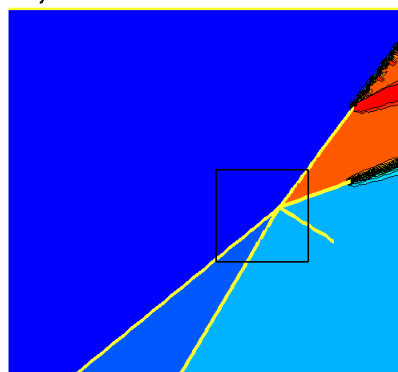


Figure 9: Interaction between two shocks of the same family: entropy iso-contour lines.

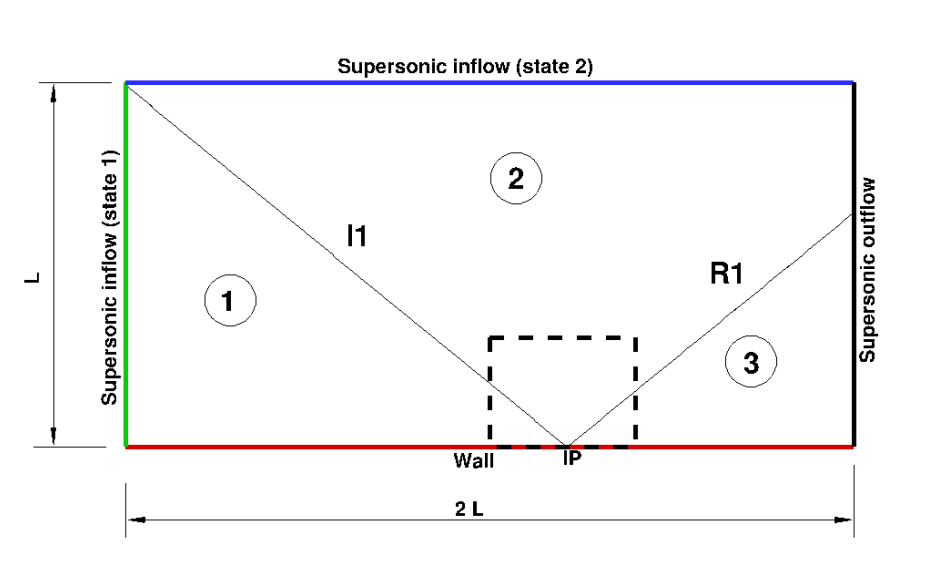


Figure 10: Regular reflection: computational domain and boundary conditions.

Triangle [53, 54] mesh generator by constraining the triangle areas not to exceed $0.0002 L^2$, is made of 10281 grid-points and 16016 triangles. Unlike the meshes used for the previous test-cases, which were almost entirely made of equilateral triangles, the one used here features a rather irregular distribution of grid-points and scalene triangles; compare the right column of frames of Figs. 9 and 11.

Figure 11 shows the computed entropy field for all three sets of calculations: shock-capturing computed on the background mesh (top row), hybrid (middle row) and fully-fitted (bottom row). All three different shock-modeling options make use of the **eulfs** solver. Frames in the left column of Fig. 11 show the entire computational domain (the dashed square points to the location of the IP), whereas those in the right column show a zoom of the region surrounding the IP. Further details and a more extensive commentary of the results can be found in [22].

4.5. Steady Mach reflection (*MachReflection-1*, *MachReflection-2*)

Whenever the flow deflection imposed by a solid surface on an impinging weak oblique shock is larger than the maximum allowable deflection for the downstream Mach number, a so-called Mach reflection takes place instead of the regular reflection already examined in Sect. 4.4. This is schematically shown in Fig. 12: a uniform, supersonic ($M_\infty = 2$) stream of air undergoes a $\Theta = 14^\circ$ deflection through the incident shock, I1. Regular reflection of I1 is however impossible for the chosen pair of M_∞, Θ parameters, so that a steady triple-point (TP) arises which joins I1, the reflected shock (R1), the Mach stem (MS) and the slip-stream (SS).

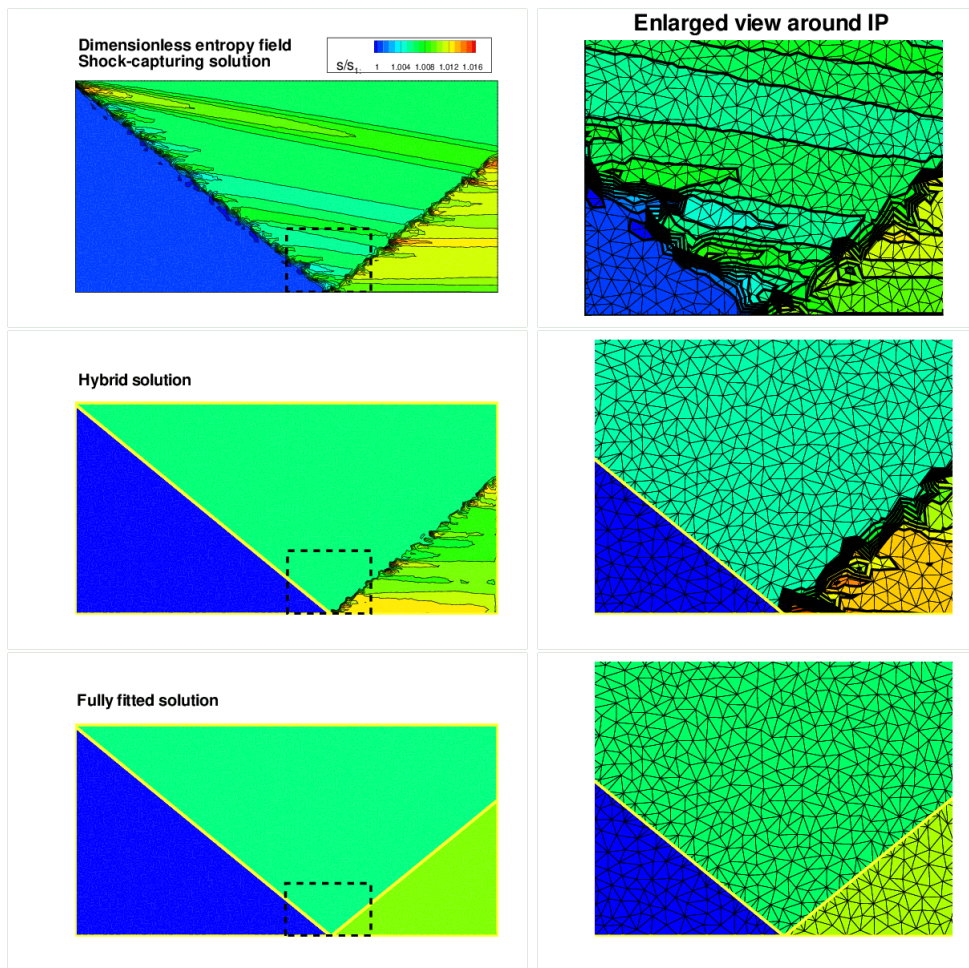


Figure 11: Regular reflection: entropy field computed using the three different shock-modeling options and the *eulfs* solver.

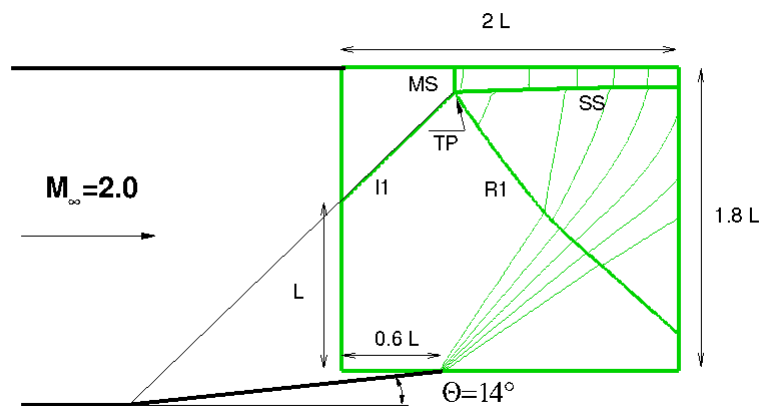


Figure 12: Steady Mach reflection: flow configuration.

This test-case has been split into two different simulations: the `MachReflection-1` directory contains the hybrid simulation, whereas a fully-fitted simulation can be run in the `MachReflection-2` directory. In the former case the incident shock and the slip-stream are captured, whereas the reflected shock and the Mach stem are fitted as a single shock. In the latter case all discontinuities, as well as the triple-point, are fitted. From the viewpoint of code-checking the two simulations are very different. The hybrid simulation only demonstrates the capability of the algorithm to run in hybrid mode, but tests the same functionalities already checked in the circular cylinder simulation, with the only addition of the interaction between a normal shock (the MS) and a flat wall. On the contrary, the fully fitted simulation tests the capability of the `UnDiFi-2D` code to fit the triple point.

The computational domain used for both simulations has been marked in green in Fig. 12. Inside this area a background grid of almost equilateral triangles has been generated using the `delaundo` [89] mesh generator by specifying a uniform distribution of grid-points along the domain boundaries with spacing $h = 0.0167 L$, see Fig. 12. Table 5 reports the number of triangles and grid-points of the background mesh along with those of the computational meshes at steady-state for both the hybrid and the fully-fitted simulations; the corresponding number of shock-points is also shown.

Table 5: Grid-points and triangles of the background and computational meshes.

Background		Computational meshes at steady-state					
Mesh		Hybrid simulation			Fully fitted simulation		
Cells	Nodes	Cells	Nodes	Shock-points	Cells	Nodes	Shock-points
29214	14833	29292	17633	142	29365	19633	294

Shock-capturing, hybrid and fully-fitted calculations obtained using the two different Residual Distribution codes, `NEO` and `eulfs`, are compared in Fig. 13.

Even though the two solvers use very similar numerical recipes, as explained in Sect. 2.3, it can be seen that the two shock-capturing solutions, Figs. 13a and 13d, exhibit non-negligible differences, in particular downstream of the Mach stem. These differences are significantly reduced in the hybrid simulations, Figs. 13b and 13e, and have almost disappeared in the fully fitted ones, Figs. 13c and 13f.

Further analyses about these numerical solutions can be found in [20, 22], whereas the algorithmic details concerning the treatment of the triple point are reported in [21].

4.6. Planar source flow (*Q1D*)

This test-case consists in a supersonic planar source flow that originates from the centre of the inner red circle of Fig. 14a and, passing through a normal shock (the green circle in Fig. 14a), turns into a subsonic source flow. Assuming that the analytical velocity field has a purely radial velocity component, it may be easily verified that the governing PDEs,

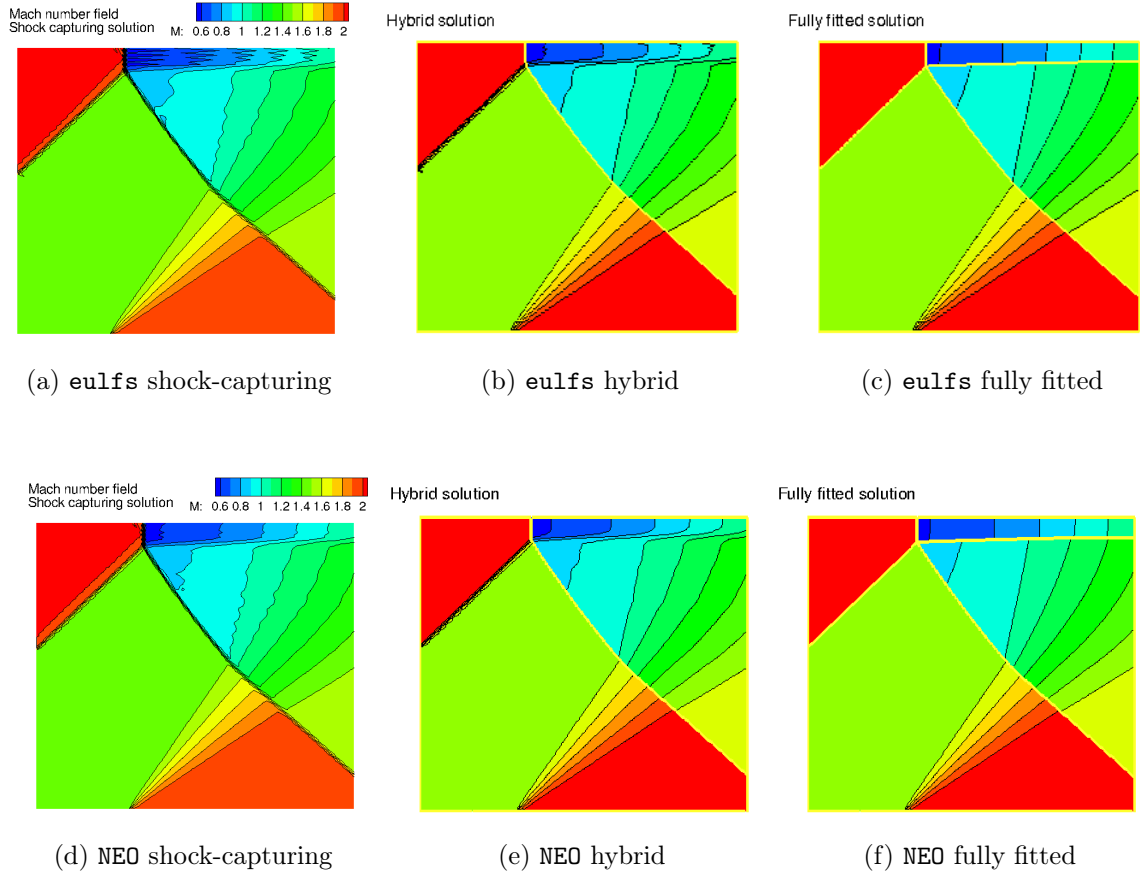


Figure 13: Mach reflection: Mach number iso-contour lines computed using three different shock-modeling options and the two different shock-capturing codes.

written in a polar coordinate system, become identical to those governing a compressible quasi-one-dimensional variable-area flow:

$$\frac{1}{M} \left[\frac{2}{\gamma + 1} \left(1 + \frac{\gamma - 1}{2} M^2 \right) \right]^{\frac{\gamma + 1}{2(\gamma - 1)}} = \frac{A}{A^*} \quad (15)$$

provided that the nozzle area $A = A(r)$ varies linearly with the radial distance, r , from the source. Equation (15) allows to calculate the solution in the smooth flow regions both upstream and downstream of the normal shock, whereas pre- and post-shock conditions are linked through the R-H jump relations.

The computational domain consists in the annulus sketched in Fig. 14a: the ratio between the radii of the outer and inner circles ($L = r_{in}$) has been set equal to $r_{out}/r_{in} = 2$ and the exact solution features a supersonic inlet flow ($M_{in} = 2$) along the inner circle and a ratio between the outlet static and inlet total pressures $p_{out}/p_{in}^0 = 0.47$ such that the shock forms at $r_{sh}/r_{in} = 1.5$. The Delaunay mesh, which is partially shown in Fig. 14b, is made of 6916 grid-points and 13456 triangles and has been generated using **Triangle** [53, 54] in such a way

that no systematic alignment occurs between the edges of the triangulation and the shock-edges, thus making the discrete problem truly two-dimensional. Due to radial symmetry,

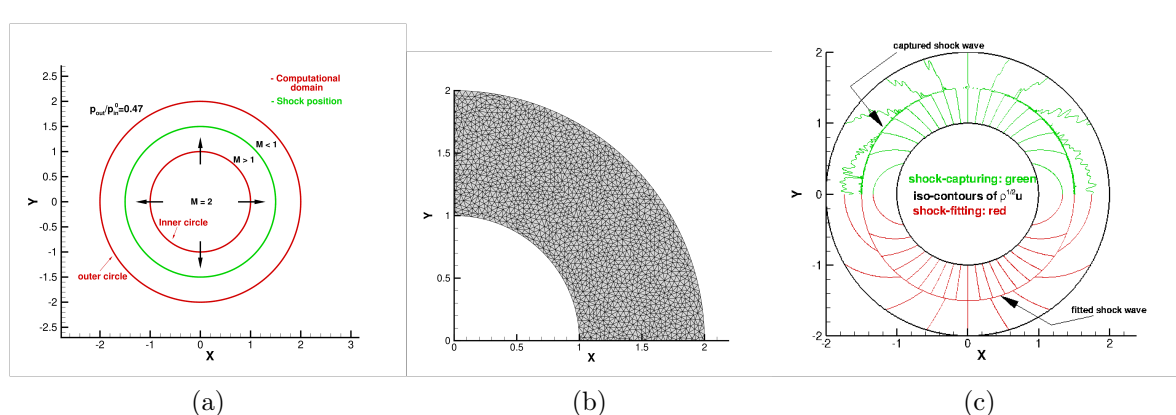


Figure 14: Planar, transonic source flow: computational domain, mesh and solutions.

the shock is a closed curve, as shown in Figs. 14a and 14c. From an algorithmic viewpoint, this is obtained by overlapping the two end-points of the shock-mesh. Therefore, beside being a powerful tool for code verification, thanks to the availability of an exact solution, the present test-case checks that the two end-points of the shock-mesh are correctly joined to form a single closed shock.

A qualitative comparison between the shock-capturing and shock-fitting solutions computed using the `eulfs` solver is shown in Fig. 14c, which shows the iso-contours of the third component, $\sqrt{\rho}u$, of the parameter vector. It is evident that the two solutions are identical within the supersonic, shock-upstream region, whereas noticeable differences are visible within the entire subsonic, shock-downstream region. This is because the captured shock wave has given rise to remarkable oscillations downstream of the shock wave. Full details concerning this test-case can be found in [23].

4.7. Shock formation due to the coalescence of compression waves (*CoaSHCK*)

Figure 15a shows a supersonic stream ($M_\infty = 2.3$) being deflected by a convex wall. The wall is straight up to point A and beyond point B, whereas the shape of the wall smoothly changes between A and B according to the following cubic polynomial law:

$$y = 0.039923 x^3 + 0.25144 x^2 + 0.46928 x - 0.52683 \quad -1.4 \leq x \leq 0.044 \quad (16)$$

As sketched in Fig. 15a, the Mach waves that originate in A and B bound a simple-wave compression region made of straight characteristic lines that merge into a shock-wave at some distance from the wall. Figure 15b shows the computational domain and the boundary conditions. The background mesh, which is made of 25531 triangles and 12965 grid-points, has been generated using the `delaundo` mesh generator by specifying a uniform distribution of the boundary nodes with spacing $h = 0.025$.

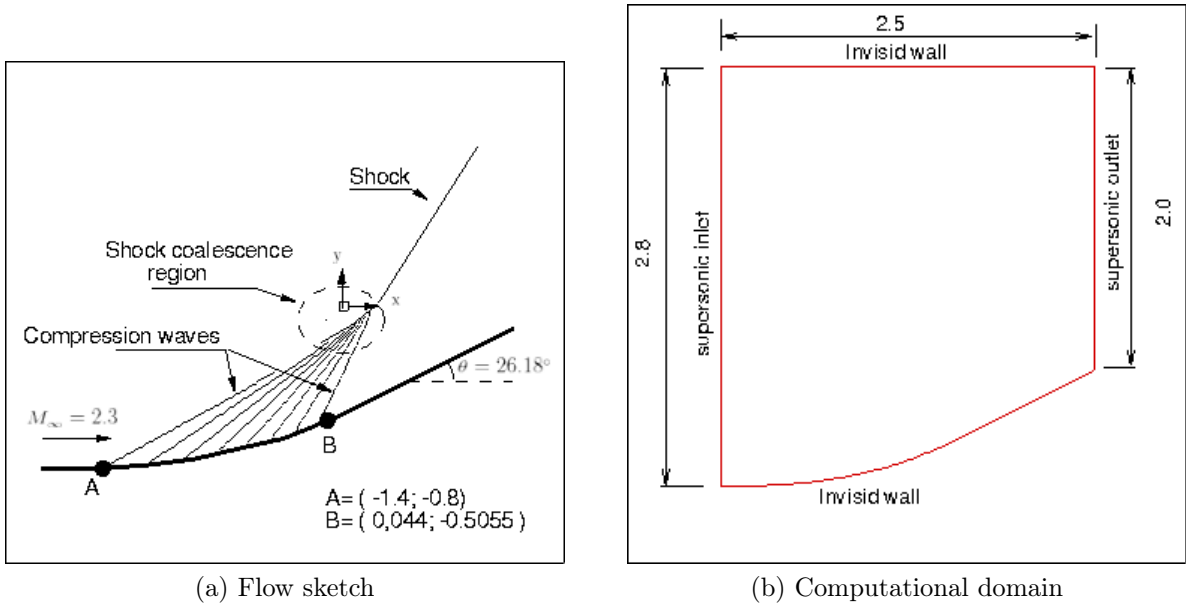


Figure 15: Shock formation due to the coalescence of compression waves.

Figure 16 compares the Mach iso-contour lines computed with the shock-capturing and shock-fitting approaches using the `eulfs` CFD solver.

Inspection of Fig.16b, which refers to the shock-fitting calculation, clearly reveals that the fitted shock-mesh (shown using a white solid line) starts well ahead of the point where the characteristics merge into the shock. This is because, in Moretti’s words [90]: “Premature fitting of the shock in the region where compression waves tend to coalesce is not harmful at all, provided that the shock behaves as one of the characteristic surfaces coalescing into a finite discontinuity”. Modeling shock-formation as described by Moretti requires an *ad-hoc* calculation of the normal to the shock within the end-point of the shock-mesh, so that the present test-case checks that this functionality works correctly.

4.8. Transonic flow past a NACA0012 profile (`NACA0012_M080_A0`)

The present test-case deals with the two-dimensional, inviscid, transonic flow past the NACA 0012 airfoil at $\alpha_\infty = 0^\circ$ degrees angle of attack and free-stream Mach number equal to $M_\infty = 0.80$. The profile is placed in the middle of a $20c \times 20c$ square domain where the characteristic length c is the profile’s chord, see Fig. 17. A symmetric flow-field has been chosen since it allows to compare the shock-capturing and shock-fitting approaches using a single numerical simulation: as shown in Fig. 17b, the shock on the upper side of the profile has been fitted, whereas the one on the lower side has been captured. Figure 17a shows the background triangulation which has been generated using `Triangle` [53, 54] and features 9912 triangles and 5024 mesh-points, 78 of which are placed along the airfoil’s profile. Grid-points have been clustered close to the airfoil (see Fig. 17a), in particular close to the leading-edge and trailing-edge (see Fig. 17b). Figure 17b shows the computational mesh at steady state: the mesh above the profile has been modified by the shock-fitting algorithm to

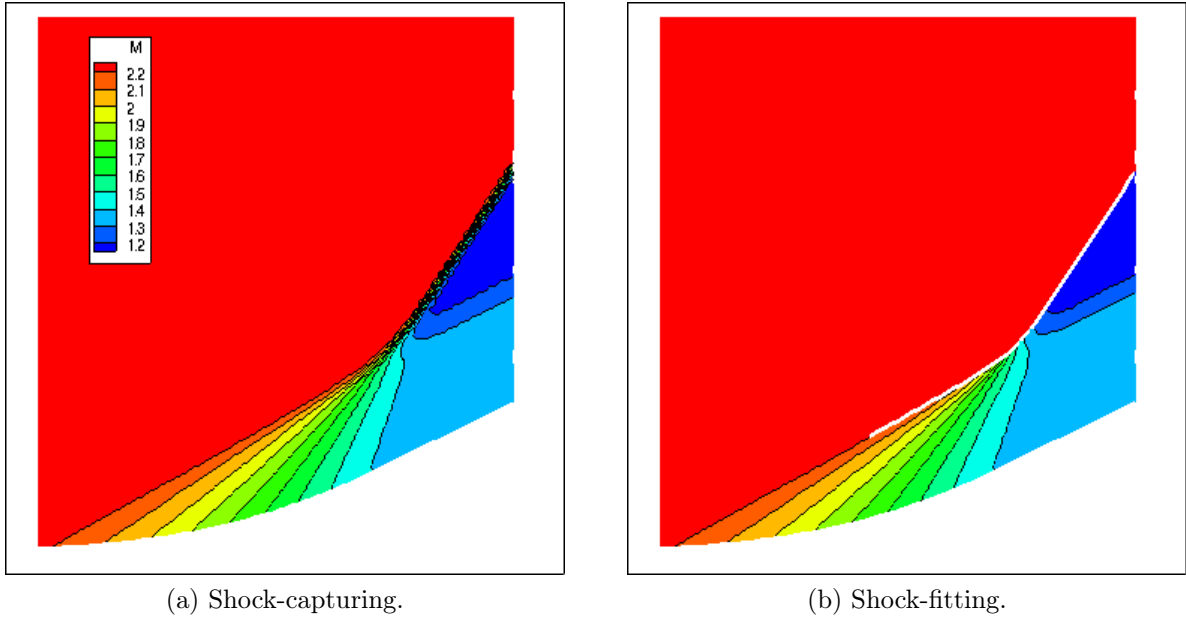


Figure 16: Shock formation due to the coalescence of compression waves: Mach iso-contour lines.

accommodate the shock-mesh and it only differs from the background triangulation in the immediate neighbourhood of the discontinuity; the mesh below the profile coincides with the background triangulation.

As shown in Fig. 17b, the shock-mesh is bounded by the foot and the tip of the shock. A close-up of the Mach number iso-contour lines close to the tip is shown in Fig. 17d, which reveals that the shock is formed through the coalescence of those characteristic curves of the steady Euler equations that form an angle μ with respect to the streamline, μ being the Mach angle. These same features could hardly be seen in a shock-capturing solution, unless a much finer, or feature-adapted, mesh were used.

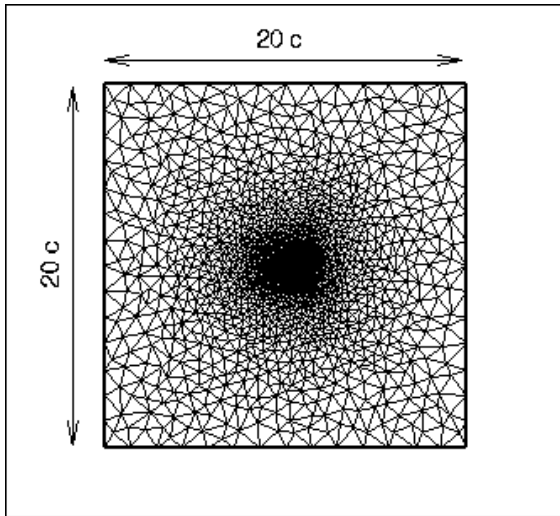
The numerical treatment of the tip of the shock is identical to that described in Sect. 4.7 so that the present test-case checks that the shock-fitting algorithm handles correctly the foot of the shock, i.e. a normal shock impinging on a curved wall.

Finally, Fig. 17c allows to compare the fitted and captured shocks using grids of nearly identical spatial resolution: it is clear that shock-fitting provides a much more realistic shock thickness than shock-capturing. Further details concerning this test case can be found in [91].

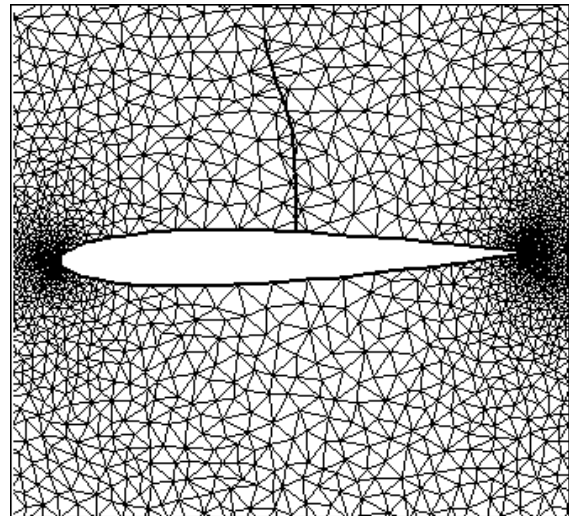
4.9. Shock-vortex interaction (*ShockVortex*)

This last test-case, which is used to verify that *UnDiFi-2D* works correctly also when dealing with unsteady flows, features the interaction between a moving vortex and a standing shock.

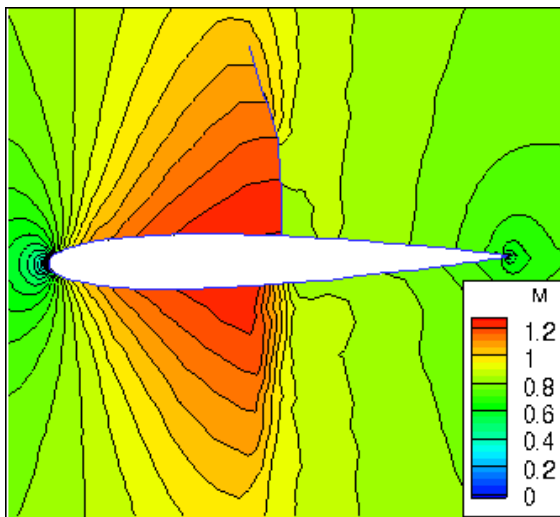
Figure 18 shows the computational domain along with the boundary and initial conditions.



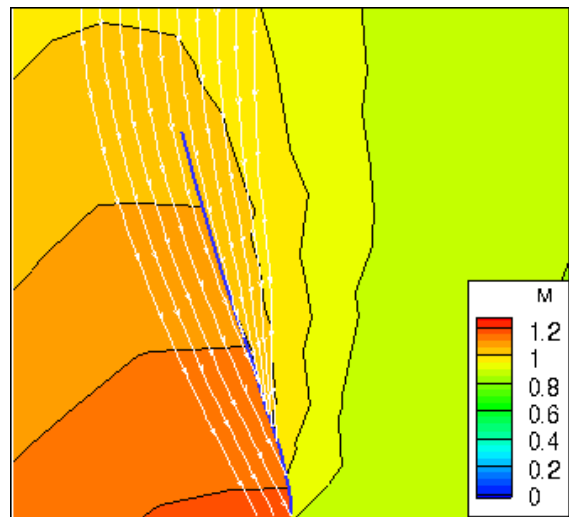
(a) Background triangulation.



(b) Computational mesh at steady-state.



(c) Zoom around the profile.



(d) Close-up of the tip of the shock.

Figure 17: Transonic flow past the NACA0012 profile: meshes and Mach number iso-contour lines.

The flow-field is initialized by adding the perturbation velocity field induced by the vortex to the uniform flow past a steady normal shock. As shown in Fig. 18, at the initial time $t = 0$, the vortex is located 0.2 unit lengths L ahead of the standing shock.

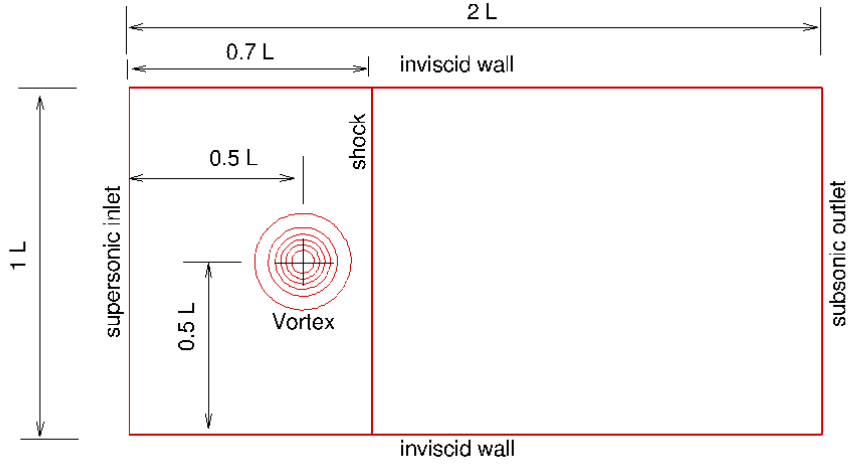


Figure 18: Shock–vortex interaction: computational domain and initial condition.

Using a cylindrical reference frame attached to the vortex core, the perturbation velocity field reads:

$$\tilde{u}_\theta = -\epsilon |\mathbf{u}_\infty| \tau \exp^{\alpha(1-\tau^2)} \quad (17a)$$

$$\tilde{u}_r = 0 \quad (17b)$$

where the dimensionless radial distance from the vortex core, $\tau = \frac{r}{r_c}$, has been used in Eq. (17a), with $r_c = 0.05 L$. The two dimensionless parameters α and ϵ , which respectively control the width and magnitude of the velocity perturbation, are mutually related via the shock and vortex Mach numbers:

$$\epsilon = \frac{M_v}{M_s} \frac{\sqrt{2\alpha}}{\exp(\alpha - \frac{1}{2})} \quad (18)$$

where:

$$M_s = \frac{|\mathbf{u}_\infty|}{a_\infty} \quad M_v = \frac{\max |u_\theta|}{a_\infty} \quad (19)$$

For the chosen pair of shock and vortex Mach numbers: $M_s = 2$, $M_v = 0.2$, which gives rise to a *weak* shock-vortex interaction, according to the nomenclature of [92], the vortex strength $\epsilon \approx 8.6 \cdot 10^{-2}$ follows from Eq. (18), having set $\alpha = 0.204$.

The Delaunay triangulation of the computational domain was generated using `Triangle`; it features 217569 grid-points and 433664 triangles and a mesh spacing along the boundaries equal to $h/L = 0.00375$. The shock-capturing and shock-fitting simulations were performed using the `NEO` solver. A qualitative comparison between the two shock-modeling options is given in Fig. 19, where total enthalpy iso-contour lines at three subsequent time instants are shown: shock-capturing on the top row and shock-fitting on the bottom row. In particular, besides the oscillations related to the approximation of the shock, we can see clearly that the contours downstream of the discontinuity are much less smooth in the captured solution.

The fitted computations, on the other hand, show very nice and smooth contours. Further details about this test-case and further simulations dealing with shock-vortex interactions can be found in [26, 27].

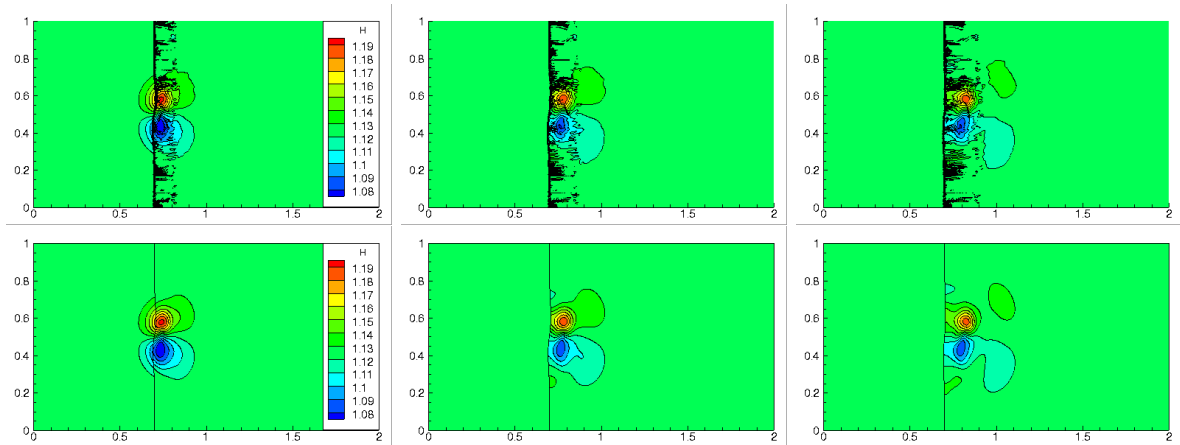


Figure 19: Shock-vortex interaction: total enthalpy contours at times $t = 0.3$ (left), $t = 0.4$ (centre) and $t = 0.5$ (right) using the LDA scheme; shock-capturing in the upper row of frames and shock-fitting in the lower row.

5. Concluding remarks and future perspectives

In this article we have presented the state-of-the-art of a 10-year-long development of a shock-fitting technique for unstructured meshes.

We have described the key algorithmic features of the technique, which has been implemented in an open-source code, available in a dedicated repository. A fairly extensive selection of ready-to-run test-cases demonstrates the features and current capabilities of the code, which we have shown to be able to deal with compressible flows featuring either isolated or mutually interacting discontinuities. The superior quality of fitted shock-waves over captured ones has also been emphasized.

A number of unsolved issues remain to be addressed, as detailed hereafter.

In all flow-cases presented in this paper the effects of viscosity have been neglected. When dealing with viscous flow, shocks cease to be discontinuities in a mathematical sense, and have a finite thickness. At high Reynolds number, however, the width of a shock is comparable to the *microscopic* length-scales, hence orders of magnitude smaller than any *macroscopic* length scale, such as Kolmogorov's, for instance. Under this circumstances, it is still appropriate to ignore the inner shock-structure and consider shock-waves as having zero thickness. The capability of dealing with shock/boundary-layer interactions is work in progress [28] and it is likely going to be introduced in a (near) future release of the project.

When dealing with complex shock-shock and/or shock-boundary interactions in *steady* flows, a preliminary shock-capturing calculation, followed by an automatic shock-detection and shock-pattern identification strategy [93], is capable of supplying a reasonably good initial guess to the shock-fitting algorithm.

When dealing with *un-steady* flows, however, things get much harder and the currently unsolved issues that remain to be addressed are well summarized in a 1986 paper by Glimm and co-workers [10]:

1. Treating changes of the topology of regions bounded by fronts from simply connected to multiply connected regions.
2. Treating the disappearance of weakening fronts and the appearance of new fronts at boundaries or at collisions of other fronts

As far as the first issue is concerned, changes in the shock topology may occur in unsteady flows for several reasons. For example, new discontinuities may appear when a shock-wave moving along a wall encounters an abrupt change in the slope of the wall or when two discontinuities begin to interact. As far as the second issue is concerned, beside the formation of a shock due to the steepening of compression waves, a mechanism which we have examined in Sect. 4.7, an existing shock may progressively weaken and finally disappear. In order to be able to manage all these topological changes it will be necessary to develop new algorithmic tools capable of detecting the occurrence of a change in the shock-topology and modify accordingly the fitted discontinuities and their mutual interactions. Some tests have been carried out on specific flow configurations, as shown in [94, 95], but the development of a general purpose tool will not be trivial and will probably require the use of advanced, multi-disciplinary techniques such as those used in [93]. The effort which will be necessary to complete the development of the unstructured, shock-fitting technique and bring it to full maturity, is not modest and probably requires the merging of different skills and expertise. In order to be successful, it will be necessary to broaden the audience of developers and technical expertise involved. With such a goal in mind, we launch the present project.

5.1. Key features

Usability, modularity, maintenance and enhancement are the key features of the *UnDiFi-2D* project. In particular, a comprehensive documentation is provided by means of exhaustive source files comments, an in-depth user-manual and a selection of ready-to-run test-cases. Moreover, in order to demonstrate code modularity, the shock-fitting algorithm has been coupled with two different CFD solvers (NEO and Eu1FS) that can be alternatively selected by the user at run-time.

5.2. Known issues

An overview of the currently open issues and future lines of development has been given in the opening of Sect. 5. It is also worth emphasizing that *UnDiFi-2D* is developed and maintained by a team of CFD developers, rather than computer scientists, meaning that *UnDiFi-2D* is designed to be as simple as possible. For instance, no effort has been made to improve the efficiency of the algorithm by introducing any form of parallelism. At present, the main computational bottleneck is due to the use of disk I/O for communicating among the different building blocks of the algorithm, as described in Sect. 1.6. In this sense, a better performing infrastructure may be devised in future releases, but up to now it has

been considered as a secondary research branch with respect to the algorithmic development. Finally, modern features of the Fortran language may be exploited in the future to enhance the intrinsic modularity of the algorithm.

5.3. Perspectives

The reported examples of applications and the corresponding bibliographic references, have shown that *UnDiFi-2D* is a mature project able to already provide superior results in terms of accuracy for many test-cases with respect to state-of-the-art shock-capturing codes. However, *UnDiFi-2D* is a relatively young project with some important limitations. Nevertheless, the software platform that we have presented in the article represents a solid starting point for developments to overcome these limitations in a collaborative framework.

References

- [1] H. G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, *Computers in Physics* 12 (6) (1998) 620–631. doi:10.1063/1.168744.
- [2] F. Palacios, J. Alonso, K. Duraisamy, M. Colonna, J. Hicken, A. Aranake, A. Campos, S. Copeland, T. Economon, A. Lonkar, T. Lukaczyk, T. Taylor, Stanford University Unstructured (SU²): An open-source integrated computational environment for multi-physics simulation and design, in: 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, American Institute of Aeronautics and Astronautics, 2013. doi:10.2514/6.2013-287.
- [3] J. VonNeumann, R. D. Richtmyer, A method for the numerical calculation of hydrodynamic shocks, *Journal of Applied Physics* 21 (3) (1950) 232–237. doi:10.1063/1.1699639.
- [4] G. Moretti, Intellectual consumerism, *Meccanica* 33 (5) (1998) 524–531. doi:10.1023/A:1004376728032.
- [5] H. W. Emmons, The numerical solution of compressible fluid flow problems, NACA-TN 932, NASA (1944).
URL <https://apps.dtic.mil/dtic/tr/fulltext/u2/b814416.pdf>
- [6] H. W. Emmons, Flow of a compressible fluid past a symmetrical airfoil in a wind tunnel and in free air, NACA-TN 1746 (1948).
URL <https://ntrs.nasa.gov/citations/19930082372>
- [7] G. Moretti, M. Abbett, A time-dependent computational method for blunt body flows., *AIAA Journal* 4 (1966) 2136–41. doi:10.2514/2.6898.
- [8] J. Glimm, E. Isaacson, D. Marchesin, O. McBryan, Front tracking for hyperbolic systems, *Advances in Applied Mathematics* 2 (1) (1981) 91–119. doi:[https://doi.org/10.1016/0196-8858\(81\)90040-3](https://doi.org/10.1016/0196-8858(81)90040-3).
- [9] J. Glimm, C. Klingenberg, O. McBryan, B. Plohr, D. Sharp, S. Yaniv, Front tracking and two-dimensional riemann problems, *Advances in Applied Mathematics* 6 (3) (1985) 259 – 290. doi:[https://doi.org/10.1016/0196-8858\(85\)90014-4](https://doi.org/10.1016/0196-8858(85)90014-4).
- [10] I.-L. Chern, J. Glimm, O. McBryan, B. Plohr, S. Yaniv, Front tracking for gas dynamics, *Journal of Computational Physics* 62 (1) (1986) 83 – 110. doi:[https://doi.org/10.1016/0021-9991\(86\)90101-4](https://doi.org/10.1016/0021-9991(86)90101-4).
- [11] G. Moretti, Thirty-six years of shock fitting, *Computers & Fluids* 31 (4-7) (2002) 719–723. doi:10.1016/s0045-7930(01)00072-x.
- [12] M. Pandolfi, D. D’Ambrosio, Numerical instabilities in upwind methods: analysis and cures for the “carbuncle” phenomenon, *Journal of Computational Physics* 166 (2) (2001) 271–301. doi:10.1006/jcph.2000.6652.
- [13] M. H. Carpenter, J. H. Casper, Accuracy of shock capturing in two spatial dimensions, *AIAA Journal* 37 (9) (1999) 1072–1079. doi:10.2514/2.835.

- [14] E. Johnsen, J. Larsson, A. V. Bhagatwala, W. H. Cabot, P. Moin, B. J. Olson, P. S. Rawat, S. K. Shankar, B. Sjögren, H. Yee, X. Zhong, S. K. Lele, Assessment of high-resolution methods for numerical simulations of compressible turbulence with shock waves, *Journal of Computational Physics* 229 (4) (2010) 1213 – 1237. doi:10.1016/j.jcp.2009.10.028.
- [15] S. Pirozzoli, Numerical methods for high-speed flows, *Annual Review of Fluid Mechanics* 43 (1) (2011) 163–194. doi:10.1146/annurev-fluid-122109-160718.
- [16] X. Zhong, X. Wang, Direct numerical simulation on the receptivity, instability, and transition of hypersonic boundary layers, *Annual Review of Fluid Mechanics* 44 (1) (2012) 527–561. doi:10.1146/annurev-fluid-120710-101208.
- [17] C. M. Romick, T. D. Aslam, High-order shock-fitted detonation propagation in high explosives, *Journal of Computational Physics* 332 (2017) 210 – 235. doi:https://doi.org/10.1016/j.jcp.2016.11.049.
- [18] F. Nasuti, M. Onofri, *Steady and Unsteady Shock Interactions by Shock Fitting Approach*, Springer International Publishing, Cham, 2017, pp. 33–55. doi:10.1007/978-3-319-68427-7_2.
- [19] P. S. Rawat, X. Zhong, On high-order shock-fitting and front-tracking schemes for numerical simulation of shock-disturbance interactions, *Journal of Computational Physics* 229 (19) (2010) 6744 – 6780. doi:10.1016/j.jcp.2010.05.021.
- [20] R. Paciorri, A. Bonfiglioli, A shock-fitting technique for 2d unstructured grids, *Computers & Fluids* 38 (3) (2009) 715 – 726. doi:10.1016/j.compfluid.2008.07.007.
- [21] M. S. Ivanov, A. Bonfiglioli, R. Paciorri, F. Sabetta, Computation of weak steady shock reflections by means of an unstructured shock-fitting solver, *Shock Waves* 20 (4) (2010) 271–284. URL <https://doi.org/10.1007/s00193-010-0266-y>
- [22] R. Paciorri, A. Bonfiglioli, Shock interaction computations on unstructured, two-dimensional grids using a shock-fitting technique, *Journal of Computational Physics* 230 (8) (2011) 3155 – 3177. doi: <http://dx.doi.org/10.1016/j.jcp.2011.01.018>.
- [23] A. Bonfiglioli, R. Paciorri, Convergence analysis of shock-capturing and shock-fitting solutions on unstructured grids, *AIAA J.* 52 (7) (2014) 1404–1416. doi:10.2514/1.J052567.
- [24] R. Pepe, A. Bonfiglioli, A. D’Angola, G. Colonna, R. Paciorri, Shock-fitting versus shock-capturing modeling of strong shocks in nonequilibrium plasmas, *Plasma Science, IEEE Transactions on* 42 (10) (2014) 2526–2527. doi:10.1109/TPS.2014.2324493.
- [25] R. Pepe, A. Bonfiglioli, R. Paciorri, A. Lani, J. Garicano-Mena, C. F. Ollivier-Gooch, *Towards a Modular Approach for Unstructured Shock-Fitting*, International Center for Numerical Methods in Engineering, Barcelona, Spain, 2014. URL <http://wccm-ecfd2014.org/admin/files/filePaper/p3220.pdf>
- [26] A. Bonfiglioli, R. Paciorri, L. Campoli, Unsteady shock-fitting for unstructured grids, *International Journal for Numerical Methods in Fluids* 81 (4) (2016) 245–261. doi:10.1002/flid.4183.
- [27] L. Campoli, P. Quemar, A. Bonfiglioli, M. Ricchiuto, Shock-fitting and predictor-corrector explicit ale residual distribution, in: M. Onofri, R. Paciorri (Eds.), *Shock Fitting: Classical Techniques, Recent Developments, and Memoirs of Gino Moretti*, Springer International Publishing, Cham, 2017, pp. 113–129. doi:10.1007/978-3-319-68427-7_5.
- [28] A. Assonitis, R. Paciorri, A. Bonfiglioli, Numerical simulation of shock boundary layer interaction using shock fitting technique, *Lecture Notes in Mechanical Engineering* (2020) 124–134doi:10.1007/978-3-030-41057-5_10.
- [29] A. Lani, V. De Amicis, Sf: An open source object-oriented platform for unstructured shock-fitting methods, in: M. Onofri, R. Paciorri (Eds.), *Shock Fitting: Classical Techniques, Recent Developments, and Memoirs of Gino Moretti*, Springer International Publishing, Cham, 2017, pp. 85–112. doi:10.1007/978-3-319-68427-7_4.
- [30] A. Lani, al., COOLFluid Wiki page (2020). URL <https://github.com/andrealani/COOLFluid/wiki>
- [31] T. Song, A. Main, G. Scovazzi, M. Ricchiuto, The shifted boundary method for hyperbolic systems: Embedded domain computations of linear waves and shallow water flows, *Journal of Computational Physics* 369 (2018) 45–79. doi:10.1016/j.jcp.2018.04.052.

- [32] M. Ciallella, M. Ricchiuto, R. Paciorri, A. Bonfiglioli, Extrapolated shock tracking: Bridging shock-fitting and embedded boundary methods, *Journal of Computational Physics* 412 (2020) 109440. doi:<https://doi.org/10.1016/j.jcp.2020.109440>.
- [33] A. Bonfiglioli, M. Grottadaurea, R. Paciorri, F. Sabetta, An unstructured, three-dimensional, shock-fitting solver for hypersonic flows, *Computers & Fluids* 73 (0) (2013) 162–174. doi:[10.1016/j.compfluid.2012.12.022](https://doi.org/10.1016/j.compfluid.2012.12.022).
- [34] D. W. Zaide, C. F. Ollivier-Gooch, Inserting a curve into an existing two dimensional unstructured mesh, in: *Proceedings of the 22nd International Meshing Roundtable*, Springer, 2014, pp. 93–107.
- [35] D. W. Zaide, C. F. Ollivier-Gooch, Inserting a surface into an existing unstructured mesh, *Int. J. Numer. Methods Eng* 106 (6) (2016) 484–500. doi:[10.1002/nme.5132](https://doi.org/10.1002/nme.5132).
- [36] D. Zaide, C. Ollivier-Gooch, *Inserting a Shock Surface into an Existing Unstructured Mesh*, Springer International Publishing, Cham, 2017. doi:[10.1007/978-3-319-68427-7_7](https://doi.org/10.1007/978-3-319-68427-7_7).
- [37] J. Liu, D. Zou, *A Shock-Fitting Technique for ALE Finite Volume Methods on Unstructured Dynamic Meshes*, Springer International Publishing, Cham, 2017, pp. 131–149. doi:[10.1007/978-3-319-68427-7_6](https://doi.org/10.1007/978-3-319-68427-7_6).
- [38] D. Zou, C. Xu, H. Dong, J. Liu, A shock-fitting technique for cell-centered finite volume methods on unstructured dynamic meshes, *Journal of Computational Physics* 345 (2017) 866 – 882. doi:<https://doi.org/10.1016/j.jcp.2017.05.047>.
- [39] S. Chang, X. Bai, D. Zou, Z. Chen, J. Liu, An adaptive discontinuity fitting technique on unstructured dynamic grids, *Shock Waves* 29 (8) (2019) 1103–1115. doi:[10.1007/s00193-019-00913-3](https://doi.org/10.1007/s00193-019-00913-3).
- [40] D. Zou, A. Bonfiglioli, R. Paciorri, J. Liu, An embedded shock-fitting technique on unstructured dynamic grids, *Computers & Fluids* 218 (2021) 104847. doi:<https://doi.org/10.1016/j.compfluid.2021.104847>.
- [41] L. M. D’Aquila, B. T. Helenbrook, A. Mazaheri, A novel stabilization method for high-order shock fitting with finite element methods, *Journal of Computational Physics* (2021) 110096doi:[10.1016/j.jcp.2020.110096](https://doi.org/10.1016/j.jcp.2020.110096).
- [42] M. Zahr, P.-O. Persson, An optimization-based approach for high-order accurate discretization of conservation laws with discontinuous solutions, *Journal of Computational Physics* 365 (2018) 105 – 134. doi:<https://doi.org/10.1016/j.jcp.2018.03.029>.
- [43] M. J. Zahr, A. Shi, P.-O. Persson, Implicit shock tracking using an optimization-based high-order discontinuous galerkin method, *Journal of Computational Physics* (2020) 109385doi:[10.1016/j.jcp.2020.109385](https://doi.org/10.1016/j.jcp.2020.109385).
- [44] A. Corrigan, A. D. Kercher, D. A. Kessler, A moving discontinuous Galerkin finite element method for flows with interfaces, *International Journal for Numerical Methods in Fluids* 89 (9) (2019) 362–406. doi:[10.1002/flid.4697](https://doi.org/10.1002/flid.4697).
- [45] A. D. Kercher, A. Corrigan, D. A. Kessler, The moving discontinuous galerkin finite element method with interface condition enforcement for compressible viscous flows, *International Journal for Numerical Methods in Fluids* n/a (n/a). doi:<https://doi.org/10.1002/flid.4939>.
- [46] **FronTier++**: a C/C++ library for front tracking.
URL http://www.ams.sunysb.edu/~linli/FronTier++_Manual/index.html
- [47] J. Glimm, M. Graham, J. Grove, X. Li, T. Smith, D. Tan, F. Tangerman, Q. Zhang, Front tracking in two and three dimensions, *Computers & Mathematics with Applications* 35 (7) (1998) 1–11, advanced Computing on Intel Architectures. doi:[https://doi.org/10.1016/S0898-1221\(98\)00028-5](https://doi.org/10.1016/S0898-1221(98)00028-5).
- [48] J. Glimm, J. W. Grove, X. L. Li, K.-m. Shyue, Y. Zeng, Q. Zhang, Three-dimensional front tracking, *SIAM Journal on Scientific Computing* 19 (3) (1998) 703–727. doi:[10.1137/S1064827595293600](https://doi.org/10.1137/S1064827595293600).
- [49] J. Glimm, J. W. Grove, X. L. Li, N. Zhao, Simple front tracking, in: *Nonlinear partial differential equations* (Evanston, IL, 1998), Vol. 238 of *Contemp. Math.*, Amer. Math. Soc., Providence, RI, 1999, pp. 133–149. doi:[10.1090/conm/238/03544](https://doi.org/10.1090/conm/238/03544).
- [50] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, L. Wu, A simple package for front tracking, *Journal of Computational Physics* 213 (2) (2006) 613–628. doi:<https://doi.org/10.1016/j.jcp.2005.08.034>.
- [51] D. She, R. Kaufman, H. Lim, J. Melvin, A. Hsu, J. Glimm, Chapter 15 - front-tracking methods, in:

- R. Abgrall, C.-W. Shu (Eds.), *Handbook of Numerical Methods for Hyperbolic Problems*, Vol. 17 of *Handbook of Numerical Analysis*, Elsevier, 2016, pp. 383–402. doi:<https://doi.org/10.1016/bs.hna.2016.07.004>.
- [52] The FLASH code, Flash Center for Computational Science at the University of Chicago (2021). URL <http://flash.uchicago.edu/site/flashcode/>
- [53] J. R. Shewchuk, Triangle: A two-dimensional quality mesh generator and delaunay triangulator. (2005). URL <http://www.cs.cmu.edu/~quake/triangle.html>
- [54] J. R. Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in: M. C. Lin, D. Manocha (Eds.), *Applied Computational Geometry: Towards Geometric Engineering*, Vol. 1148 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry. doi:10.1007/BFb0014497.
- [55] John Burkardt’s home page, last accessed: January 19, 2021. URL <https://people.sc.fsu.edu/~jburkardt/index.html>
- [56] A. Bonfiglioli, Fluctuation splitting schemes for the compressible and incompressible Euler and Navier-Stokes equations, *Int. J. Comput. Fluid Dyn.* 14 (1) (2000) 21–39. doi:10.1080/10618560008940713.
- [57] A. Bonfiglioli, R. Paciorri, A mass-matrix formulation of unsteady fluctuation splitting schemes consistent with Roe’s parameter vector, *International Journal of Computational Fluid Dynamics* 27 (4-5) (2013) 210–227. doi:10.1080/10618562.2013.813491.
- [58] L. Arpaia, M. Ricchiuto, R. Abgrall, An ALE formulation for explicit Runge-Kutta residual distribution, *Journal of Scientific Computing* (2014) 1–46doi:10.1007/s10915-014-9910-5.
- [59] M. Ricchiuto, R. Abgrall, Explicit Runge-Kutta residual distribution schemes for time dependent problems: Second order case, *Journal of Computational Physics* 229 (16) (2010) 5653 – 5691. doi:10.1016/j.jcp.2010.04.002.
- [60] M. Ricchiuto, An explicit residual based approach for shallow water flows, *Journal of Computational Physics* 280 (2015) 306 – 344. doi:<https://doi.org/10.1016/j.jcp.2014.09.027>.
- [61] H. N. (as K. Masatsuka), *I Do Like CFD*, 2013, ISBN 9781304827937, www.cfdbooks.com/.
- [62] J. Anderson, *Fundamentals of Aerodynamics*, McGraw-Hill Education, 2010. URL <https://books.google.it/books?id=xwY8PgAACAAJ>
- [63] M. Onofri, R. Paciorri, *Shock Fitting: Classical Techniques, Recent Developments, and Memoirs of Gino Moretti*, Springer, 2017.
- [64] E. Tadmor, A minimum entropy principle in the gas dynamics equations, *Applied Numerical Mathematics* 2 (3) (1986) 211 – 219, special Issue in Honor of Milt Rose’s Sixtieth Birthday. doi:10.1016/0168-9274(86)90029-2.
- [65] R. Abgrall, M. Ricchiuto, *High-Order Methods for CFD*, John Wiley & Sons, Ltd, 2017, pp. 1–54. doi:10.1002/9781119176817.ecm2112.
- [66] H. Deconinck, M. Ricchiuto, *Residual Distribution Schemes: Foundations and Analysis*, John Wiley & Sons, Ltd, 2017, pp. 1–53. doi:10.1002/0470091355.ecm054.
- [67] R. Abgrall, K. Mer, B. Nkonga, A Lax–Wendroff type theorem for residual schemes, in: M. Hafez, J. Chattot (Eds.), *Innovative methods for numerical solutions of partial differential equations*, World Scientific, 2002, pp. 243–266. doi:10.1142/9789812810816.
- [68] R. Abgrall, P. L. Roe, High order fluctuation schemes on triangular meshes, *Journal of Scientific Computing* 19 (1) (2003) 3–36. doi:10.1023/A:1025335421202.
- [69] P. Roe, Approximate riemann solvers, parameter vectors, and difference schemes, *Journal of Computational Physics* 43 (2) (1981) 357–372. doi:10.1016/0021-9991(81)90128-5.
- [70] H. Deconinck, P. Roe, R. Struijs, A multidimensional generalization of Roe’s flux difference splitter for the euler equations, *Computers & Fluids* 22 (2-3) (1993) 215–222. doi:10.1016/0045-7930(93)90053-c.
- [71] A. Csík, M. Ricchiuto, H. Deconinck, A conservative formulation of the multidimensional upwind residual distribution schemes for general nonlinear conservation laws, *Journal of Computational Physics* 179 (1) (2002) 286–312. doi:<https://doi.org/10.1006/jcph.2002.7057>.
- [72] M. Ricchiuto, A. Csík, H. Deconinck, Residual distribution for general time-dependent conservation

- laws, *Journal of Computational Physics* 209 (1) (2005) 249–289. doi:10.1016/j.jcp.2005.03.003.
- [73] R. Abgrall, Toward the ultimate conservative scheme : Following the quest, *Journal of Computational Physics* 167 (2) (2001) 277–315. doi:10.1006/jcph.2000.6672.
- [74] E. van der Weide, H. Deconinck, Positive matrix distribution schemes for hyperbolic systems, in: *Computational Fluid Dynamics*, Wiley, New York, 1996, pp. 747–753.
- [75] R. Abgrall, M. Mezone, Construction of second-order accurate monotone and stable residual distribution schemes for steady problems, *Journal of Computational Physics* 195 (2) (2004) 474 – 507. doi:https://doi.org/10.1016/j.jcp.2003.09.022.
- [76] P. L. Roe, Linear advection schemes on triangular meshes, Tech. Rep. CoA 8720, Cranfield Institute of Technology (1987).
- [77] P. L. Roe, “optimum” upwind advection on a triangular mesh, Tech. Rep. ICASE 90-75, ICASE (1990).
- [78] P. Roe, D. Sidilkover, Optimum positive linear schemes for advection in two and three dimensions., *SIAM J. Numer. Anal.* 29 (6) (1992) 1542–1568. doi:10.1137/0729089.
- [79] H. Paillere, H. Deconinck, Multidimensional upwind residual distribution schemes for the 2d euler equations, in: H. Deconinck, B. Koren (Eds.), *Notes on Numerical Fluid Mechanics*, Vieweg-Verlag, Braunschweig, Germany, 1997, pp. 51–112.
- [80] A. Bonfiglioli, H. Deconinck, Multidimensional upwind residual distribution schemes for the 3d euler equations, in: H. Deconinck, B. Koren (Eds.), *Notes on Numerical Fluid Mechanics*, Vieweg-Verlag, Braunschweig, Germany, 1997, pp. 141–185.
- [81] T. J. R. Hughes, G. Scovazzi, T. E. Tezduyar, Stabilized methods for compressible flows, *Journal of Scientific Computing* 43 (3) (2010) 343–368. doi:10.1007/s10915-008-9233-5.
- [82] M. Hubbard, P. Roe, Compact high-resolution algorithms for time-dependent advection on unstructured grids, *International Journal for Numerical Methods in Fluids* 33 (5) (2000) 711–736. doi:10.1002/1097-0363(20000715)33:5<711::AID-FLD27>3.0.CO;2-0.
- [83] A. Csik, H. Deconinck, Space-time residual distribution schemes for hyperbolic conservation laws on unstructured linear finite elements, *International Journal for Numerical Methods in Fluids* 40 (3-4) (2002) 573–581. doi:10.1002/flid.315.
- [84] D. Caraeni, L. Fuchs, Compact third-order multidimensional upwind scheme for Navier-Stokes simulations, *Theoretical and Computational Fluid Dynamics* 15 (6) (2002) 373–401. doi:10.1007/s00162-002-0060-2.
- [85] R. Abgrall, M. Mezone, Construction of second-order accurate monotone and stable residual distribution schemes for unsteady flow problems., *Journal of Computational Physics* 188 (1) (2003) 16 – 55. doi:10.1016/S0021-9991(03)00084-6.
- [86] M. Hubbard, M. Ricchiuto, Discontinuous upwind residual distribution: A route to unconditional positivity and high order accuracy, *Computers & Fluids* 46 (1) (2011) 263 – 269. doi:10.1016/j.compfluid.2010.12.023.
- [87] R. Abgrall, High order schemes for hyperbolic problems using globally continuous approximation and avoiding mass matrices, *Journal of Scientific Computing* 73 (2) (2017) 461–494. doi:10.1007/s10915-017-0498-4.
- [88] R. Abgrall, P. Bacigaluppi, S. Tokareva, High-order residual distribution scheme for the time-dependent euler equations of fluid dynamics, *Computers & Mathematics with Applications* 78 (2) (2019) 274 – 297. doi:https://doi.org/10.1016/j.camwa.2018.05.009.
- [89] J. D. Müller, Grid generation tools: Delaunay triangulation with delaundo.
URL <http://www.ae.metu.edu.tr/tuncer/ae546/prj/delaundo/>
- [90] G. Moretti, On the matter of shock fitting, in: R. D. Richtmyer (Ed.), *Proceedings of the Fourth International Conference on Numerical Methods in Fluid Dynamics*, Springer, Berlin, Heidelberg, 1975, pp. 287–292. doi:10.1007/BFb0019764.
- [91] A. Bonfiglioli, R. Paciorri, Unstructured shock-fitting calculations of the transonic flow in a gas turbine cascade, *Aerotecnica Missili & Spazio* 97 (4) (2018) 189–197. doi:10.1007/BF03406053.
- [92] F. Grasso, S. Pirozzoli, Shock-wave-vortex interactions: Shock and vortex deformations, and sound production, *Theoretical and Computational Fluid Dynamics* 13 (6) (2000) 421–456. doi:10.1007/

s001620050121.

- [93] R. Paciorri, A. Bonfiglioli, Accurate detection of shock waves and shock interactions in two-dimensional shock-capturing solutions, *Journal of Computational Physics* 406 (2020) 109196. doi:10.1016/j.jcp.2019.109196.
- [94] A. Bonfiglioli, R. Paciorri, L. Campoli, V. De Amicis, M. Onofri, Development of an unsteady shock-fitting technique for unstructured grids, in: G. Ben-Dor, O. Sadot, O. Igra (Eds.), *30th International Symposium on Shock Waves 2*, Springer International Publishing, Cham, 2017, pp. 1501–1504. doi:10.1007/978-3-319-44866-4_124.
- [95] R. Paciorri, A. Bonfiglioli, Basic elements of unstructured shock-fitting: Results achieved and future developments, in: M. Onofri, R. Paciorri (Eds.), *Shock Fitting: Classical Techniques, Recent Developments, and Memoirs of Gino Moretti*, Springer International Publishing, Cham, 2017, pp. 59–84. doi:{10.1007/978-3-319-68427-7_3}.