UnDiFi-2D

User Guide

Version 1.4

Dipartimento di Ingegneria Meccanica e Aerospaziale, (DIMA), Sapienza University, Rome, Italy
Scuola di Ingegneria, Università degli Studi della Basilicata, Potenza, Italy
Team CARDAMOM, Inria Bordeaux Sud-Ouest, Talence, France
Saint-Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, Russia

September 17, 2021

Contents

1 Introduction

UnDiFi-2D is an open source Unstructured Discontinuity Fitting algorithm code. It is written in in standard (compliant) Fortran 77/95 with highly modularity as design target The aim of UnDiFi-2D is to explicitely manage discontinuities in the flow field.

In our unstructured shock-fitting approach the shock front is described using a double-sided, polygonal curve. Two sets of flow states, corresponding to the upstream and downstream sides of the discontinuity, are assigned to the grid-points located on either side of the shock front. This is allowed to move, while obeying to the Rankine-Hugoniot jump relations, throughout a background triangular mesh that covers the entire computational domain. At each time step, a local, constrained Delaunay triangulation is applied in the neighbourhood of the shock front to ensure that the edges that make up the shock front are part of the overall triangular grid. The fitted shock acts as an interior boundary for the shock-capturing solver that is used to solve the discretised governing equations in the smooth regions of the flow-field.

A typical shock-fitting calculation starts with the initial shock position guessed using a previous shock-capturing calculation The simulation is then advanced in time until the shock front has reached its steady state position, corresponding to vanishing shock speed.

Three different *in-house* gas-dynamic solvers, have been used so far:

- **EulFS**, developed by Aldo Bonfiglioli [1], [2]
- **NEO**, developed by Mario Ricchiuto at INRIA [3]–[5]
- **COOLFluiD**, developed by Andrea Lani at VKI [6], [7]

All three codes share the same vertex-centered, Fluctuation Splitting [8], [9] discretization of the governing PDEs on linear triangles and tetrahedra. However, virtually any vertex-centered FE or FV solver featuring a linear representation of the dependent variables can be used as gas-dynamic solver within the shock-fitting procedure. Only EulFS and NEO are included in the present repository and documentation.

Three key software components (or modules) can be identified in the unstructured discontinuity-fitting approach:

- Algorithm (described in the paper)
- Solver (Chapter 4)
- Meshing (Chapter 5)

Modularity stems from the fact that these three different components communicate through ad-hoc interfaces. This programming approach may not be the most efficient from the standpoint of computational speed, because, for instance, one has to switch, at each time step, among the different data-structures used by the three different modules. However, the approach is very convenient, since it allows us to use "off the shelf" gas-dynamics solvers and mesh generation tools that are treated as black boxes and can be replaced by similar ones only by changing the interfaces, with a modest coding effort. In the following, we shall describe each module.

1.1 How this documentation is organized

This user guide is organized to both guide the first steps as well as provide a complete overview of the simulation code's features from a user and a developer point of view.

- Chapter 2 contains step by step instructions from obtaining the source code, installation, up to running a first simulation and visualizing the simulation results. In addition, it provides an overview of the whole simulation framework and the currently implemented features.
- Chapter 3 outlines the workflow starting with mesh generation and concluding with the visualization of results produced with UnDiFi-2D.
- Chapter 4 presents main features of the gasdynamic solvers currently coupled with UnDiFi-2D.
- Chapter 5 gives and overview of the meshing software packages used in UnDiFi-2D.
- Simulation tutorials are contained in Chapter 6.

2 Installation

The following chapter describes the installation process on a Linux machine. This also includes the installation of all required prerequisites UnDiFi-2D has been developed on GNU/Linux architectures. Other OS are not supported (and in general there is no better alternative to GNU/Linux :-)

The provided codes (UnDiFi-2D, NEO, EuIFS and auxiliary packages) have been successfully compiled with the following compilers:

- GNU gfortran (version 4.7.0 or higher)
- Intel Fortran Compiler ifort (version 12.0 or higher)

Other compilers are not supported.

The codes are constituted by several modules, therefore, there are many dependences on auxiliary sofwares (see Section 2.3). The easiest way to compile the code is to start with the provided script **compile_all.sh**. It is then necessary that the system has "Make" program (preferably GNU make http://www.gnu.org/software/make/).

2.1 Prerequisites

UnDiFi-2D has been tested for various Linux distributions. This includes Ubuntu 14.04 LTS, 16.04 LTS and 18.04 LTS, 20.04.2 LTS, OpenSUSE 42.1, Leap 15.1, Leap 15.2, Tumbleweed.

The suggested packages in this section can be replaced by self compiled versions. The required packages for the Ubuntu Linux distributions are listed below. Under Ubuntu, they can be obtained using the apt environment:

```
sudo apt-get install git
```

- git
- cmake
- make
- automake
- gfortran
- gcc/g++
- texlive-latex and texlive-lstaddons
- libX11 (see for xorg package)
- LibX11-dev (Ubuntu) LibX11-devel (Opensuse)
- pandoc (version <= 2.9) and pandoc-citeproc
- libkrb5-dev (Ubuntu) / krb5-devel (Opensuse)

Sometimes on some the sytems (based on OpenSuse) the program krb5-config is installed in a directory from which is not possible to run it. In this case is sufficient to add the path /usr/bin/mit/bin to the $PATH variable.

On some systems it may be necessary to increase the size of the stack (part of the memory used to store information about active subroutines) in order to execute UnDiFi-2D correctly. This is done using the command:

```
ulimit -s unlimited
```

from the command line. For convenience, you can add this line to your `.bashrc`.

2.2 Obtaining the source

The UnDiFi-2D repository is available at GitHub. To obtain the most recent version you have two possibilities:

- Clone the UnDiFi-2D repository from Github

  git clone https://github.com/UnDiFi-2D/UnDiFi-2D.git

- Download UnDiFi-2D from Github:

  wget https://github.com/UnDiFi-2D/UnDiFi-2D/archive/master.tar.gz tar xzf master.tar.gz

2.3 Auxiliary softwares

In addition to the shock-capturing gasdynamic solvers, EulFS and NEO, there are several auxiliary codes, contained in `source_utils` directory, which are called during the steps of the UnDiFi-2D algorithm or during the post-processing phase. They are listed below:

- dat2triangle: reads files in EulFS format then writes two input files for Triangle, .node and .poly
- dat2paraview: converts both Neo and EulFS output into Paraview files (*.dat)
- fsplot: performs post-processing for EulFS
- Grid_0: creates the neogrid0.grd file
- Na00x2vvvv: writes the file vvvv_input.dat
- NEO2triangle: reads files in NEO format then writes two input files for Triangle, .node and .poly
- triangle2dat: converts a Triangle mesh to a .dat file format. (The input file for Triangle can be created using dat2triangle)
- Triangle2grd: converts triangle format to .grd format (NEO)
- Triangle: mesh generator for UnDiFi-2D
- Tecplot: post-processing output generator for UnDiFi-2D (optional)
- Petsc: necessary to use EulFS (ver. 3.14.6)

2.4 Compiling the code

- Open a terminal
- Go to the UnDiFi-2D directory
- Compile the code by running the script **./compile_all.sh**

For a list of all compiler options visit Section 3.1. Finally, all the executables are contained in the UnDiFi-2D `bin` directory.

2.5 Directory paths

In order to run a testcase, there is no need to manually set any environment variable since they are automatically set during the compilation. Nevertheless, it may be useful to make the executables system-wise visible. There are at least two different ways to enable it:

1. You can add an alias for the path to your executable. Add a command of the form:

```
alias undifi='$UNDIFIROOT/bin/UnDiFi-2D-2D_x86_64'
```

to the bottom of the file `~/.bashrc`. Source your `~/.bashrc` afterwards with:

```
. ~/.bashrc
```

2. You can add the UnDiFi-2D binary directory to your `$PATH` environment variable by adding:

```
export PATH=$PATH:$UNDIFIROOT/bin
```

to the bottom of the file `~/.bashrc` and sourcing your `~/.bashrc` afterwards.

Now you are ready for the utilization of UnDiFi-2D and all the auxiliary executables.

3 Workflow

In this chapter, the complete process of setting up a simulation with UnDiFi-2D is detailed.

3.1 What happens when running compile_all.sh?

Let's see what happens when the script **compile_all.sh** is executed.

First, we generate the documentation:

```
echo Compiling in doc
cd doc
./makedoc.sh
echo Done!
cd ..
```

Then, f77split and f90split are compiled. They split the modules of a fortran file into separate files. A "module" is a blockdata, function, module, program, or subroutine program subunit.

```
echo Compiling in tools
cd tools
./compile.sh
echo Done!
cd ..
```

Then, several libraries are compiled, specifically:

- libfxdr_2.1
- libport
- libmylib
- libsparse-blas
- SPARSKIT2
- libgeometry
- libtriangulation
- libtirpc

```
echo Compiling in lib
cd lib
make
echo Done!
cd ..
```

Then, the PETSC library is downloaded and compiled. **Note**: it is recommended to use the proposed version of the library in order to avoid issues with the compilation of EulFS. It may be updated in future releases.

The C and Fortran compilers in the configuration settings may be changed. Tested compilers are Intel and GNU pre-installed version of MPI and LAPACK libraries may be used. In this case, the PATH should be provided.

```
echo Compiling PETSC 3.14.6
wget ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.14.6.tar.gz
tar -xvzf petsc-3.14.6.tar.gz
```

```
rm petsc-3.14.6.tar.gz
#
cd petsc-3.14.6
PETSC_DIR1=${PWD}
export PETSC_DIR=${PETSC_DIR1}
./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran --download-mpich --download-
    fblaslapack
#Extract petsc_arch from newest folder name
PETSC_ARC=$(ls -td -- */ | head -n 1 | cut -d'/' -f1)
export PETSC_ARCH=${PETSC_ARC}
#test petsc-3.14.6
make PETSC_DIR=${PETSC_DIR} PETSC_ARCH=${PETSC_ARCH} all
echo Done!
cd ..
```

The gasdynamic solver, EulFS, is then compiled and installed:

```
echo Compiling EulFS 3.14
cd EulFS.3.14

FSPL_DIR1=${PWD}
export FSPL_DIR=${FSPL_DIR1}

mkdir ${FSPL_DIR}/lib/${PETSC_ARCH}
echo ${FSPL_DIR}/lib/$PETSC_ARCH

cd src
make
echo Done!
cd ../..
```

As explained in the previous section, in addition to the shock-capturing gasdynamic solvers, EulFS and NEO, there are several auxiliary codes, contained in the `source_utils` directory, which are called during the steps of the UnDiFi-2D algorithm or during the post-processing phase. In particular, the following packages are compiled and installed:

- dat2triangle: reads files in EulFS format then writes two input files for Triangle, .node and .poly
- dat2paraview: converts both Neo and EulFS output into Paraview files (*.dat)
- Grid_0: creates the neogrid0.grd file
- Na00x2vvvv: writes the file vvvv_input.dat
- Na_creation: creates the triangle mesh files for pre-set testcases
- NEO2triangle: reads files in NEO format then writes two input files for Triangle, .node and .poly
- triangle2dat: converts a Triangle mesh to a .dat file format. The input file for Triangle can be created using dat2triangle
- Triangle2grd: converts triangle format to .grd format (NEO)
- Triangle: mesh generator for UnDiFi-2D

Note: the Tecplot visualizer, may not be necessary as solutions can be outputted in ASCII format and post-processed by Paraview or Visit, open-source softwares.

```
echo Compiling in source_utils
cd source_utils
./compile.sh
echo Done!
cd ..
```

Finally, UnDiFi-2D is compiled and installed.

```
echo Compiling in source
cd source
make install
echo Done!
cd ..
```

3.2 Mesh generation/conversion with Triangle

The shock is treated as an internal boundary by the unstructured gasdynamic solver so that no modifications in the computational kernel of the CFD code are required. Nonetheless, the shock points can freely move over the mesh points of the background grid, as it was done in the floating-shock fitting approaches proposed in the past. To achieve this, the computational mesh is locally re-meshed at each time-step using the background grid and the shock edges and it differs from the background mesh only in the neighbourhood of the shock front, thus keeping the overall number of grid points to a minimum.

For the present release version of UnDiFi-2D, the preferred mesh generator is Triangle, which will be described in Chapter 5.

3.3 Solver settings input files

Before setting up a simulation, depending on the gasdynamic solver used, few parameters must be set in the input files.

3.3.1 NEO

In order to use UnDiFi-2D coupled with NEO, two files must be provided: **type.dat** and **inputfile-exp.txt**.

The ascii file type.dat contains the definition of boundary conditions needed by NEO and it is read by the auxiliary utility converter program Triangle2grd.

```
Face  Type
1      1
2     -1
3      3
4     -1
5     -1
6     -1
7     -1
8     -1
9     -1
10     2

# boundary #1 is the inflow
# boundary #2 is the upper outlow
# boundary #3 is the wall
# boundary #4 is the upper outlow
#
# This file is read by Triangle2grd
#
# bndry code 1 is supersonic
# bndry code -1 is outflow
# bndry code 3 is inviscid wall
```

Note: it is implicitely assumed that the computational domain is made up at most of 10 edges. If more that 10 edges are present, the source code in `Triangle2grd` should be modified accordingly, and faces added in this file.

Note: in the case that other boundary conditions are necessary, they should be implemented in NEO, an additional Type flag added in the source code of `Triangle2grd` and this file, as well.

Note: the type.dat file is not required to use NEO in shock-capturing mode.

The inputfile-exp.txt file defines all the other settings required by NEO, as detailed below. It must be located in the `NEO_data/textinput` folder.

```
          NEO's basic parameters

                 PHYSICS

Model                                        0
Initial state                                0
Initial solution file                 vvvv_input
Problem                                      0
Gamma                                      1.4
Zero density                          0.000001

                 GEOMETRY

External grid file                       neogrid

                 NUMERICS

0(LDA),1(LLF),2(LDAN),3(LWLF),4(LW),5(N)     5
Lumping type: 0 (selective),1 (global)       0
Shield factor                              0.2

                 ALGEBRA

Convergence treshold                      -3.0
Convergence limit                         -13.
Check Steady State                           0
Maximum number of iterations                 2
Convergence variable (0->Neq.s-1)           -1

                  STOP

Final time                              1000000.
Maximum number of time steps                 1
Stop                                         0

                 OUTPUT

Movie (0/1)                                  0
Steps to save                              100
Information Frequency                        1
Output format  (0/1)                         1
Output file                               vvvv
```

For unsteady simulations the file timesteps.dat must be provided for the NEO solver. It contains the time step values used in the Predictor-Corrector time accurate integration method.

```
0.002 # predictor step dt
0.004 # corrector step dt
```

Note: this file is required by the current version of NEO but it may not be necessary in feature releases due to a change in the time accurate integration method.

3.3.2 EulFS

In order to use UnDiFi-2D coupled with EulFS, one file must be provided: **.petscrc**.

```
-equation Euler
-fluid compressible
```

```
-preconditioning No
-nondimensionalisation external

-scalar_scheme NL
-matrix_scheme B

#-scalar_scheme N
#-matrix_scheme N

#-scalar_scheme LDA
#-matrix_scheme LDA

-itmax 1
-ibak 500
-cfl 0.5
-timestepping explicit
-timestep local
-cflmax 1.
-linearization picard

# flow conditions
#
# Mach = 0.63
# 2 degrees of incidence

-freestream_Mach_number 20.0
-flow_angles -1.,0.,0.
-bc_type weak
-colors -1,5,5,4,5,-1,-1,-1,-1,-1,0,-1,-1,-1
-data_dir ./
-restart_file file003.dat
```

The syntax used in this file is defined by the PETSC library. The reader is referred to its documentation for further details.

All other options are set in the following parameter files.

3.4 UnDiFi-2D setup parameter input files

In addition to the files required by the solvers, two other files are necessary for UnDiFi-2D: **input.dat** and **sh00.dat**.

The ascii file input.dat contains the main options to set and tune UnDiFi-2D. A template of this file is as following:

```
0.20d-9   ! EPS
0.30      ! SNDMIN
0.10      ! DXCELL
0.9       ! SHRELAX
50        ! IBAK
1.40d+0   ! GA
0         ! number of additional hole points
0         ! number of periodic boundary
0         ! filter on discontinuity speeds  [0-1.0]  0=disactive
0         ! iteration of mesh topology freezing (must be equal to a backup iteration)
```

EPS represents the minimum relative distance between shock edges

SNDMIN

DXCELL represents an estimate of the cell edge length

SHRELAX allows for a relaxation from/to the captured solution

IBAK sets the solution saving rate

GA is the gas constant

The sh00.dat file contains the structure of the discontinuities present in the flow field at the beginning of the simulation. An example is reported below.

```
1      # number of discontinuities
22 'S' # number and type of points forming the discontinuity.
  2  0.30 0.1000E+01 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  3  0.30 0.9523E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  4  0.30 0.9047E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  5  0.30 0.8571E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  6  0.30 0.8095E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  7  0.30 0.7619E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  8  0.30 0.7142E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
  9  0.30 0.6666E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 10  0.30 0.6190E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 11  0.30 0.5714E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 12  0.30 0.5238E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 13  0.30 0.4761E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 14  0.30 0.4285E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 15  0.30 0.3809E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 16  0.30 0.3333E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 17  0.30 0.2857E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 18  0.30 0.2380E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 19  0.30 0.1904E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 20  0.30 0.1428E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 21  0.30 0.9523E-01 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 22  0.30 0.4761E-01 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 23  0.30 0.0000E+00 0.1639E+01 0.5944E+01 0.1204E+01 0.0 0.1183E+01 0.2958E+01 0.0 0
 24 2        # number of special points
 25 'WPNRX'  # type of special points: wall point normal ...
 26 1 1      # shock 1, ending point 1
 27 'WPNRX'  #
 28 1 2      # shock 1, ending point 2
```

The data points correspond, respectively, to the X and Y coordinates (in 2D) and to the downstream and upstream shock states, in term of Roe variables Z defined as Z $= \sqrt{\rho}$ (1, H, u), where H is the total enthalpy and u the velocity vector.

3.5 Converters

As stated elsewhere, in order to couple the UnDiFi-2D algorithm code with gas-dynamic solvers, it is necessary to provide "converters" which allow data transfer between possibly different formats.

It is important to note that each converter is called during the normal running mode of the UnDiFi-2D code but it can also be used in "stand-alone mode", as described below. This capability can be useful for debugging or testing purposes or when creating or restarting a simulation. In general, a simple script is provided to compile them in the corresponding directory.

3.5.1 Na00x2vvvv

This converter writes the triangle mesh files into the NEO solution data format file, vvvv.dat.

To compile it, just run, for example:

```
gfortran -o na2vvvv main_n2v.f90
```

By default, it read the na00.?.node file and writes the vvvv_input.dat file in the `NEO_data/output` directory.

3.5.2 Na_creation

This program creates na00.node and na00.poly files for a steady planar shock. It could be easily adjusted to generate other testcases.

To compile it, just run, for example:

```
gfortran -o exe_na physical_values.f90 reading_mod.f90 writing_mod.f90 main.f90
```

3.5.3 Grid_0

This program creates the neogrid0.grd which is the grid file used by NEO when running in capturing mode.

To compile it, just run, for example:

```
gfortran -o neogrid0 main.f90
```

3.5.4 NEO2triangle

This converter reads the NEO grid file neogrid.grd in `NEO_data/input` and the solution file vvvv.dat at the current time step, in `NEO_data/output` and writes the corresponding triangle mesh file na00.?.1.node and na00.?.1.poly file. Note that the .poly file does not change respect to the previous step as far as the element connectivity does not change.

To compile it, just run, for example:

```
gfortran -o NEO2triangle main_d2t.f90
```

3.5.5 Triangle2grd

This program converts the mesh triangle file into the NEO (.grd) mesh format. It requires the triangle mesh files `.node`, `.ele`, `edge`, `.neigh`, `.poly` and `type.dat` and writes the `neogrid.grd`.

To compile it, just run, for example:

```
gfortran -o triangle2grd boundary_edge_mod.f90 r_files_mod.f90 main.f90
```

3.5.6 dat2triangle

This program reads files in EulFS format and then writes two input files for triangle: *.node and* .poly. It uses the file *inp* which should be present in the same directory to select the EulFS solution files, typically file001.dat, file002.dat, file003.dat.

3.5.7 triangle2dat

This program converts a triangle mesh to a *.dat* file format used by EulFS. The input file for triangle can be created using dat2triangle. This program needs .edge .neigh and .poly files, hence run:

```
triangle -n -e -p filename.poly
```

where:

- -n outputs (to a .neigh file) a list of triangles neighboring each triangle.
- -e outputs (to an .edge file) a list of edges of the triangulation.

- -p triangulates a Planar Straight Line Graph (.poly file).

For a detailed explanation about all the flags available in Triangle, the user is referred to its main page.

3.6 Simulation

After the mesh generation, compilation of the binary and setup of the parameter files, the code can be executed by running the **start_run_x86.sh** script file present in each testcase directory.

For convenience, a script file is made available, **run.sh** which allows to launch any simulation. Its usage is the following:

```
./run.sh -s Solver -m Mode -f Flow
    -s Solver: neo or eulfs
    -m Mode: capturing or fitting
    -f Flow: steady or unsteady
```

Each IBAK iterations, where the parameter IBAK has been defined in the input.dat file, the solution is saved in a newly created folder named step##### corresponding to the saving rate.

Depending on the gasdynamic solver coupled with UnDiFi-2D, different files will be present in the solution folders.

The following files will be always present in the solution folders.

- na0000?.1.edge:
- na0000?.1.ele:
- na0000?.1.neigh:
- na0000?.1.node:
- na0000?.1.node.BAK:
- na0000?.1.poly:
- na0000?.node:
- na0000?.poly:
- na99.node:
- sh99.dat:
- shocknor.dat:

If EulFS it is used, the following additional files will be present:

- file001.dat:
- file002.dat:
- file003.dat:
- file010.dat:

If NEO it is used, the following additional files will be present:

- neogrid.grd: grid file
- vvvv.dat: flow field solution
- vvvv_input.dat: initial flow field solution

3.7 Post-processing

Solution files generated by NEO are readily available in each solution step folder, step?????. At the moment, the preferred visualizer is Tecplot but Paraview/Visit can also be selected in the inputfile-exp.txt file.

Post-processing of the solutions computed by EulFS into a format suitable for a visualisation is accomplished through fsplot which reads run-time options from a control file named *inp* located in the same directory. More information about fsplot can be found in Chapter 7 of the EulFS user manual present in its doc folder.

To make the conversion process easier, a script file, *plot.sh* is made available in the testcase directory: it requires to specify the solver used for the computation (neo or eulfs option), as shown below. ~~~~~~~ ./plot.sh -s solver ~~~~~~~ Conversion instructions are briefly reported below: in particular, Paraview users have to uncomment the lines which refer to dat2paraview converter: in this case, the code will automatically detect whether an EulFS or Neo output file is stored in the step????? folder. Then a step?????.dat file will be created in the test-case directory and it can be read by Paraview using the Tecplot reader option.

```
 for dir in `ls -d step?????`

do
   echo $dir
   cp inp $dir
   cd $dir
   if [ "$Solver" = "eulfs" ]
   then
     ../../../bin/fsplot-$HOSTTYPE
      preplot file012.dat $dir-eulfs.plt
   elif [ "$Solver" = "neo" ]
   then
      preplot vvvv.dat $dir-neo.plt
   fi

###### FOR PARAVIEW USERS #######
#       ../../../bin/dat2paraview
#         cp paraview.dat ../$dir.dat
##############################

   mv $dir*.plt ..
   cd ..
done
```

This conversion step can be directly incorporated into the run.sh script to make it automatic.

4 Gasdynamic Solvers

In this chapter, the gasdynamic solvers are coupled with UnDiFi, are detailed.

Both solvers, EulFS and NEO, relies on *Fluctuation Splitting* (or *Residual Distribution*) schemes, meaning that the way of discretizing the equations is based on the same approach, though there are few features that differentiate the two codes that will be addressed in the following sub-sections.

Fluctuation Splitting, first introduced in the late 80s by Roe [10] to study scalar convection problems, have emerged as an alternative to Finite Volumes and Finite Elements methods. The introduction of wave modeling, in the same years [11], allowed to adapt the approach to hyperbolic systems, and in particular the compressible Euler equations. Indeed, it consists in decomposing the system into the superposition of contributions from simple waves. Although originally conceived for solving the compressible Euler equations, this class of schemes is applicable to those systems whose inviscid terms are of hyperbolic nature.

In general, the computational domain $\Omega \in \Re^d$ is tessellated into triangles in the 2D space and tetrahedra in 3D. A dual tesselation is also defined, which consists in the medial dual cells, obtained by joining the centroids of gravity of all cells surrounding a given grid-point. Both tesselation are shown in Fig. [RD]: C_i is the median dual cell centered about grid-point i and Ω_e corresponds to triangle e. To simplify the process, the discretization of the inviscid fluxes will be described for the following scalar conservation law:

$$\int_{C_i} \frac{\partial u}{\partial t} dV = \int_{\partial C_i} \vec{F} \cdot d\vec{n} \text{ with } \vec{F} = \vec{a}u \tag{4.1}$$

The surface integral in Eq. 4.1 is written as a weighted sum of the flux integrals of the elements sharing the node:

$$\int_{\partial C_i} \vec{F} \cdot d\vec{n} = \sum_{e \ni i} \beta_i^T \int_{\partial e} \vec{F} \cdot d\vec{n} \tag{4.2}$$

Introducing the inflow parameters:

$$k_i^e = \frac{1}{d}(\vec{a}\vec{n_i^e}) \tag{4.3}$$

where $\vec{n_i^e}$ denotes the normal of the face opposite to vertex i of element e, scaled by its measure. Therefore, the fluctuation ϕ^e reads:

$$\phi^e = \int_{\partial e} \vec{F} \cdot d\vec{n} = \int_{\partial e} \vec{a}u \cdot d\vec{n} = -\sum_{j \in e} k_j^e u_j \tag{4.4}$$

where the summation in Eq.4.4 ranges over the $d+1$ vertices of element e. Replacing Eq. 4.4 in Eq. 4.2, the right hand side of Eq. 4.1 can be written in the follwing form:

$$\int_{\partial C_i} \vec{F} \cdot d\vec{n} = \sum_{e \ni i} \beta_i^e \phi^e \tag{4.5}$$

Thus, it is the choice of the distribution coefficients β_i^e that determines the properties of the discrete solution. Several criteria have been proposed and used in the construction of the discrete schemes such as:

- *Upwinding* An upwind scheme distributes fractions of the cell fluctuation only among its downstream vertices;
- *Positivity* The positivity criterion ensures that the maximum principle holds also at the discrete level;
- *Linearity preservation* Linearity preservation is an accuracy requirement and refers to the ability of the discrete scheme to reproduce exactly a linear polynomial, steady solution of Eq. 4.1.

Residual Distribution concept

Both gasdynamic solvers, EULFS and NEO have the purpose of updating the solution at time level $t + \Delta t$. In particular, they use the computational mesh generated as input that is cut in non-communicating parts by the shock which is treated by the code as if it were an internal boundary.

4.1 EulFS

The EULFS code is an in-house, unstructured CFD solver that has been developed over the last 20 years by Aldo Bonfiglioli (see [1] for a detailed description). This tool is able to work in both two and three space dimensions and stores the solution at the vertices of triangles, in 2-D, and tetrahedra, in 3-D. In both cases, the solution is assumed to vary linearly and continuously in space. The inviscid cell fluctuation ϕ^e is evaluated over each triangular/tetrahedral element e by means of a conservative linearization [Deconinck,Roe,Struijs,1993] based on the parameter vector $Z = (\sqrt{\rho}, \sqrt{\rho}H, \sqrt{\rho}u, \sqrt{\rho}v)^T$ and scattered to the element vertices using signals ϕ_i^e (see Fig. [RD].a). Within a cell e, the signals have to sum up to the net flux for conservation, $\sum_{i \in e} \phi_i^e = \phi^e$. The different Fluctuation Splitting schemes proposed in the literature differ by the way cell residuals are split into signals. The schemes that may used are several and based on the different features that they present. Starting from a monotonicity preserving but first-order-accurate scheme, named N scheme, and a second-order accurate, which may lead to unphysical oscillations in the neighbourhood of a captured discontinuity, called LDA scheme; EULFS is also provided with a non-linear scheme, which captures discontinuities monotically and preserves second order of accuracy in smooth regions, that blends the linear N and LDA schemes in such a way that the former is activated only where discontinuities occur. The blend is based on a smoothness sensor that makes the new scheme non-linear.

4.2 NEO

The NEO code has been developed by Mario Ricchiuto [3] and has been mainly used to study time-dependent problems. It is based on a different formulation aimed at designing explicit Runge-Kutta residual distribution schemes exhaustively described in [3]. This explicit approach is based on three main ingredients: first recast the RD discretization as a stabilized Galerkin scheme, then use a shifted time discretization in the stabilization operator, and lastly apply high order mass lumping on the Galerkin component of the discretization. In particular, this approach turned out to be very useful in simulating unsteady flows by coupling NEO with the presented shock-fitting technique. Some of the results obtained from this work have been published in the chapter [12], whose contributions showed very promising results in the development

of the unsteady shock-fitting version. The computations shown in the chapter show the possibility of using the aforementioned linear first- (monotone) and second-order N and LDA schemes, which are based on a multidimensional upwind distribution of the cell residual, their non-linear blend (B scheme), and two non-upwind methods. In particular, the explicit predictor-corrector formulation of the second order linear Streamline-Upwind (SU) method proposed in [3] and the nonlinear blended central (Bc) discretization obtained when blending the SU method with a limited Lax-Friedrich's distribution.

5 Mesh Software

In the present version of the UnDiFi code, the Triangle mesh generator has been choosen. Since detailed information about Triangle can be found in the given link, and since no changes were done to the original code, we point the reader to the main website.

Note: future releases of the code may include several other mesh generators such as Tetgen, Yams, Delaundo.

6 Tutorials

This chapter will give a detailed overview of how to set up and run simulations with UnDiFi-2D. In particular, one steady and one unsteady testcase will be described. It assumes that some familiar with how to set the compiler options and how to compile the code. The paths to the executables are omitted. It is assumed that you either added aliases for all the executables, or the binary directories to your `$PATH` variable as described in 2.5.

The provided examples are contained in the `tests` directory. There are two types of examples:

- 2D Steady testcases:
  - CircularCylinder-1
  - CoaSHCK
  - MachReflection-1
  - MachReflection-2
  - NACA0012_M080_A0
  - NACA0012_M095_A0_FISHTAIL-1
  - Q1D
  - RegularReflection-1
  - RegularReflection-2
  - SSInteractions1-2
  - SSInteractions2-1
  - SSInteractions2-2
- 2D Unsteady testcases:
  - ShockVortex

Each testcase can run in shock-capturing and shock-fitting mode. Once all the simulation options and files are set as explained in Chapter 3, the simulation can be run by simply launching the script file **start_run_x86.sh** available in each testcase directory. The starting step, number of iterations, the gasdynamics solver to be coupled with UnDiFi-2D, the type of simulation (steady/unsteady) should be defined. Optionally, output can be redirected to a logfile.

```
#                          nbegin, nsteps, eulfs, steady, testcase,          logfile
../../bin/UnDiFi-2D-2D_x86_64 0       501     true   true    "testcase_name" | tee run.log
```

Note: in future releases of the code, the start_run_x86.sh may change.

The script **clean.sh** which removes all files and directories is also available. In the upper level directory there are two scripts to recursively run and clean all testcases, respectively:

```
./run_all_x86.sh
./clean_all_dir.sh
```

6.1 CircularCylinder-1

This testcase represents a typical blunt body problem which deals with the high speed flow past a circular cylinder at free-stream Mach number, $M_1 = 20$. The computational domain surrounds a half circular

cylinder having radius $R = 1$ (see Fig. 6.1).

6.1.1 Creation and visualization of the mesh

The mesh is created using Triangle [13] mesh generator, and it is defined by the files na00.1.node, na00.1.poly, na00.1.edge, na00.1.ele, na00.1.neigh. The files are already present in the testcase folder but in general the user shall create them according to the problem's domain.

To visualize the mesh, the following command can be run:

```
showme na00.1
```

and by selecting the appropriate tag it will show the corresponding mesh entity.

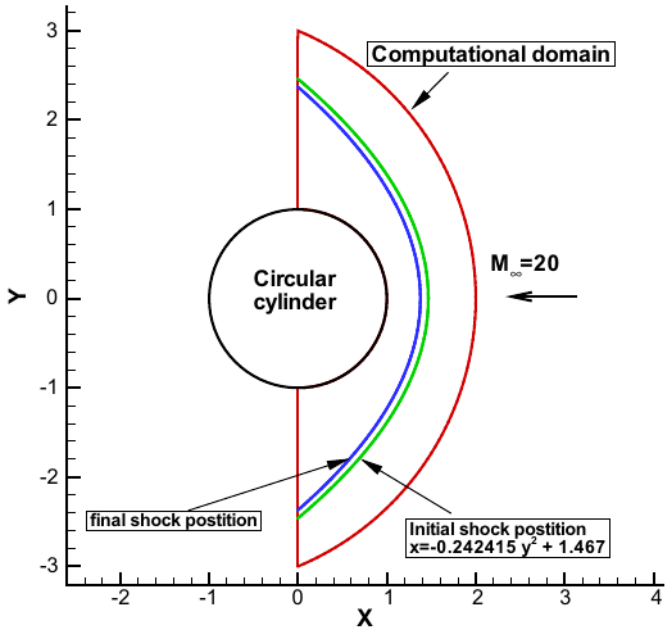

Figure 6.1: Computational domain for the CircularCylinder-1 testcase.

6.1.2 Creation of shock file

The solution computed by the unstructured code in shock-capturing mode is used to initialize the flow-field and to determine the initial position of the shock front. In the shock-fitting simulation, the initial upstream state in the shock points is set equal to the free stream conditions, while the initial downstream state is computed from the upstream state and the shock slope assuming zero shock speed ($w = 0$). The upstream and downstream shock states should be defined in the input shock file, *sh00.dat* according to a specific format as explained in the *sh00.dat* file. These values should be in agreement with those defined in the mesh files otherwise an error may be raised in the first iterations because the jump conditions which relate upstream and downstream states are violated.

6.1.3 Definition of boundary conditions

The next step consists in defining the boundary conditions for the testcase. Depending of the shock-capturing gasdynamic solver which will be used, different files are necessary, as described in Chapter 3.

6.1.4 Setting input options

The UnDiFi-2D input options are described in Chapter 3.

6.1.5 Run the simulation

Once completed all the previous steps, it is possible to run the simulation both in shock-capturing and shock-fitting mode.

6.1.5.1 How to run in shock-capturing mode?

In order to run with the NEO solver, both in shock-capturing and shock-fitting mode it is necessary to set its basic parameters.

First, move to `NEO_data` and be sure that the `output` is empty. Then, move to `NEO_data/textinput` and set the main parameters in the *inputfile-exp.txt* file, as explained in Chapter 3.

To run the simulation in shock-capturing mode, it is sufficient to modify the settings in `NEO_data/textinput/inputfile-exp.txt` as desired and directly run the shock-capturing solver from the testcase directory, by typing:

```
../../bin/CRD_euler
```

Note that the number of iterations is defined in the field "Maximum number of time steps" in the `inputfile-exp.txt` file.

To run the simulation in shock-capturing mode with EulFS, launch:

```
../../bin/EulFS_x86_64
```

6.1.5.2 How to run in shock-fitting mode?

To run the simulation in shock-fitting mode, execute the script:

```
./start_run_x86.sh
```

The flow-field and the shock position are integrated in time until steady state is reached. Fig. 6.1 shows the shock displacement which occurs between the initial position and the one reached at steady state. Fig. 6.2 displays the background mesh and the computational mesh used during the last iteration. The box on the right, which shows an enlargement of the near shock region, allows comparing the two meshes. It can be observed 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 (continuous lines) are due to the addition of the shock points and edges. Fig. 6.2 clearly shows that the re-meshing technique does not significantly increase the number of points and cells with respect to the background mesh, since the addition of the shock nodes in the computational mesh is partly balanced by the removal of the phantom nodes.

6.1.6 Visualization of the solution

Solutions are automatically saved in folders named `step#####` each `IBAK` number iterations, where `IBAK` is defined in the *input.dat* file. In particular, each folder contains:

- mesh files of Triangle

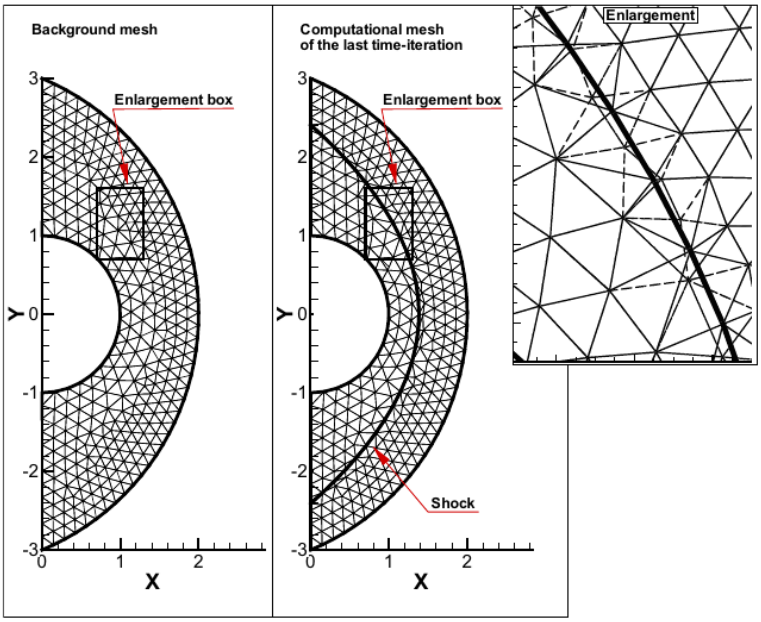

Figure 6.2: Background and computational mesh.

- na99.dat file with the shock nodes
- sh99.dat file with the updated shock points position and states
- shocknor.dat file with information about shock normals
- node velocity file, vel.dat

Depending on the gasdynamic solver adopted, the solution will be written in different files.

If EulFS is used, in each folder will be present the binary files:

- file001.dat
- file002.dat
- file003.dat

In order to visualize the solution produced by EulFS it is necessary to post-process the aforementioned files. For this purpose, the scripts *conv.sh* and *plot0.sh* are provided.

If NEO is used, there will be:

- NEO mesh file, neogrid.grd
- NEO initial solution file, vvvv_input.dat
- NEO solution file, vvvv.dat

Figure 6.3 shows the comparison between the solutions computed by the EulFS code working in shock-capturing and shock-fitting mode on a coarse mesh. Specifically, the coarse grid solution computed in shock capturing mode is characterized by a very large shock thickness and by strong spurious disturbances affecting the shock layer region. These disturbances originate from the shock and are caused by the mis-alignment between the mesh and the shock. On the contrary, the shock-fitting solution shows a very neat field. The shock-capturing solution shows the presence of spurious oscillation even if the shock thickness has been halved. The fitted solution shows a significant gain in terms of solution quality with respect the shock-capturing solution. Of course, the resolution of the shock-capturing solutions could be significantly improved if local grid-refinement was used in the shock region, but this typically involves a significant

increase in the number of cells and nodes.

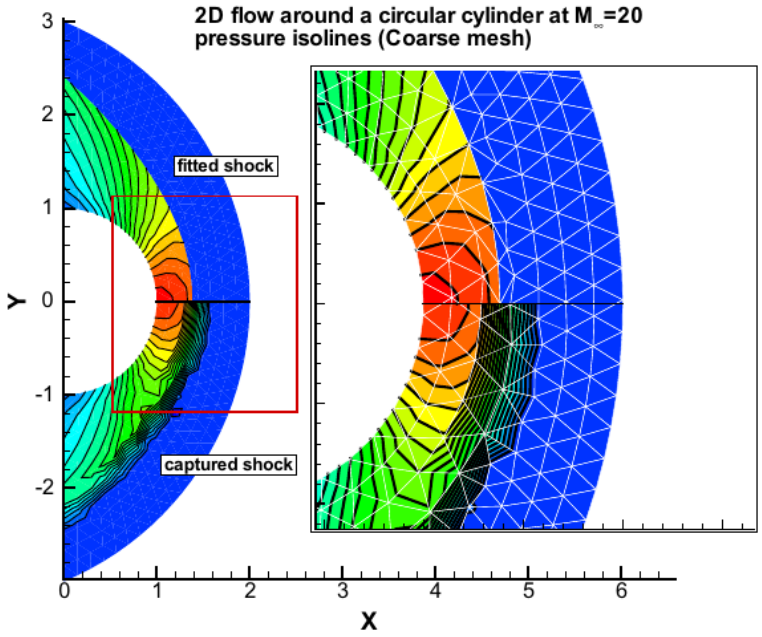

Figure 6.3: Comparison of shock-captured and shock-fitted solutions.

A quantitative analysis of the shock-fitting solutions was carried out by comparing the estimates of the shock position and the pressure distribution computed with the reference solution computed by Lyubimov and Rusanov [14]. This comparison shows that the solutions computed by the proposed shock-fitting methodology are not only in good agreement with the reference solution, but nearly superimposed. Moreover, this analysis clearly proves that the coarse grid solution is in practice grid-independent in spite of the extremely limited number of cells enclosed in the shock layer. A comparison between the shock-fitting and the shock-capturing solutions is displayed in Fig. 6.4 where the normalized pressure distributions along the line at 45° is plotted. The shock-capturing solutions are characterized by a finite shock thickness that significantly affects the distribution even on the fine grid. Moreover, visible differences between the shock-capturing solutions and the reference one are also visible in regions far from the shock and close to the body. On the contrary, the differences between the shock fitting solutions and the reference one are very small and the solution is nearly superimposed to the reference one.

6.2 Shock-Vortex interaction

This test case considers a weak shock-vortex interaction.

The interaction between a shock and a vortex has been frequently reported in the literature as a tool for understanding the mechanisms of noise generation due to the interaction between a shock-wave and a turbulent flow [15].

Figure 6.5 shows the computational domain with the boundary and the initial conditions. In particular, at the time $t = 0$ the field is characterized by the presence of a normal shock that divides the a supersonic region from a subsonic region.

The subsonic field is initialized with the uniform field that is computed from the steady R-H jump relations for a normal shock wave with a $M_s = 2$ stream in the upstream region. The initial position of the shock and of the vortex is reported in Fig. 6.5. Since the solution for this test is non-stationary and its execution

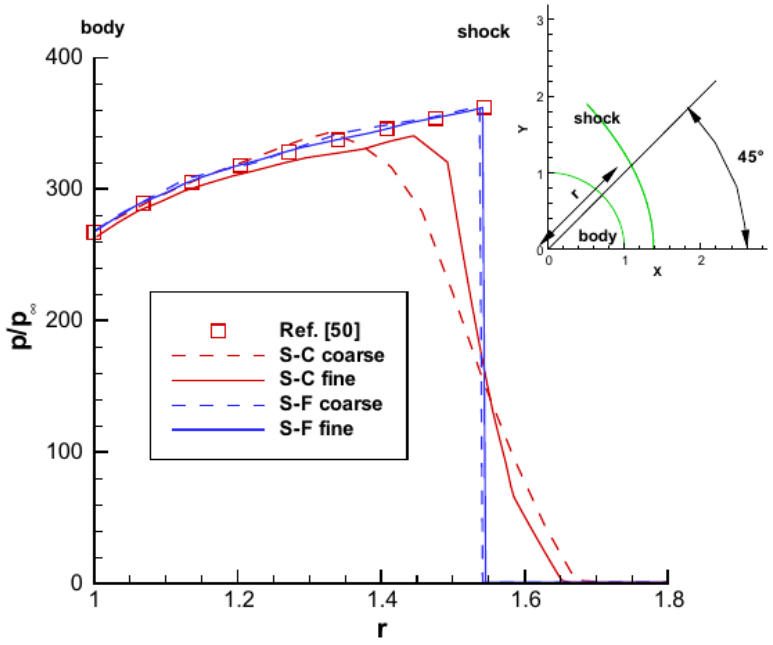

Figure 6.4: Pressure jump obtained with the captured and fitted solutions.

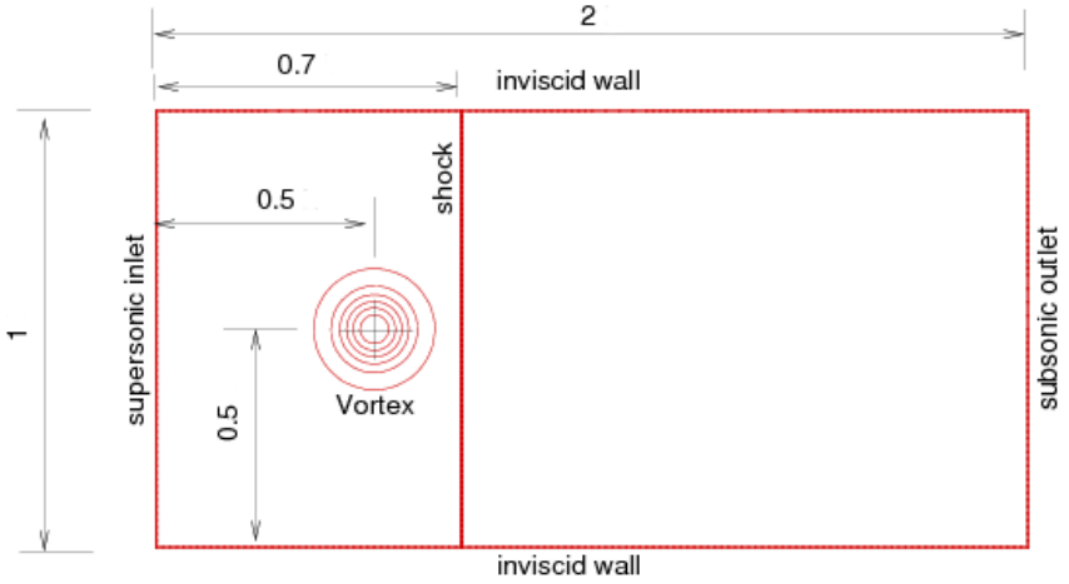

Figure 6.5: Computational domain for the ShockVortex testcase.

requires the activation of unsteady mode of the UnDiFi-2D code, this test was included to verifies all parts and subroutines of the code that ensure the time accurate computation.

The shock-capturing and shock-fitting simulations were performed using the NEO solver. The computational domain has been discretized using a Delaunay triangulation generated with Triangle. The presented results have been computed on a grid made up of 433664 elements, 217569 points with a $\triangle x = 0.00375$. A qualitative view of the solutions obtained with the two approaches is given in Fig. 6.6 and Fig. 6.7. The pictures show the total enthalpy contours in the solutions obtained with shock capturing and fitting. 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 solutions. The fitted computations, on the other hand, show very nice and smooth contours Further details about this test case and further simulations concerning the interaction between shock and vortex can be found in [16].

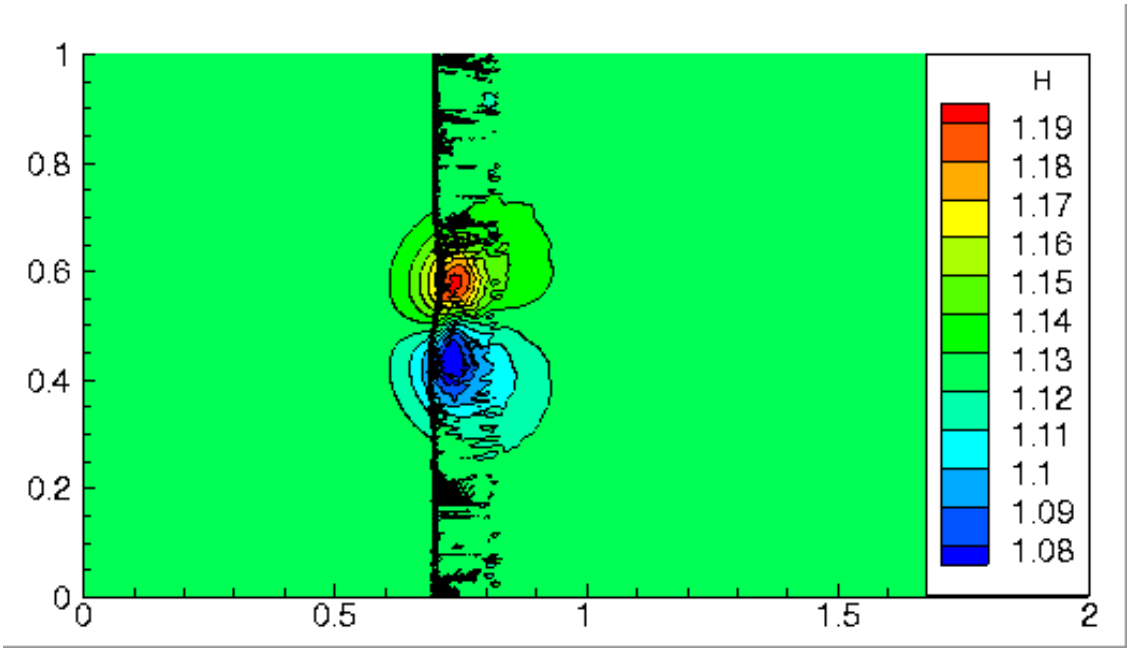

Figure 6.6: Shock-captured solution at $t = 0.3$.

All the steps to obtain these solutions are the same as those described in the previous testcase, thus, they will not be repeated. There is only one important difference respect to the steady testacases, that is, for the unsteady testacases, the *timesteps.dat* file must be present in the directory. This file contains two lines representing the dt to be used in the predictor-corrector time-accurate integration.

```
0.0008 # predictor step
0.0016 # corrector step
```

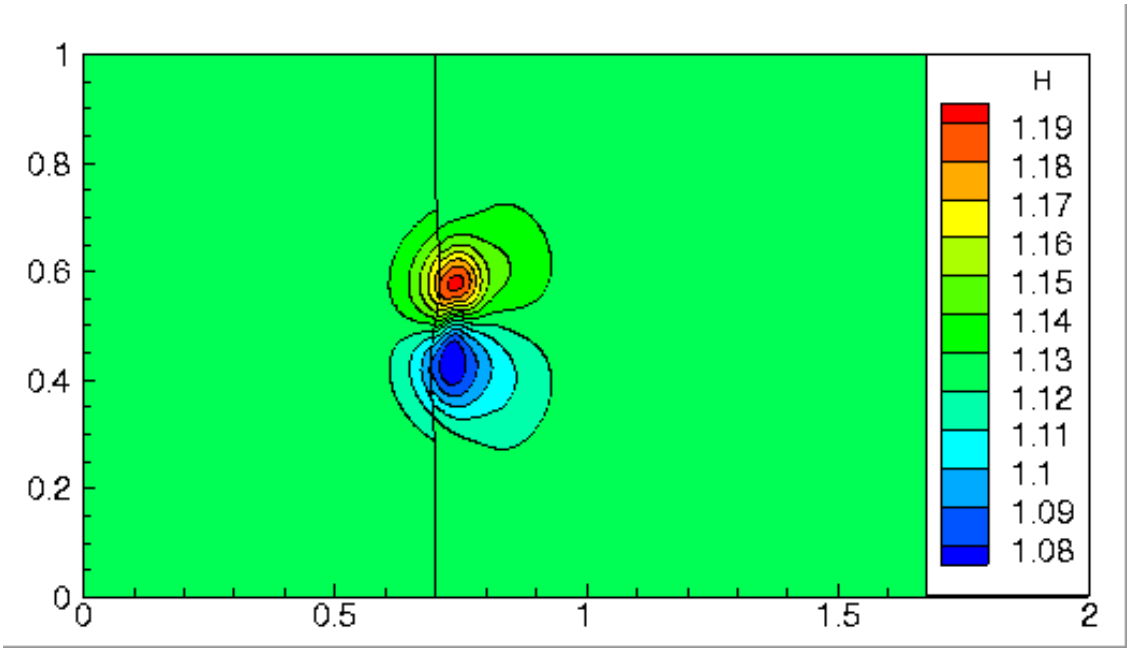

Figure 6.7: Shock-fitted solution at $t = 0.3$.

7 EulFS tutorial

This chapter will provide a guide for the user to set up and run simulations with EulFS, using both a shock capturing and a shock fitting mode.

7.1 Shock capturing mode

- **Step 1: how to convert a Triangle mesh into EulFS input files**

As stated in Chap. 3, EulFS code requires the following binary files in input:

- file001.dat
- file002.dat
- file003.dat

Containing respectively the informations linked to the nodes, the grid connectivity and the flow-field variables. They can be generated executing the **triangle2dat-NEW_$(HOSTTYPE)** program, provided in the *source_utils* directory: the user has to run this code in the test-case folder, where the Triangle mesh files (see Chap. 5 for details) are available.

```
This code converts triangle files into EulFS files

Enter fname
```

At this point, the user has to type the Triangle filename, without any extension: the code runs a check on the boundary faces number, then it asks for the presence of periodic surfaces:

```
Are there any periodic surfaces? y/n
```

If the mesh is periodic, the user enters the two (different) colours of the two periodic surfaces: it is also required to enter x/X if the corresponding nodes on the two periodic patches have the same x coordinate, otherwise enter y/Y (have the same y coordinate). If no error messages occur, the code creates the binary files *file???.dat* in the same folder of the Triangle mesh.

- **Step 2: set the runtime options for EulFS in .petscrc file**

In order to run a numerical simulation, the user has to include in this directory also the .petscrc file, which collects the runtime options for that particular test-case: all PETSc specific options are described in full details in the Users Manual. Some of the EulFS options are mandatory, other are specific to the different sets of equations the code can deal with. There are also some other options that, though necessary to run properly the code, are assigned default values whenever the user does not set them explicitly. In particular, using the runtime option -colors it is possible to assign a different boundary condition to each color, hence to each group of boundary faces. Boundary faces are usually grouped together in the meshfile by giving the same colour to faces belonging to the same boundary, e.g. body faces will be given a different colour for each different solid surface, far field will be given another colour, etc. In order to assign a specific boundary conditions it is required to fill the position° corresponding to the color of the boundary faces with the boundary condition code, as explained in the following table:

```
 Colors   Boundary condition
|  0  |   Unused
|  1  |   Supersonic inlet / Dirichlet
|  2  |   Subsonic outlet (constant static pressure)
|  3  |   Supersonic outlet / Neumann
|  4  |   Inviscid wall
|  5  |   Inviscid far field
|  6  |   Viscous wall
|  8  |   Subsonic inlet
```

° *The position must be counted after the first -1 value in the -colors line.*

The solution is saved every *ibak* iterations (see option -ibak in the .petscrc file) in *file010.dat*, which is the EulFS output file. Moreover, EulFS code is able to restart a previous numerical simulation by reading the solution from file003.dat or file010.dat: in this case, the user has to add the following line in the .petscrc file:

```
- restart   file003.dat (or file010.dat)
```

- **Step 3: how to convert EulFS output into Triangle/Tecplot files**

In order to update the solution stored in the Triangle mesh files, EulFS output must be converted into a new *.node* file, which contains the grid nodes and the flow-field variables associated with them, running **dat2triangle-NEW_$(HOSTTYPE)** in the test-case directory.

```
 This code converts the EulFS code files into triangle files

 Enter triangle file
```

The user is asked for providing a name for the output *.node* file. Then, the program reads the mesh data stored in *file001.dat-file002.dat*, the solution contained in *file010.dat* and it creates a *filename.node*, using the name entered by the user.

Post-processing of the solutions computed by EulFS into a format suitable for a visualisation is accomplished through **fsplot_$(HOSTTYPE)**. It uses the file *inp*, which should be present in the same directory, to select the EulFS solution files to be converted in a *file012.dat* readable by Tecplot: further details about the *inp* syntax can be found in Chapter 7 of the EulFS user manual.

7.2 Shock fitting mode

UnDiFi-2D code runs at each iteration all the steps described above: anyway, in order to run EulFS in a shock-fitting mode, the following options must be present in .petscrc file:

- Only an iteration must be performed:

```
- itmax 1
```

- EulFS computation must restart from file003.dat, where the solution computed in the previous iteration is stored.

```
- restart   file003.dat
```

- Shock edges, which are associated to color 10 in the computational mesh, need no particular boundary condition in EulFS: thus, the option -colors in .petscrc file must contain a 0 in the 10-th position.

```
- colors   -1,.,.,.,.,.,.,.,.,.,0,.,.,.
```

References

[1] A. Bonfiglioli, "Fluctuation splitting schemes for the compressible and incompressible euler and navier-stokes equations," *International Journal of Computational Fluid Dynamics*, vol. 14, no. 1, pp. 21–39, 2000.

[2] A. Bonfiglioli and R. Paciorri, "A mass-matrix formulation of unsteady fluctuation splitting schemes consistent with roe's parameter vector," *International Journal of Computational Fluid Dynamics*, vol. 27, nos. 4-5, pp. 210–227, 2013.

[3] M. Ricchiuto and R. Abgrall, "Explicit runge–kutta residual distribution schemes for time dependent problems: Second order case," *Journal of Computational Physics*, vol. 229, no. 16, pp. 5653–5691, 2010.

[4] M. Ricchiuto, "An explicit residual based approach for shallow water flows," *Journal of Computational Physics*, vol. 280, pp. 306–344, 2015.

[5] L. Arpaia, M. Ricchiuto, and R. Abgrall, "An ale formulation for explicit runge–kutta residual distribution," *Journal of Scientific Computing*, vol. 63, no. 2, pp. 502–547, 2015.

[6] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle, and S. Poedts, "The coolfluid framework: Design solutions for high performance object oriented scientific computing software," in *International conference on computational science*, 2005, pp. 279–286.

[7] A. Lani and others, "COOLFluiD wiki page." 2016.

[8] R. Abgrall, "Residual distribution schemes: Current status and future trends," *Computers & Fluids*, vol. 35, no. 7, pp. 641–669, 2006.

[9] H. Deconinck, H. Paillere, R. Struijs, and P. L. Roe, "Multidimensional upwind schemes based on fluctuation-splitting for systems of conservation laws," *Computational Mechanics*, vol. 11, nos. 5-6, pp. 323–340, 1993.

[10] P. Roe, *Linear advection schemes on triangular meshes*. Cranfield Institute of Technology, 1987.

[11] P. L. Roe, "Discrete models for the numerical analysis of time-dependent multidimensional gas dynamics," in *Upwind and high-resolution schemes*, Springer, 1986, pp. 451–469.

[12] L. Campoli, P. Quemar, A. Bonfiglioli, and M. Ricchiuto, "Shock-fitting and predictor-corrector explicit ale residual distribution," in *Shock fitting*, Springer, 2017, pp. 113–129.

[13] J. R. Shewchuk, "Triangle: Engineering a 2D quality mesh generator and delaunay triangulator," in *Workshop on applied computational geometry*, 1996, pp. 203–222.

[14] A. N. Lyubimov and V. V. Rusanov, *Gas flows past blunt bodies-part 1: Calculation method and flow analysis*. NASA, 1973.

[15] F. Grasso and S. Pirozzoli, "Shock-wave-vortex interactions: Shock and and vortex deformations, and sound production," *Theor. Comput. Fluid Dyn.*, vol. 13, no. 6, pp. 421–456, 2000.

[16] L. Campoli, P. Quemar, A. Bonfiglioli, and M. Ricchiuto, "Shock-fitting and predictor-corrector explicit ale residual distribution," in *Onofri*, P. and M., Ed. Shock Fitting: Classical Techniques, Recent

Developments,; Memoirs of Gino Moretti, Springer International Publishing, Berlin, 2017, pp. 113–129.